Enforcing globally dependent flow policies in message-passing systems

Li, Ximeng; Nielson, Flemming; Nielson, Hanne Riis

Published in:
Journal of Computer Languages

Link to article, DOI:
10.1016/j.cola.2019.100904

Publication date:
2019

Document Version
Peer reviewed version

Enforcing Globally Dependent Flow Policies in Message-Passing Systems

Article *in* Journal of Computer Languages · June 2019

DOI: 10.1016/j.cola.2019.100904

3 authors, including:

Ximeng Li
Capital Normal University
25 PUBLICATIONS **23** CITATIONS

Enforcing Globally Dependent Flow Policies in Message-Passing Systems

Ximeng Li[1⋆], Flemming Nielson[2], and Hanne Riis Nielson[2]

[1] Beijing Key Laboratory of Electronic System Reliability and Prognostics, Capital Normal University, China,
6642@cnu.edu.cn
[2] Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark,
{fnie|hrni}@dtu.dk

Abstract. The flow of information in a computing system is a crucial indicator for the security of the system. In a system of multiple message-passing processes, the flow of information could depend on the states of different processes. We devise a type-based verification technique for flow policies with such multi-process (global) dependencies, to provide confidentiality guarantees. In this technique, the confidentiality requirements for the presence and content of messages are dealt with separately. We develop a pair of synergetic static analyses to over-approximate the potential sets of values of the variables depended upon by the flow policies – covering global value correspondence between the variables of different processes. We significantly improve the permissiveness of security typing by exploiting information about which variables are live, and by specializing the flow policies using the conditional expressions of branching and looping constructs. We prove the soundness of our verification technique, provide a proof-of-concept implementation of it, and illustrate its effectiveness at an example system where the flow of information depends on how the headers of the messages from different processes correlate.

Keywords: Information Flow Security, Dependent Flow Policies, Security Type System, Static Analysis

1 Introduction

The flow of information in a computing system is a key factor in assessing how secure the system is. Confidentiality, for instance, requires that sensitive information may only flow to the parties that are authorized to access it. The flow of information in a system is often articulated by an *information-flow policy* [49].

Practical information-flow policies are often *content-dependent*. Content-dependent information-flow policies specify information flow to different sinks of a system with different content of data (e.g., the content of a variable or a

⋆ Part of the work was done while Ximeng Li was at the Technical University of Denmark.

communication channel). For instance, in a run of a communication protocol, a message from a party can be transmitted to different parties depending on the state of any of the parties.

In the general setting of a system with multiple components, the information-flow policy for the data of one component may depend on the data content of a different component. Still taking a protocol run for example, the information from some party, say `A`, may flow to different parties, depending on the current state of any non-`A` party. We refer to information-flow policies with such (non-local) content-dependency as globally dependent flow policies.

A considerable amount of prior effort has been poured into the enforcement of content-dependent information-flow policies [53, 8, 3, 9, 2, 31, 34, 42, 29, 28, 41, 37, 26, 10]. Among these developments, only [37] addresses the enforcement of information-flow policies with global content-dependency. This development targets shared-memory communication between the different components of a system. More concretely, the information-flow policy of a variable may depend on any variable that is shared between all the threads of a concurrent program.

In today's IT systems, the use of shared-memory communication is complemented by the use of *message-passing* communication. To name a few, prime examples of message-passing communication are: socket-based communication in networked systems, port-based communication between partitions of OS kernels [56], communicator-based communication between MPI processes [24], and rendezvous-based communication in parallel computing (e.g., [44]).

Despite the widespread use of message-passing communication in IT systems, the enforcement of content-dependent information-flow policies in message-passing systems has attracted much less attention in comparison to their memory-sharing counterpart. In particular, global content-dependency has hitherto not been addressed for message-passing systems. To enforce globally dependent flow policies in message-passing systems, the following two problems (non-existing in the shared-memory case) are to be resolved:

1. The correspondence between the variables of different processes is implicitly formed via sending and receiving messages over the communication channels that bridge the processes. Since an information-flow policy can be dependent jointly on the variables of multiple processes, it takes an explicit analysis of the inter-process correspondence of variables to find out about the targets of information disclosure as permitted by the flow policies.
2. Oftentimes, the confidentiality requirement for the *content* of a communication is different than that for the *presence* of the communication. Hence, the presence and content of communication are to be considered as separate aspects in formulating and enforcing content-dependent information-flow policies in message-passing systems (see, e.g., [48, 46, 27] for the non-content-dependent case).

In this paper, we address the problem of verifying information-flow security under globally dependent flow policies, in systems with message-passing processes. Our solution consists of a security type system [54] and two static analyses. The security type system is aided by the two static analyses to provide

confidentiality guarantees wrt. a formal security property [32]. The two static analyses are:

1. a global analysis about the values of variables, and the most recent branching decisions made, when reaching different program points, and
2. a local analysis providing the association of the values of variables in each process and the most recent branching decisions made by the same process.

When combined, the two analyses provide over-approximative information about the values of variables (i.e., which values the variables *may* have) with high precision. This information helps the security type system evaluate the targets of information disclosure as permitted by the content-dependent flow policies, and thereby increases the precision of the security type system.

In security type systems for dependent flow policies, the security types of variables are dependent on their values. This dependency could cause changes of security types due to value changes. Such type changes could render a system without information leakage un-typable. We develop two methods to mitigate this effect. Firstly, we design our security type system such that only the changes in the security types of live variables can affect the typability of a message-passing system. Secondly, for a security type that specifies different sets of information-flow destinations depending on a conditional expression in the program, we specialize the type to a concrete set of destinations according to the evaluation result of the conditional. Hence, changes of the type inside a conditional branch or a loop body are avoided by the removal of content dependency in the scope of the conditional branch or the loop. Both methods improve the permissiveness of the security type system without affecting its soundness.

To summarize, the following technical contributions are made in this article:

1. a sound information-flow type system for enforcing globally dependent flow policies in message-passing systems,
2. the utilization of liveness information and the scoped specialization of variable types that result in substantial permissiveness gain in security typing,
3. the combination of a global analysis and a local analysis that provides information about the values of variables with high precision, in support of security typing, and
4. soundness proofs for the security type system and the static analyses.

Accompanying our technical development, we show how the building blocks of our solution play together, to prove the information-flow security of a message-passing system, where two parties communicate under the regulation of a stateful packet filter. In this system, the flow of information depends on how the headers of the most recent messages from the two parties correlate. We provide a proof-of-concept implementation[3] of our solution in the OCaml language [1], using the SMT solver Z3 to solve the value constraints [14].

[3] The implementation is accessible at `https://github.com/lixm/gdif_checker`

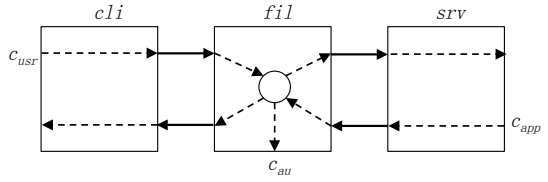

Fig. 1. The Scenario of Stateful Packet Filtering

Structure. This article is structured as follows. In Section 2, we outline the key problems to be addressed in enforcing globally dependent flow policies in message passing systems. In Section 3, we formally define the computation model of message passing systems. In Section 4, we formally define globally dependent flow policies. In Section 5, we present the security type system for enforcing globally dependent flow policies in message-passing systems. In Section 6, we formally define the security property enforced by the security type system, and present the soundness theorem of the type system (wrt. the static analyses in Section 7). In Section 7, we present the global and local static analyses supporting the security type system. In Section 8, we discuss related work in detail. In Section 9, we conclude our work. In Appendix A, we explain our use of notation in detail, and provide the formal definitions that are omitted from the main text. In Appendix B and Appendix C, we present proofs of our theoretical results.

2 Motivating Scenario

To explicate the challenges in enforcing globally dependent flow policies in message-passing systems, we consider a concrete scenario of stateful packet filtering.

As illustrated in Fig. 1, a client module cli communicates with a server module srv under the regulation of a filter module fil. In the filter module, each message from the client (originally received from the user via the communication channel c_{usr}) is checked against the current state of the filter. If the check is passed, then the message is forwarded as is to the server side. Otherwise, information about the failed check on the message is communicated to an auditing module au. The same happens to each message from the server module (originally received from a server-side application via the communication channel c_{app}) to the client.

To make things more concrete, we consider the case where the check at the filter ensures a numbering scheme on the headers of the messages between the client and the server. We capture this numbering scheme by the formula $\phi(h_1, h_2)$, where h_1 and h_2 are the headers of the messages received from c_{usr} and from c_{app}, respectively. That is, information about the payload of each message from the client is revealed to the server-side application *if* $\phi(h_1, h_2)$ holds, *and* is revealed to the auditing module *if* $\phi(h_1, h_2)$ does not hold. We formally express this statement as

$$(\phi(h_1, h_2) \triangleright \{\mathsf{app}\}) \curlywedge (\neg\phi(h_1, h_2) \triangleright \{\mathsf{au}\}) \tag{1}$$

The aforementioned statement is an information-flow policy. The symbols $\curlywedge$ and $\triangleright$ can be understood as conjunction and implication, except that they are

used to form a policy, rather than a logical formula. The policy explains how the flow of information is regulated in the system. It also mandates that the user's information is not always revealed to the auditing module – this happens only when the numbering of messages goes wrong. In this policy, the targets of information disclosure are dependent on the content of the variables h_1 and h_2. These two variables belong to two different components of the system. Hence, this content-dependency is global.

To verify that the disclosure of the payload over the channel c_{usr} indeed follows the policy, the following two problems have to be addressed:

1. To obtain whether information may be disclosed to the application or the auditing module, it has to be found out when $\phi(h_1, h_2)$ holds and when it does not hold. Hence, the correspondence between the variables of different components (such as h_1 and h_2) is to be identified. This correspondence is established at runtime via the communication between the different components. The correspondence has to be analyzed statically, to support static information-flow verification.
2. The presence of communication from the client to the filter is easily revealed to both the server-side application and the auditing unit – if this communication does not happen, subsequent ones (whether to the application or to the auditing unit) will not happen either. For the policy (1), the information subject to protection (wrt. confidentiality) is the content of the messages from the client. The leakage of their presence does not harm the protection of sensitive information possessed by the user. Hence, the content of communication has to be considered separately from its presence.

The two problems above are specific to the enforcement of globally dependent flow policies in message-passing systems (as opposed to shared-memory systems). In this article, we propose a solution of both problems, that consists of a security type system and two static analyses.

Scoped Specialization of Security Types. For the example scenario, messages from the client to the server are temporarily stored in the program variables of the filter module. Correspondingly, the flow policy (1) is used as the security types for the relevant variables. In our security type system, this security type is specialized into $\{\mathsf{app}\}$ throughout the scope where $\phi(h_1, h_2)$ holds in the filter module, and into $\{\mathsf{au}\}$ throughout the scope where $\phi(h_1, h_2)$ does not hold. Here, $\{\mathsf{app}\}$ constantly restricts information flow to the server-side application only, while $\{\mathsf{au}\}$ constantly restricts information flow to the auditing unit only. We term this feature of specializing security types throughout complete scopes the *scoped specialization* of security types. For this example, scoped specialization removes the dependencies of the security type (1) on h_1 and h_2 for continuous code regions. Hence, updates of h_1 and h_2 in these code regions cannot change the actual security types on the data sent by the client from $\{\mathsf{app}\}$ to $\{\mathsf{au}\}$, or in the other direction (such changes are undesirable since they would violate the restriction on information flow imposed by the original security type). This is

an advantage over the more traditional approach of specializing security types only at individual actions such as assignments (e.g., [29, 37, 26]).

Before discussing our security type system and its supportive static analyses, we formally define the model of message-passing systems (Section 3) and globally dependent flow policies (Section 4).

3 Computation in a Message-Passing System

We consider systems each consisting of processes communicating with each other and with the environment of the system through synchronous message passing.

3.1 Structure of a System

Each process executes a *statement* under a *principal*. We denote the set of statements by Stmt. We denote the set of principals by Pr. Each process has a (local) *memory*. We denote the set of memories by $\mathit{Mem} = \mathit{Var} \to \mathit{Val}$. Here, Var is the set of *variables*, and Val is the set of *values*.

We model the communication lines of the processes of a system as *polyadic channels*. The set of polyadic channels is PCh. Each polyadic channel has a set of indexed *channel components*. For a polyadic channel c, the number of channel components of c is denoted by $|c|$, and the i-th component of c is denoted by $c.i$. We denote the set of all channel components by $\mathit{Ch} = \{c.i \mid c \in \mathit{PCh} \wedge i \in \{1, \ldots, |c|\}\}$. We model the set of communication lines that can be used to communicate with outside the system by the set EPCh of *external communication channels*. We denote the set $\mathit{PCh} \setminus \mathit{EPCh}$ by IPCh, which comprises the *internal communication channels*.

We model the state of a system by *a global configuration*. The set of global configurations is $\mathit{GCnf} = (\mathit{Pr} \times \mathit{Stmt} \times \mathit{Mem})^*$. Hence, a global configuration is a list of triples, each consisting of a principal, a statement, and a memory. Given a global configuration $\langle\langle p_1, S_1, m_1\rangle, \ldots, \langle p_n, S_n, m_n\rangle\rangle$, the state of the i-th process is modeled by $\langle p_i, S_i, m_i\rangle$. We call this triple a *local configuration*. The set of local configurations is $\mathit{LCnf} = \mathit{Pr} \times \mathit{Stmt} \times \mathit{Mem}$.

3.2 Execution of a System

We distinguish between different kinds of execution steps performed by a system by considering a set of *actions* Act. This set contains actions α of the form $c!\vec{v}$, representing an output of the list $\vec{v}$ of values over the polyadic channel c, actions of the form $c?\vec{v}$, representing an input of the list $\vec{v}$ of values over the polyadic channel c, and actions of the form τ, representing an internal communication between two processes in the system. Further actions in the set Act are to be introduced later in this section.

We model the execution of a system by a *trace*. The set of traces is $\mathit{Tr} = \mathit{GCnf}\,(\mathit{Act} \times \mathit{GCnf})^*$. A trace is thus a non-empty alternating list of global configurations and actions, that starts and ends with global configurations.

$$\frac{last(tr) = \langle \ldots, lcnf_1, \ldots, lcnf_2, \ldots \rangle \quad lcnf_1 \xrightarrow{c!\vec{v}} lcnf_1' \quad lcnf_2 \xrightarrow{c?\vec{v}} lcnf_2'}{tr \rightarrow_\xi tr.\tau.\langle \ldots, lcnf_1', \ldots, lcnf_2', \ldots \rangle} \text{ if } \xi(tr) = (prin\text{-}of(lcnf_1), prin\text{-}of(lcnf_2))$$

$$\frac{last(tr) = \langle \ldots, lcnf_1, \ldots, lcnf_2, \ldots \rangle \quad lcnf_1 \xrightarrow{c?\vec{v}} lcnf_1' \quad lcnf_2 \xrightarrow{c!\vec{v}} lcnf_2'}{tr \rightarrow_\xi tr.\tau.\langle \ldots, lcnf_1', \ldots, lcnf_2', \ldots \rangle} \text{ if } \xi(tr) = (prin\text{-}of(lcnf_1), prin\text{-}of(lcnf_2))$$

$$\frac{last(tr) = \langle \ldots, lcnf, \ldots \rangle \quad lcnf \xrightarrow{\alpha} lcnf'}{tr \rightarrow_\xi tr.\alpha.\langle \ldots, lcnf', \ldots \rangle} \text{ if } \exists ci : \xi(tr) = (prin\text{-}of(lcnf), ci) \wedge match(\alpha, ci)$$

$$\frac{\neg(\exists gcnf, \alpha : \alpha \neq \diamond \wedge tr \rightarrow_\xi tr.\alpha.gcnf)}{tr \rightarrow_\xi tr.\diamond.last(tr)}$$

Fig. 2. The Calculus Rules for the Judgment $tr \rightarrow_\xi tr'$

We model the environment of a system by a *strategy* [55]. The set of strategies is $\Xi = Tr \rightarrow II$. Here, $II = ((Pr \times CI) \cup (Pr \times Pr))$ is the set of *interaction intents* of the environment, and $CI = \{c!\vec{v}, c?\cdot \mid c \in EPCh \wedge \vec{v} \in Val^*\}$ is the set of *communication intents* of the environment. More specifically, a $c!\vec{v}$ models an attempt of the environment to output the list $\vec{v}$ of values to the system over c, while a $c?\cdot$ models an attempt of the environment to input from the system over c. For a strategy ξ and a trace tr, the case where $\xi(tr)$ is a principal paired with a communication intent models the situation where the environment schedules the process of principal p for execution, attempting to communicate with the system according to the communication intent. The case where $\xi(tr)$ is a pair of principals models the situation where the environment schedules the processes of the principals for internal communication.

We model a single execution step of the system by the judgment $tr \rightarrow_\xi tr'$. Intuitively, the trace tr is extended after a step of the system into the trace tr', under the strategy ξ. We specify the calculus rules for this judgment in Fig. 2 In these rules, instances of the judgment $lcnf \xrightarrow{\alpha} lcnf'$ are used. This latter judgment models a single execution step of a process with local configuration $lcnf$, resulting in the local configuration $lcnf'$, performing the action α.

Intuitively, if two scheduled processes can perform an output and an input, respectively, over the same channel, then the two processes can communicate internally with each other (first and second rules). If a scheduled process locally performs an action α, and α matches the communication intent of the environment, then the system performs the same action, and the local configuration of the process is updated correspondingly (third rule). The predicate $match : Act \times CI \rightarrow \{tt, ff\}$ is defined as

$$\begin{aligned} match(\alpha, ci) \triangleq\ & \exists c \in EPCh : \exists \vec{v} \in Val^* : \alpha = c!\vec{v} \wedge ci = c?\cdot \vee \\ & \exists c \in EPCh : \exists \vec{v} \in Val^* : \alpha = c?\vec{v} \wedge ci = c!\vec{v} \vee \\ & \alpha \notin \{c!\vec{v}, c?\vec{v} \mid c \in PCh \wedge \vec{v} \in Val^*\} \end{aligned}$$

$$\langle p, {}^{\imath}\mathsf{skip}, m\rangle \xrightarrow{\tau} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle$$

$$\langle p, x :=^{\imath} e, m\rangle \xrightarrow{\tau} \langle p, {}^{\iota_f}\mathsf{stop}, m[x \mapsto [\![e]\!]_m]\rangle$$

$$\frac{\langle p, S_1, m\rangle \xrightarrow{\alpha} \langle p, S_1', m'\rangle}{\langle p, S_1; S_2, m\rangle \xrightarrow{\alpha} \langle p, S_1'; S_2, m'\rangle} \text{ if } S_1' \neq {}^{\iota_f}\mathsf{stop} \qquad \frac{\langle p, S_1, m\rangle \xrightarrow{\alpha} \langle p, {}^{\iota_f}\mathsf{stop}, m'\rangle}{\langle p, S_1; S_2, m\rangle \xrightarrow{\alpha} \langle p, S_2, m'\rangle}$$

$$\langle p, {}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, m\rangle \xrightarrow{\mathsf{brt}^{p,\imath}} \langle p, S_1, m\rangle \quad \text{if } [\![e]\!]_m = tt$$

$$\langle p, {}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, m\rangle \xrightarrow{\mathsf{brf}^{p,\imath}} \langle p, S_2, m\rangle \quad \text{if } [\![e]\!]_m = ff$$

$$\langle p, {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brt}^{p,\imath}} \langle p, S; {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, m\rangle \quad \text{if } [\![e]\!]_m = tt$$

$$\langle p, {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brf}^{p,\imath}} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle \quad \text{if } [\![e]\!]_m = ff$$

$$\langle p, {}^{\imath}\mathsf{send}(c, \vec{e}), m\rangle \xrightarrow{c!\vec{v}} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle \quad \text{if } [\![\vec{e}]\!]_m = \vec{v}$$

$$\langle p, {}^{\imath}\mathsf{recv}(c, \vec{x}), m\rangle \xrightarrow{c?\vec{v}} \langle p, {}^{\iota_f}\mathsf{stop}, m[(x_j \mapsto v_j)_j]\rangle$$

Fig. 3. The Semantics of Processes

If the system can perform no ordinary action, then the system can take an empty step as represented by $\diamond$ (last rule).

We model the program code executed by a system by a *concurrent statement* $CS = p_1 : S_1 || \ldots || p_n : S_n$. The set of traces of a system executing this concurrent statement is

$$\mathit{traces\text{-}of}_\xi(CS) = \{tr \in Tr \mid \langle\langle p_1, S_1, m_\star\rangle, \ldots, \langle p_n, S_n, m_\star\rangle\rangle \rightarrow_\xi^* tr\}$$

Here, $m_\star = \lambda x.0$ is the initial memory mapping all variables to 0.

3.3 Programming Language

We introduce the syntax and semantics of our programming language for the statements of a message-passing system. This is a While language augmented with synchronous communication.

We denote the set of expressions by *Exp*. We denote the set of program points by *Pt*. We specify the statements in the set *Stmt* by the following syntax, where $\vec{e} \in Exp^*$, $\vec{x} \in Var^*$, and $\imath \in Pt$.

$$S ::= {}^{\imath}\mathsf{skip} \mid x :=^{\imath} e \mid {}^{\imath}\mathsf{send}(c, \vec{e}) \mid {}^{\imath}\mathsf{recv}(c, \vec{x}) \mid S; S \mid$$
$${}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ S\ \mathsf{else}\ S\ \mathsf{fi} \mid {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od} \mid {}^{\imath}\mathsf{stop}$$

The derivability of the judgment $lcnf \xrightarrow{\alpha} lcnf'$ is specified by a small-step semantics [43] (with semantic rules given in Fig. 3). The statement ${}^{\imath}\mathsf{skip}$ terminates without effects, the statement $x :=^{\imath} e$ assigns the value of e to x, the statement ${}^{\imath}\mathsf{send}(c, \vec{e})$ sends the list of values resulting from the evaluation of $\vec{e}$ over c, the statement ${}^{\imath}\mathsf{recv}(c, \vec{x})$ receives the next list of values over c, the statement ${}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$ branches to either S_1 or S_2 depending on the evaluation result of e, the statement ${}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}$ enters the statement S or terminates depending on the evaluation result of e, and the statement ${}^{\imath}\mathsf{stop}$ is

an indication of termination. Note that $^{\imath}\mathsf{stop}$ is not in the surface syntax. It is introduced to avoid having both triples and pairs for the local configurations.

We completely specify the set Act of actions by the following syntax.

$$\alpha ::= c!\vec{v} \mid c?\vec{v} \mid \tau \mid \diamond \mid \mathsf{brt}^{p,\imath} \mid \mathsf{brf}^{p,\imath}$$

The first four actions have already been introduced. The action $\mathsf{brt}^{p,\imath}$ represents a conditional branching to the true branch at the program point $\imath$ in the statement of the principal p. The action $\mathsf{brf}^{p,\imath}$ represents a conditional branching to the false branch at the program point $\imath$ in the statement of the principal p. These two actions for branching are used to formulate the correctness conditions for our static analyses supporting the security type system. On the other hand, no action for joining control flow branches is needed.

In Fig. 3, the communication and branching actions of appropriate types are specified for the execution of different language constructs. The ι_{f} represents the *final program point* of the statement executed. We assume ι_{f} is different from any program point explicitly annotated in the statement, and that ι_{f} may only be used to annotate a stop.

3.4 Program Graphs

We model the possible executions of a concurrent statement with synchronization between different processes by a program graph.

The vertices of a program graph record the potentially reachable program points of the constituent sequential statements of the concurrent statement, and the edges of the program graph record the actions of the sequential statements.

Formally, the program graph of a concurrent statement CS is a pair (V_{CS}, E_{CS}), where V_{CS} is the set of nodes, and E_{CS} is the set of edges. Here, $V_{CS} \subseteq Pt^{|CS|}$, and $E_{CS} \subseteq V_{CS} \times \mathcal{P}(\mathbb{N} \times SAct) \times V_{CS}$. In particular, $SAct$ is the set of *syntactical actions* whose elements are given by

$$sa ::= \mathsf{sk} \mid x \leftarrow e \mid c!\vec{e} \mid c?\vec{x} \mid \mathsf{brt} \mid \mathsf{brf}$$

In the above, sk represents the action of a skip, $x \leftarrow e$ represents an assignment of the expression e to the variable x, $c!\vec{e}$ represents an output of the list $\vec{e}$ of expressions over the polyadic channel c, $c?\vec{x}$ represents an input from the polyadic channel c into the list $\vec{x}$ of variables, and brt and brf represent true and false branching decisions, respectively. For a syntactical action sa, we write $upd(sa)$ for the set of variables (syntactically) updated in sa. That is, $upd(x \leftarrow e) \triangleq \{x\}$, $upd(c?\vec{x}) \triangleq \{\vec{x}\}$, and $upd(sa) \triangleq \emptyset$ otherwise.

We define (V_{CS}, E_{CS}) as the least pair satisfying the set of inference rules in Fig. 4. The first rule says that the combination of the initial program points of the various processes are always jointly reachable. Hence, the list of these initial program points constitute a node of the program graph. The function $fst : Stmt \to Pt$ gives the initial program point of each statement. This function is formally defined in Appendix A. The second rule deals with synchronous communication inside the system. More concretely, if the i-th process takes one step from the program point $\imath_i$ to the program point $\imath_i'$ with the syntactical

$$\frac{}{fst(CS(1))...fst(CS(n)) \in V_{CS}} \text{ where } n = |CS|$$

$$\frac{\vec{\imath} \in V_{CS} \quad (\imath_i, c!\vec{e}, \imath'_i) \in succ(CS[i], \iota_{\mathrm{f}}) \quad (\imath_j, c?\vec{x}, \imath'_j) \in succ(CS[j], \iota_{\mathrm{f}})}{\vec{\imath}[i \mapsto \imath'_i][j \mapsto \imath'_j] \in V_{CS} \quad (\vec{\imath}, \{(i, c!\vec{e}), (j, c?\vec{x})\}, \vec{\imath}[i \mapsto \imath'_i][j \mapsto \imath'_j]) \in E_{CS}}$$

$$\frac{\vec{\imath} \in V_{CS} \quad c \in EPCh \quad (\imath_i, c!\vec{e}, \imath'_i) \in succ(CS(i), \iota_{\mathrm{f}})}{\vec{\imath}[i \mapsto \imath'_i] \in V_{CS} \quad (\vec{\imath}, \{(i, c!\vec{e})\}, \vec{\imath}[i \mapsto \imath'_i]) \in E_{CS}}$$

$$\frac{\vec{\imath} \in V_{CS} \quad c \in EPCh \quad (\imath_i, c?\vec{x}, \imath'_i) \in succ(CS(i), \iota_{\mathrm{f}})}{\vec{\imath}[i \mapsto \imath'_i] \in V_{CS} \quad (\vec{\imath}, \{(i, c?\vec{x})\}, \vec{\imath}[i \mapsto \imath'_i]) \in E_{CS}}$$

$$\frac{\vec{\imath} \in V_{CS} \quad (\imath_i, sa, \imath'_i) \in succ(CS(i), \iota_{\mathrm{f}}) \quad sa \notin Output \cup Input}{\vec{\imath}[i \mapsto \imath'_i] \in V_{CS} \quad (\vec{\imath}, \{(i, sa)\}, \vec{\imath}[i \mapsto \imath'_i]) \in E_{CS}}$$

Fig. 4. The Inference Rules for the Derivation of Program Graphs

action $c!\vec{e}$, and the j-th process takes one step from the program point $\imath_j$ to the program point $\imath'_j$ with the syntactical action $c?\vec{x}$, then the system of the concurrent statement CS takes one step, changing (only) the program points of the i-th and the j-th processes. On the inferred edge, syntactical actions of the i-th process and the j-th process are recorded, respectively. The function *succ* used in the premises of the rule is formally defined in Appendix A. The next two rules concerns communication with the environment. The last rule describes the case where one single process may perform an internal computation step.

4 Flow Policies with Global Content-Dependency

Information enters and leaves a system through the communication channels of the system. These communication channels are subject to protection under content-dependent information-flow policies. The syntax for these information-flow policies is

$$\begin{aligned} P &::= \{p_1, ..., p_n\} \mid \phi \triangleright P \mid P \curlywedge P \\ \phi &::= \mathsf{true} \mid t \ cop \ t \mid \neg\phi \mid \phi \wedge \phi \\ t &::= n \mid x@p \mid c.j \mid g(t, ..., t) \end{aligned}$$

A policy P can be a set of principals to whom the disclosure of information is restricted, an implicative policy $\phi \triangleright P$ that imposes the policy P only when the logical formula ϕ holds, or a conjunctive policy $P_1 \curlywedge P_2$ that jointly imposes the policies P_1 and P_2. We shall use $\star$ as shorthand notation for the policy that is the set of all principals. In a policy, the formula ϕ can be the formula true for logical truth, the comparison of two terms, the negation of a formula, or the conjunction of two formulas. A term t can be a numeral, a variable of (the statement of) a principal $x@p$, a channel component $c.j$ or a function application on terms.

We use the restricted syntax $P ::= \{p_1, \ldots, p_n\}$ for the flow policies governing the presence of communication over the polyadic channels, while the full syntax

$$[\![\{p_1,\ldots,p_2\}]\!]_{gcnf,\delta} \triangleq \{p_1,\ldots,p_2\}$$

$$[\![\phi \triangleright P]\!]_{gcnf,\delta} \triangleq \begin{cases} [\![P]\!]_{gcnf,\delta} \text{ if } [\![\phi]\!]_{gcnf,\delta} = tt \\ Pr \qquad \text{otherwise} \end{cases}$$

$$[\![P_1 \curlywedge P_2]\!]_{gcnf,\delta} \triangleq [\![P_1]\!]_{gcnf,\delta} \cap [\![P_2]\!]_{gcnf,\delta}$$

$$\ldots$$

$$[\![x@p_j]\!]_{\langle\ldots,\langle p_j,S,m\rangle,\ldots\rangle,\delta} \triangleq m(x)$$

$$[\![c.j]\!]_{gcnf,\delta} \triangleq \delta(c.j)$$

$$\ldots$$

Fig. 5. The Semantics of Content-Dependent Information-Flow Policies

is permitted for the flow policies governing the content of communication over the channel components. Hence, content dependency (that is potentially global) is permitted for the flow policies on the content of communication. Furthermore, the flow policies of different channel components of a polyadic channel may admit different content dependency.

The semantics of a policy is the set of principals to which information may be disclosed according to the policy. This set of principals results from interpreting the policy under a global configuration *gcnf* and a partial function $\delta \in \mathit{Ch} \rightharpoonup \mathit{Val}$. Here, *gcnf* carries information about the contents of variables, and δ carries information about the contents communicated over polyadic channels.

The key part of the definition on the interpretation of policies is given in Fig. 5 (with the complete definition given in Fig. 12 of Appendix A). The definition meets the intuitive meaning of policies given when introducing them. The interpretation of formulas ϕ is omitted, while the interpretation of terms t is partially presented. The interpretation of a variable of (the process of) a principal gives the value of the variable in the memory of that principal. The interpretation of a channel component gives the value of the channel component according to the partial function δ.

Let *Pol* be the set of content-dependent information-flow policies. We introduce a relation $\sqsubseteq \subseteq \mathit{Pol} \times \mathit{Pol}$ to capture the relative levels of confidentiality required by different policies.

Definition 1. $P_1 \sqsubseteq P_2 \triangleq \forall \mathit{gcnf}, \delta : [\![P_1]\!]_{gcnf,\delta} \supseteq [\![P_2]\!]_{gcnf,\delta}$

Intuitively, $P_1 \sqsubseteq P_2$ says that less principals are allowed to (directly or indirectly) access information under P_2 than under P_1. Hence, P_2 requires a higher level of confidentiality than P_1 does.

In an information-flow analysis, policies are compared syntactically. For two policies of the form $\curlywedge_i(\phi_i \triangleright R_i)$ where $R_i \in \mathcal{P}(Pr)$ for each i (the so-called Implication Normal Form, or INF), we define the relation $\preceq_{\text{INF}} \in \mathit{Pol} \times \mathit{Pol}$.

Definition 2. $\curlywedge_i(\phi_i \triangleright R_i) \preceq_{\text{INF}} \curlywedge_j(\phi'_j \triangleright R'_j)$ iff $\forall i : \forall p \notin R_i : \exists j : \phi_i \Rightarrow \phi'_j \wedge p \notin R'_j$

For policies in INF, it is not difficult to show that the relation $\preceq_{\text{INF}}$ is sound wrt. the semantic ordering $\sqsubseteq$.

We define the function *inf* that gives the implication normal form of each given policy in Fig. 6.

$$
inf(P) \triangleq \begin{cases} P & \text{if } P \text{ is in INF} \\ \mathsf{true} \triangleright \{p_1, \ldots, p_n\} & \text{if } P = \{p_1, \ldots, p_n\} \\ \curlywedge_{1 \le j \le n} (\phi \wedge \phi_j \triangleright R_j) & \text{if } \exists P' : P = (\phi \triangleright P') \wedge \\ & \qquad inf(P') = \curlywedge_{1 \le j \le n} (\phi_j \triangleright R_j) \\ (\phi_{11} \triangleright R_{11}) \curlywedge \ldots \curlywedge (\phi_{1m} \triangleright R_{1m}) & \text{if } \exists P_1, P_2 : P = P_1 \curlywedge P_2 \wedge \\ \curlywedge (\phi_{21} \triangleright R_{21}) \curlywedge \ldots \curlywedge (\phi_{2n} \triangleright R_{2n}) & \qquad inf(P_1) = \curlywedge_{1 \le j \le m} (\phi_{1j} \triangleright R_{1j}) \\ & \qquad inf(P_2) = \curlywedge_{1 \le j \le n} (\phi_{2j} \triangleright R_{2j}) \end{cases}
$$

Fig. 6. The Definition of the Function *inf*

Lemma 1. *For all policies P, $inf(P)$ is well-defined and is in the implication normal form. Furthermore, it holds that $\forall gcnf, \delta : [\![P]\!]_{gcnf,\delta} = [\![inf(P)]\!]_{gcnf,\delta}$.*

To syntactically compare two arbitrary policies, we define the relation $\preceq \subseteq Pol \times Pol$.

Definition 3. $P_1 \preceq P_2$ iff $inf(P_1) \preceq_{\text{INF}} inf(P_2)$

The relation $\preceq$ is sound wrt. the semantic ordering $\sqsubseteq$.

Lemma 2. *If $P_1 \preceq P_2$, then $P_1 \sqsubseteq P_2$.*

Thus, we have a sound way to syntactically check the semantic ordering between arbitrary globally dependent flow policies via transformation of policies into the Implication Normal Form. In the next section, we develop a type-based information-flow analysis where the security types take the form of the flow policies. The way of comparing flow policies developed in the above applies directly to the comparison of these security types.

5 Security Type System

We devise an information-flow type system that can be used to statically verify the security of a system under globally-dependent information-flow policies. Three kinds of information leakage are detected by the security type system: explicit leakage, implicit leakage, and internal timing leakage [51]. Explicit leakage is information leakage caused by assigning confidential data directly to a public sink. Implicit leakage is information leakage caused by reaching an assignment to a public sink depending on confidential information. Internal timing leakage is information leakage caused by different timing of reaching multiple assignments to the same public sink in concurrent processes. For a system with message-passing communication, information can be leaked through not only assignments, but also communications to public sinks.

5.1 Environments for the Security Type System

Environments for Security Types. Our security type system is used to enforce end-to-end information-flow security under flow policies for the polyadic communication channels. All channels and variables are given security types. The security types for a channel include the security type for the presence of communication over the channel, and the security types for the content of communication over the components of the channel. Furthermore, the security type for the presence of communication over a channel is the same as the flow policy for the presence of communication over the channel, while the security type for the content of communication over each channel component is the same as the flow policy for the content of communication over the channel component. On the other hand, the security types for variables are a subset of the flow policies (c.f. Section 4) that contain all the flow policies without dependency on channel components – i.e., the security types for variables may only depend on variables. The security types for variables are used to enforce the flow policies on channels.

Formally, we introduce the *channel type environment* $\mathcal{C} : (PCh \cup Ch) \to Pol$. For each $c \in PCh$, $\mathcal{C}(c)$ is the security type for the presence of communication over c. For each $c \in PCh$ and $j \in \{1, \ldots, |c|\}$, $\mathcal{C}(c.j)$ is the security type for the content communicated over the channel component $c.j$. We introduce for each statement a *variable type environment* $\mathcal{V} : Var \to Pol$. For each variable x of the statement, $\mathcal{V}(x)$ is the security type for x.

Remark 1. The security types of variables do not depend on channel components because channel components do not always have current valuations. Hence, if the security types of variables depended on channel components, the values communicated over channel components in the history or the future would be needed to evaluate the security type of a variable.

Environments for Values and Live Variables. In security type checking, information about the values of program variables is used to refine the security types for the polyadic channels and variables. We introduce for each statement an *assertion environment* $\Phi : Pt \to Asst$ that records the value assertion at each program point. The environment Φ provides an over-approximation of the computation in the sense that the values of variables satisfying the assertions recorded in Φ constitute supersets of the values actually arising in the computation.

The security type system relies on the condition that if the execution of the given concurrent statement CS could reach some global configuration $gcnf$, such that the k-th statement S_k can take one further step from $gcnf$, then the assertion in Φ at the first program point of S_k is satisfied in $gcnf$.

We define $\mathit{may\text{-}step}(CS, \vec{\imath}, k)$ to state that the k-th process admits a further step, when all the processes are jointly at the program points $\vec{\imath}$, in the execution of the concurrent statement CS[4].

[4] Note that according to the semantics, an action can always be performed locally by a process that has not terminated, although the action can be impossible at the system-level due to the environment.

Definition 4. *may-step*$(CS, \vec{\imath}, k) \triangleq \exists A, \vec{\imath'}, sa : (\vec{\imath}, A, \vec{\imath'}) \in E_{CS} \wedge (k, sa) \in A$

We then define the aforementioned condition on Φ below.

Definition 5. *We define* $asst(\Phi, CS, k)$ *to express if* $tr \in$ *traces-of*$_\xi(CS)$ *for some* $\xi \in \Xi$, $last(tr) = gcnf = \langle\langle p_1, S'_1, m_1\rangle, \ldots, \langle p_n, S'_n, m_n\rangle\rangle$, *and it holds that* *may-step*$(CS, fst(S'_1) \ldots fst(S'_n), k)$, *then we have* $[\![\Phi(fst(S'_k))]\!]_{gcnf} = tt$.

In Section 7, we devise static analyses to compute a vector $\vec{\Phi}$ of assertion environments such that $asst(\Phi_k, CS, k)$ holds for all k.

The security types for variables are used to enforce the flow policies for polyadic channels because the variables pass information from input to output. In fact, only the live variables [40] are essential in this process. We introduce for each statement a *live variables environment* $\Lambda : Pt \to \mathcal{P}(Var)$ that records the set of live variables at each program point.

The design of the security type system is also based on the condition that the environment Λ for a statement S (in the concurrent statement CS to be typed) properly records the set of semantically live variables [39] at each program point of S. Intuitively, a variable is semantically live if varying its value does affect the subsequent execution, or the computation result. We denote this condition by $live(\Lambda, S)$, which is formally defined in Appendix A. The satisfaction of $live(\Lambda, S)$ can be ensured by using a standard live variables analysis (e.g., [40]) on S to infer Λ. For self-containedness, we present such an analysis in Appendix A.

5.2 Typing Judgments and Typing Rules

Typing Judgment and Typing Rules for Statements.

The typing judgment for a statement S is

$$\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ S\ (l', X') : \imath$$

The judgment says: S is well-typed under the channel type environment $\mathcal{C}$, the variable type environment $\mathcal{V}$, the assertion environment Φ, the live variables environment Λ, the pre-context (l, X), and the post-context (l', X'). To understand the use of the pre-context and the post-context, note that the reaching of a program point may rely on the presence of certain communications, as well as the evaluation of certain conditional expressions to specific Boolean values. The l in the pre-context is used to maintain an upper bound on the confidentiality levels for the presence of communications that contributes to reaching the beginning of S (hence, the presence of these communications could be inferred from the fact that the beginning of S has been reached). The X in the pre-context is used to maintain a superset of the variables in the conditional expressions whose specific evaluation results contribute to reaching the beginning of S (hence, information about these variables could be inferred from the fact that the beginning of S has been reached). The role of l' and X' is like that of l and X, except that l' and X' are used to record information relating to reaching the program point immediately after S (as opposed to the program point in the beginning of S).

The pre-context and the post-context are used to obtain an upper bound on the confidentiality level for the source of implicit information flow. Their role resembles that of the PC label in classical security type systems (e.g., [49]).

We introduce a few pieces of notation that allow for a concise presentation of our security typing rules. For a function f of type $U \to \mathit{Pol}$ where $U \subseteq \mathit{Var} \cup \mathit{Ch} \cup \mathit{PCh}$, we write $\phi \triangleright f$ for the function $\lambda x.(\phi \triangleright f(x))$, and if $X \subseteq \mathit{Var}$, we write $f[X]$ for the security type $\bigvee_{x \in X} f(x)$, which is $\star$ if $X = \emptyset$. For such a function f, we also write $f[x \mapsto P]$ for the function that is as f except that x is mapped to the security type P, write $f[(u_j \mapsto P_j)_j]$ for the function that is as f except that each $u_j \in \mathit{Var} \cup \mathit{Ch}$ is mapped to the security type P_j, write $f[e/x]$ for the function that maps each variable x' to the security type resulting from applying the substitution e/x on the formulas in the security type $f(x')$, and write $f[(e_j/u_j)_j]$ for the function that maps each variable x' to the security type resulting from applying a series of substitutions e_j/u_j on $f(x')$. For two functions f_1 and f_2 of type $U \to P$ where $U \subseteq \mathit{Var} \cup \mathit{Ch} \cup \mathit{PCh}$, we write $f_1 \preceq_U f_2$ for the condition $\forall u \in \mathit{dom}(f_1) \cap \mathit{dom}(f_2) \cap U : f_1(u) \preceq f_2(u)$.

The typing rules for statements are given as a calculus for the judgment $\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ S\ (l', X') : \imath$ in Fig. 7.

The rule for $^\imath\mathsf{skip}$ says that $^\imath\mathsf{skip}$ is always typable, and the fact of reaching the completion point of this statement has the same level of confidentiality as the fact of reaching the starting point of this statement does.

The rule for $x :=^\imath e$ requires the condition $(\Phi(\imath) \triangleright l \curlywedge \mathcal{V}[X \cup \mathit{fvs}(e)]) \preceq \mathcal{V}(x)[[e]@p/x@p]$ for the variable x subject to the assignment. With $\mathit{fvs}(e)$ on the left of the constraint, explicit information leakage from e to x is prevented. With l and X on the left of the constraint, implicit information leakage caused by leaking the fact of reaching the assignment through the value of x after the assignment is prevented. By imposing the pre-condition $\Phi(\imath)$ (which holds in the beginning of the assignment), the left-hand side of the constraint is made less confidential in general and the overall constraint is relaxed. With the substitution $[[e]@p/x@p]$ on the right of the constraint, the security type of x is interpreted in the state reached after the assignment rather than before it. For a variable $x' \neq x$, if x' is live at the end of the assignment, then it is required that $\Phi(\imath) \triangleright \mathcal{V}(x') \preceq \mathcal{V}(x')[[e]@p/x@p]$ (because of the decoration of $\preceq$ with the set $\Lambda(\imath')$ of live variables at $\imath'$). With this requirement, it is ensured that the live variables do not leak information when their security types are weakened due to any change in the value of x.

Example 1 (Detection of Explicit Leakage). Let $\mathcal{V} = [x \mapsto \{p\}, x' \mapsto \{p'\}]$, $\Phi = [1 \mapsto \mathsf{true}, \iota_\mathrm{f} \mapsto \mathsf{true}]$, and $\Lambda = [1 \mapsto \{x'\}, \iota_\mathrm{f} \mapsto \emptyset]$. The assignment $x :=^1 x'$ causes information flow from the variable x' to the variable x. According to the security type of x', the only principal permitted to access the information in x' is p'. However, the information flow from x' to x gives p (in addition to p') access to some of this information. Hence, the assignment induces explicit information leakage. This leakage is detected by the typing rule for assignments. Formally, it is impossible to establish $\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (\star, \emptyset)\ x :=^1 x'\ (\star, \emptyset) : \iota_\mathrm{f}$ using this rule. The premise of this rule requires $\Phi(1) \triangleright \mathcal{V}[x \mapsto \star \curlywedge \mathcal{V}[\emptyset \cup \mathit{fvs}(x')]] \preceq_{\emptyset \cup \{x\}}$

$$\frac{}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{skip}\ (l, X) : \imath'}$$

$$\frac{\Phi(\imath) \triangleright \mathcal{V}[x \mapsto l \curlywedge \mathcal{V}[X \cup fvs(e)]] \preceq_{\Lambda(\imath') \cup \{x\}} \mathcal{V}[[e]@p/x@p]}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ x :=^{\imath} e\ (l, X) : \imath'}$$

$$\frac{\begin{array}{c} l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}[X]) \preceq \mathcal{C}(c) \preceq l' \\ \Phi(\imath) \triangleright \mathcal{V}[(c.j \mapsto \mathcal{V}[fvs(e_j) \cup X])_j] \preceq_{\Lambda(\imath') \cup \{c.1, \ldots, c._{|c|}\}} (\mathcal{V} \uplus \mathcal{C})[([e_j]@p/c.j)_j] \end{array}}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{send}(c, \vec{e})\ (l', X) : \imath'}$$

$$\frac{\begin{array}{c} l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}[X]) \preceq \mathcal{C}(c) \preceq l' \\ \Phi(\imath) \triangleright \mathcal{V}[(x_j \mapsto \mathcal{C}(c.j) \curlywedge \mathcal{V}[X])_j] \preceq_{\Lambda(\imath') \cup \{\vec{x}\}} \mathcal{V}[(c.j/x_j@p)_j] \end{array}}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{recv}(c, \vec{x})\ (l', X) : \imath'}$$

$$\frac{\begin{array}{c} \mathcal{C}, \mathcal{V}^{p,e}_{l',X',\imath'} \vdash_p^{\Phi,\Lambda} (l, X \cup fvs(e))\ S_1\ (l', X') : \imath' \\ \mathcal{C}, \mathcal{V}^{p,\neg e}_{l',X',\imath'} \vdash_p^{\Phi,\Lambda} (l, X \cup fvs(e))\ S_2\ (l', X') : \imath' \end{array}}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}\ (l', X') : \imath'}$$

$$\frac{X \cup fvs(e) \subseteq X' \qquad \mathcal{C}, \mathcal{V}^{p,e}_{l,X',\imath} \vdash_p^{\Phi,\Lambda} (l, X')\ S\ (l, X') : \imath}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}\ (l, X') : \imath'}$$

$$\frac{\begin{array}{c} \mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ S_1\ (l'', X'') : fst(S_2) \\ \mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l'', X'')\ S_2\ (l', X') : \imath' \end{array}}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l, X)\ S_1; S_2\ (l', X') : \imath'}$$

$$\frac{\begin{array}{c} \mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l_1', X_1')\ S\ (l_2', X_2') : \imath' \\ l_1 \preceq l_1' \quad l_2' \preceq l_2 \quad X_1 \subseteq X_1' \quad X_2' \subseteq X_2 \end{array}}{\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (l_1, X_1)\ S\ (l_2, X_2) : \imath'}$$

Fig. 7. The Information Flow Type System

$\mathcal{V}[[x']@p/x@p]$. For x, this requirement boils down to $\mathsf{true} \triangleright (\star \curlywedge \{p'\}) \preceq \{p\}$, which does not hold. In a slightly more involved program, the data in x' could be received from a channel before the assignment, and the data in x could be sent to another channel after the assignment. Thus, explicit leakage from the input channel to the output channel could be induced via the assignment. □

In the rule for ${}^{\imath}\mathsf{send}(c, \vec{e})$, the condition $l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}[X]) \preceq \mathcal{C}(c)$ ensures that no confidential information about reaching $\imath$ may be leaked through the presence of communication over c. The condition $\mathcal{C}(c) \preceq l'$ ensures that no confidential information about the presence of communication over c may be leaked through observing the completion of the statement. The second condition of the typing rule for ${}^{\imath}\mathsf{send}(c, \vec{e})$ can be understood by an analogy of the statement with the imaginary assignments $c.1 := e_1, \ldots, c.n := e_n$, where $n = |c|$. On the right-hand side of this condition, the environment $\mathcal{V} \uplus \mathcal{C}$ is formed because the security types of the destinations $c.1, \ldots, c.n$ of information flow are needed.

In the rule for $^{\imath}\mathsf{recv}(c, \vec{x})$, the first condition provides similar guarantees to those provided by the first condition of the rule for $^{\imath}\mathsf{send}(c, \vec{e})$. The second condition can be understood by an analogy of the statement with the imaginary assignments $x_1 := c.1, \ldots, x_n := c.n$.

The rules for the if and while statements implement the scoped specialization of security types: the conditions in the security types of variables can be turned into logical truth or falsehood in the whole branches of the if or the complete body of the while, after considering the conditional expression of the if or while. This operation mitigates the problem that changes of the values of variables in the scope of a if or while could cause changes of a security type that in turn render the system un-typable. Below, we define the specialization of a single security type in the INF form with a conditional expression in the statement of a specific principal.

Definition 6. *The specialization of the security type* $\bigwedge_j(\phi_j \triangleright R_j)$ *under the Boolean expression* e *of principal* p *is given by*

$$(\bigwedge_j(\phi_j \triangleright R_j))^{p,e} \triangleq \bigwedge_j(\phi'_j \triangleright R_j) \text{ where for each } j,\ \phi'_j = \begin{cases} \mathsf{true} & \text{if } [e]@p \Rightarrow \phi_j \\ \mathsf{false} & \text{if } \mathit{unsat}([e]@p \wedge \phi_j) \\ \phi_j & \text{otherwise} \end{cases}$$

In the definition, $[e]@p$ represents the expression that is as e except that each variable x of e is replaced with $x@p$. For each conjunct in the security type (in INF form) with condition ϕ_j, if $[e]@p$ implies ϕ_j, then the condition of the conjunct is replaced with true. If $[e]@p$ and ϕ_j cannot be satisfied at the same time, then the condition of the conjunct is replaced with false. Otherwise, the condition of the conjunct is left unchanged.

Example 2 (Specialization of Security Types). We have $(x@p{<}0 \triangleright \{p, p'\})^{p,\, x<-1}$ $= (\mathsf{true} \triangleright \{p, p'\})$. This is because we have $[x < -1]@p \Rightarrow (x@p < 0)$. □

We lift the type specialization operation to type environments.

Definition 7. *The specialization of the variable type environment* $\mathcal{V}$ *under expression* e *of principal* p, $l \in \mathcal{P}(Pr)$, $X \in \mathcal{P}(Var)$, *and* $\imath \in Pt$, *is given by*

$$\mathcal{V}^{p,e}_{l,X,\imath}(x) \triangleq \begin{cases} (\mathit{inf}(\mathcal{V}(x)))^{p,e} & \text{if } x \notin \Lambda(\imath) \wedge l \cap \bigcap_{x \in X} \mathit{frs}(\mathcal{V}(x)) = Pr \\ \mathcal{V}(x) & \text{otherwise} \end{cases}$$

In this definition, $\mathit{frs}(P)$ represents $\bigcap_k R_k$, where the R_k's satisfy $\bigwedge_k(\_ \triangleright R_k) = \mathit{inf}(P)$. Hence, the specialization of a variable type environment $\mathcal{V}$ under expression e of principal p, $l \in \mathcal{P}(Pr)$, $X \in \mathcal{P}(Var)$, and $\imath \in Pt$, is a new variable type environment $\mathcal{V}'$. The variable type environment $\mathcal{V}'$ maps each variable x to its specialized security type (wrt. p and e) only if x is not live at $\imath$, and all principals are constantly in l and are constantly readers of x. Here, $\imath$ represents the program point at the end of the scope in which the specialized security types are used, and l and X indicate the confidentiality of reaching $\imath$. Hence, the condition required for the specialization of the security types of variables ensures that the security type of each live variable is unaffected when recovering from

specialized security types to the original security types, and that the recovery to the original security types does not depend on confidential information. This avoids information leakage caused by exiting the scope in which the specialized type environment is used.

In the rule for $^{\iota}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$, the branches S_1 and S_2 are typed using $X \cup \mathit{fvs}(e)$ to capture that the level of confidentiality for reaching S_1 and S_2 is higher than the level of confidentiality for e. This reflects the fact that knowing which branch is executed reveals information about the evaluation result of e (known as implicit information leakage). In addition, each variable in $\mathit{fvs}(e)$ is kept inside the X' that is a part of the post-context for the if. This avoids internal timing leakage incurred by the racing of processes in concurrent execution.

Example 3 (Detection of Implicit Leakage). Let $\mathcal{V} = [x \mapsto \{p\}, x' \mapsto \{p'\}]$, Φ be the assertion environment that maps all program points to true, and $\Lambda = [1 \mapsto \{x'\}, 2 \mapsto \emptyset, 3 \mapsto \emptyset, \iota_{\mathrm{f}} \mapsto \emptyset]$. The execution of $^1\mathsf{if}\ x' > 0\ \mathsf{then}\ x :=^2 2\ \mathsf{else}\ ^3\mathsf{skip}\ \mathsf{fi}$ causes an information flow from the variable x' to the variable x. As a result, implicit information leakage is incurred according to the security types of x and x'. This leakage is detected by the typing rule for if. Formally, it is impossible to establish $\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (\star, \emptyset)\ ^1\mathsf{if}\ x' > 0\ \mathsf{then}\ x :=^2 2\ \mathsf{else}\ ^3\mathsf{skip}\ \mathsf{fi}\ (l', X') : \iota_{\mathrm{f}}$ by this typing rule, for any l' and X'. The premises of the rule require $\mathcal{C}, \mathcal{V}^{p,x'>0}_{l',X',\iota_{\mathrm{f}}} \vdash_p^{\Phi,\Lambda} (\star, \{x'\})\ x :=^2 2\ (l', X') : \iota_{\mathrm{f}}$, which cannot be established due to the existence of x' in the pre-context. □

Example 4 (Detection of Internal Timing Leakage). Let $\mathcal{V} = [x \mapsto \{p\}, x' \mapsto \{p'\}]$, Φ be the assertion environment that maps all program points to true, and Λ be the live variables environment that maps the program point 1 to $\{x'\}$ and all the other program points to $\emptyset$. Consider the system that consists of two processes – executing the statement $^1\mathsf{if}\ x' > 0\ \mathsf{then}\ ^2\mathsf{skip}; {}^3\mathsf{skip}\ \mathsf{else}\ ^4\mathsf{skip}\ \mathsf{fi}; x :=^5 2$ and the statement $x :=^6 1$, respectively. If the system is scheduled by a round-robin scheduler with time slice 3, then the final value of x is 2 if the initial value of x' is positive, and the final value of x is 1 otherwise. This information flow from x' to x causes an internal timing leakage due to the security types of the two variables. This leakage is detected by the typing rules for if, assignments, and sequential composition. More concretely, the variable x' is included in the post-context for the if statement, due to the typing rule for if. Hence, x' is also included in the pre-context for the assignment $x :=^5 2$, as is mandated by the rule for sequential composition. However, this assignment cannot be typed with such a pre-context, according to the rule for assignments. □

In the rule for $^{\iota}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}$, l and X' constitute an invariant for the level of confidentiality for the control flow. That is, after each round of the loop, the level of confidentiality for the presence of communications and for the conditional expressions that contribute to the completion of the round stays at what is captured by l and X'. Like the rule for $^{\iota}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}$, the body of the loop is typed with $\mathit{fvs}(e)$ as part of its pre-context, which avoids implicit information leakage. In addition, $\mathit{fvs}(e)$ is made part of the post-context for $^{\iota}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}$, which avoids internal timing leakage.

The last rule in Fig. 7 is a sub-typing rule. The rule says that if a statement S has a way of being typed that reflects a higher level of confidentiality for reaching S, and a lower level of confidentiality for completing S, then S has a way of being typed that reflects a lower level of confidentiality for reaching S, and a higher level of confidentiality for completing S. The use of this rule is illustrated using the example below.

Example 5 (Subtyping). Consider the case where $\mathcal{V} = [x \mapsto \{p\}, x' \mapsto \{p'\}]$, $\Phi = [1 \mapsto \mathsf{true}, \iota_{\mathrm{f}} \mapsto \mathsf{true}]$, $\Lambda = [1 \mapsto \emptyset, \iota_{\mathrm{f}} \mapsto \emptyset]$. Using the typing rule for assignments, it can be derived that $\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (\{p\}, \{x\})\ x :=^1 2\ (\{p\}, \{x\}) : \iota_{\mathrm{f}}$. This is because

$$\mathsf{true} \triangleright \mathcal{V}[x \mapsto \{p\} \curlywedge \mathcal{V}[\{x\} \cup \emptyset]] \preceq_{\emptyset \cup \{x\}} \mathcal{V}[[2]@p/x@p] \tag{2}$$

Using the sub-typing rule, it can be derived, e.g.,

$$\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (\{p, p'\}, \emptyset)\ x :=^1 2\ (\{p\}, \{x\}) : \iota_{\mathrm{f}}$$

In the above, the pre-context $(\{p, p'\}, \emptyset)$ captures a lower level of confidentiality than the pre-context $(\{p\}, \{x\})$ does. This is in line with the fact that replacing the $\{p\}$ and $\{x\}$ on the left side of (2) with $\{p, p'\}$ and $\emptyset$, respectively, makes the constraint more easily satisfied. This relaxation of the pre-context can be used when typing a sequential composition where the assignment is the second component, or a conditional branching or loop statement containing the assignment.

Using the sub-typing rule, it can also be derived, e.g.,

$$\mathcal{C}, \mathcal{V} \vdash_p^{\Phi,\Lambda} (\{p\}, \{x\})\ x :=^1 2\ (\emptyset, \{x, x'\}) : \iota_{\mathrm{f}}$$

In the above, the post-context $(\emptyset, \{x, x'\})$ captures a higher level of confidentiality than the post-context $(\{p\}, \{x\})$ does. This is in line with the fact that the l' in the post-context is an *upper bound* on the confidentiality level for the presence of communications that contributes to reaching ι_{f}, and the X' in the post-context is a *superset* of the variables in the conditional expressions whose evaluation contributes to reaching ι_{f}. □

Typing Judgment and Typing Rules for Concurrent Statements.

The typing judgment for a concurrent statement CS is of the form

$$\mathcal{C}, \vec{\mathcal{V}} \vdash_{\vec{\Lambda}}^{\vec{\Phi}} CS$$

This judgment says: the concurrent statement CS is well-typed under the channel type environment $\mathcal{C}$, the list $\vec{\mathcal{V}}$ of variable type environments, the list $\vec{\Phi}$ of assertion environments, and the list $\vec{\Lambda}$ of live variables environments, where each $\mathcal{V}_i$ (resp. Φ_i, Λ_i) is for the i-th sequential statement in the concurrent statement.

The only calculus rule for the aforementioned judgment is

$$\frac{\forall i : \mathcal{C}, \mathcal{V}_i \vdash_{p_i}^{\Phi_i,\Lambda_i} (\star, \emptyset)\ S_i\ (l_i, X_i) : \iota_{\mathrm{f}}}{\mathcal{C}, \vec{\mathcal{V}} \vdash_{\vec{\Lambda}}^{\vec{\Phi}} p_1 : S_1 || \ldots || p_n : S_n} \text{ if } nip(\mathcal{C}, \vec{\mathcal{V}}) \wedge no\text{-}upd(||_i\, p_i : S_i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{X})$$

Hence, the well-typedness of a concurrent statement requires the well-typedness of all its sequential components under the initial context $(\star, \emptyset)$, and two additional conditions. The initial context $(\star, \emptyset)$ reflects that no confidential information is leaked by starting the execution of each sequential statement.

The condition $nip(\mathcal{C}, \vec{\mathcal{V}})$ prevents information leakage via security types – the values of variables or channel components depended upon by a security type could be leaked into the concrete set of principals resulting from the evaluation of the security type. The condition $nip(\mathcal{C}, \vec{\mathcal{V}})$ is formulated via the following auxiliary condition that the security type $\bigwedge_{1\leq j\leq n}(\phi_j \triangleright R_j)$ in INF does not leak information.

$$nip(\mathcal{C}, \vec{\mathcal{V}}, \bigwedge_{1\leq j\leq n}(\phi_j \triangleright R_j)) \triangleq (Pr \setminus (\bigcap_{1\leq j\leq n} R_j)) \subseteq \bigcap_{1\leq j\leq n}\bigcap_{u\in fvs(\phi_j)} frs\text{-}cv(\mathcal{C}, \vec{\mathcal{V}}, u)$$

$$\text{where } frs\text{-}cv(\mathcal{C}, \vec{\mathcal{V}}, u) \triangleq \begin{cases} frs(\mathcal{V}_j(x)) & \text{if } u = x@p_j \\ frs(\mathcal{C}(c.j)) & \text{if } u = c.j \end{cases}$$

In the above, $frs\text{-}cv(\mathcal{C}, \vec{\mathcal{V}}, u)$ syntactically characterizes a set of principals that are always allowed as readers of u. Here, u is either of the form $x@p_j$ or a channel component. Overall, the condition $nip(\mathcal{C}, \vec{\mathcal{V}}, \bigwedge_{1\leq j\leq n}(\phi_j \triangleright X_j))$ says: Each principal not in all the R_j's must be constantly allowed as a reader by the security types of all the variables in all the ϕ_j's. Hence, the observation that a principal makes about its own observational privilege according to a security type P, does not leak information about the variables or channel components in P.

Using the condition $nip(\mathcal{C}, \vec{\mathcal{V}}, \bigwedge_{1\leq j\leq n}(\phi_j \triangleright R_j))$, the condition $nip(\mathcal{C}, \vec{\mathcal{V}})$ is formulated as

$$\begin{aligned} nip(\mathcal{C}, \vec{\mathcal{V}}) \triangleq\ & (\forall x \in Var : nip(\mathcal{C}, \vec{\mathcal{V}}, inf(\mathcal{V}(x)))) \wedge \\ & (\forall c, j\ s.t.\ c.j \in Ch : nip(\mathcal{C}, \vec{\mathcal{V}}, inf(\mathcal{C}(c.j)))) \end{aligned}$$

Hence, it is required that the security type of each variable and each channel component must not leak information.

The condition $no\text{-}upd(CS, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{X})$ is used to avoid the weakening of the security types governing the data flow or the control flow of a process by the concurrent update of variables. This condition is defined by

$$no\text{-}upd(CS, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{X}) \triangleq \forall j \neq i : \forall x, x', \vec{\imath} : \left(\begin{pmatrix} x \in \Lambda_j(\imath_j) \vee \\ x \in X_j \wedge frs(\mathcal{V}_j(x)) \neq Pr \end{pmatrix} \wedge x'@p_i \in fvs(\mathcal{V}_j(x)) \right) \Rightarrow no\text{-}upd\text{-}var(CS, \vec{\imath}, i, x')$$

$$no\text{-}upd\text{-}var(CS, \vec{\imath}, i, x') \triangleq \forall \vec{\imath'}, A, sa : (\vec{\imath}, A, \vec{\imath'}) \in E_{CS} \wedge (i, sa) \in A \Rightarrow x' \notin upd(sa)$$

In the above, the auxiliary condition $no\text{-}upd\text{-}var(CS, \vec{\imath}, i, x')$ represents that the variable x' of the i-th statement in CS is not updated in the next execution step when the processes are jointly at the program points in $\vec{\imath}$. The condition $no\text{-}upd(CS, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{X})$ says: (1) no concurrent update of any variable in the security type of a live variable (i.e., a member of $\Lambda_j(\imath_j)$ for some j) may happen in any potential execution of the concurrent statement CS; (2) no concurrent update of any confidential variable in the context (i.e., X_j for some j) of a statement may happen in any potential execution of the concurrent statement

CS. Hence, the weakening of the security types of the confidential variables or confidential control flow (by remotely updating variables in policies) is avoided.

The information-flow analysis realized by our security type system is compositional for its main part. This is because the well-typedness of a concurrent statement can be deduced (conditionally) from the well-typedness of the individual sequential components of the concurrent statement.

5.3 Typing the Stateful Packet Filtering System

We concretize the stateful packet filtering scenario in Section 2 by implementing each module of the system in the programming language of Section 3.3. This implementation is shown in Fig. 8. The state of the filter is maintained in the variable s. The numbering scheme realized (via s) is such that

- messages from the client arrive properly at the server if and only if $h_1@\mathsf{cli} = (h_2@\mathsf{srv} + 1)\%N$ holds, and
- messages from the server arrive properly at the client if and only if $h_2@\mathsf{srv} = (h_1@\mathsf{cli} + 1)\%N$ holds.

The realization of the numbering scheme is mainly achieved through the conditions at the program points 3 and 8 in the statement of fil.

```
cli :                    fil :                          srv :
[1]while true do         [1]while true do               [1]while true do
  [2]recv(c_usr, h1, pl1);   [2]recv(c1, h1, pl1);        [2]recv(c2, h1, pl1);
  [3]send(c1, h1, pl1);      [3]if h1 = (s+1)%N           [3]send(c'_app, h1, pl1);
  [4]recv(c'1, h2, pl2);       then s :=[4] h1;           [4]recv(c_app, h2, pl2);
  [5]send(c'_usr, h2, pl2)           [5]send(c2, h1, pl1) [5]send(c'2, h2, pl2)
 od                            else[6]send(c_au, h1, pl1)  od
                               fi;
                             [7]recv(c'2, h2, pl2);
                             [8]if h2 = (s+1)%N
                               then s :=[9] h2;
                                    [10]send(c'1, h2, pl2)
                               else[11]send(c_au, h2, pl2)
                               fi
                             od
```

Fig. 8. The Concurrent Statement CS_{SPF} for the Stateful Packet Filtering System

We use our security type system to verify that the system is information flow secure under globally dependent flow policies of the kind that is informally introduced in Section 2. To avoid tediousness, we only discuss the part of the typing that reflects the key technical elements of our security type system.

We use the policy $\star$ for the presence of communication over all polyadic channels, and for the content of the first components of all polyadic channels. That is, information about the presence of communication over all polyadic channels and all message headers may be revealed to all principals in the system.

$$
\begin{aligned}
\mathcal{C}(c_{\mathrm{usr}}.2) &= (c_{\mathrm{usr}}.1 = (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{app}\}) \curlywedge (c_{\mathrm{usr}}.1 \neq (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{C}(c_1.2) &= (c_1.1 = (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{app}\}) \curlywedge (c_1.1 \neq (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{C}(c_2.2) &= \mathcal{C}(c'_{\mathrm{app}}.2) = \{\mathsf{app}\}\\
\mathcal{C}(c_{\mathrm{app}}.2) &= (c_{\mathrm{app}}.1 = (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{usr}\}) \curlywedge (c_{\mathrm{app}}.1 \neq (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{C}(c'_2.2) &= (c'_2.1 = (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{usr}\}) \curlywedge (c'_2.1 \neq (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{C}(c'_1.2) &= \mathcal{C}(c'_{\mathrm{usr}}.2) = \{\mathsf{usr}\}\\
\mathcal{C}(c_{\mathrm{au}}.2) &= \{\mathsf{au}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathcal{V}_1(pl_1) &= (h_1@\mathsf{cli} = (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{app}\}) \curlywedge (h_1@\mathsf{cli} \neq (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{V}_2(pl_1) &= (h_1@\mathsf{fil} = (s@\mathsf{fil}+1)\%N \triangleright \{\mathsf{app}\}) \curlywedge (h_1@\mathsf{fil} \neq (s@\mathsf{fil}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{V}_3(pl_1) &= \{\mathsf{app}\}\\
\mathcal{V}_1(pl_2) &= \{\mathsf{usr}\}\\
\mathcal{V}_2(pl_2) &= (h_2@\mathsf{fil} = (s@\mathsf{fil}+1)\%N \triangleright \{\mathsf{usr}\}) \curlywedge (h_2@\mathsf{fil} \neq (s@\mathsf{fil}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{V}_3(pl_2) &= (h_2@\mathsf{srv} = (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{usr}\}) \curlywedge (h_2@\mathsf{srv} \neq (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{au}\})\\
\mathcal{V}_2(s) &= \star
\end{aligned}
$$

Fig. 9. The Security Types Used to Type-check the Stateful Packet Filtering System

The rest of the policy assignment for channels is reflected by the channel type environment $\mathcal{C}$ as specified in Fig. 9 (recall that the security types for channels are the same as their flow policies). The security types for variables are specified in the variable type environment $\mathcal{V}$ in the same figure. As is shown in Fig. 9, the payload of messages over c_{usr}, c_{app}, c_1, and c'_2 is under the protection of globally dependent flow policies. The security types for the variable pl_1 of the client and the filter, and the variable pl_2 of the filter and the server, also admit content-dependency.

We then illustrate the key points of the security typing with three examples. In the security type system, the environments $\vec{\Phi}$ and $\vec{\Lambda}$ are needed. Since a full specification of them is tedious (with $\vec{\Phi}$ containing complex formulas), we only partially describe these environments in our examples.

Example 6. For ${}^3\mathsf{send}(c_1, h_1, pl_1)$ in the statement of cli, it can be established that $\mathcal{C}, \mathcal{V}_1 \vdash^{\Phi_1,\Lambda_1}_{\mathsf{cli}} (\star, \emptyset)\ {}^3\mathsf{send}(c_1, h_1, pl_1)\ (\star, \emptyset) : 4$ using the typing rule for output. The first premise of the rule amounts to $\star \curlywedge (\Phi_1(3) \triangleright \mathcal{V}_1[\emptyset]) \preceq \star \preceq \star$. With the definition of $\preceq$, this condition holds, irrespective of the value of $\Phi_1(3)$. The second premise of the rule amounts to

$$
\begin{aligned}
&\Phi_1(3) \triangleright \mathcal{V}_1[c_1.1 \mapsto \mathcal{V}_1[\{h_1\} \cup \emptyset]][c_1.2 \mapsto \mathcal{V}_1[\{pl_1\} \cup \emptyset]]\\
&\quad \preceq_{\Lambda_1(4) \cup \{c_1.1, c_1.2\}} (\mathcal{V}_1 \uplus \mathcal{C})[h_1@\mathsf{cli}/c_1.1][pl_1@\mathsf{cli}/c_1.2]
\end{aligned}
$$

We have $\Lambda_1(4) = \emptyset$. Hence, the condition above amounts to

$$
\begin{aligned}
(\Phi_1(3) \triangleright \mathcal{V}_1[\{h_1\} \cup \emptyset]) &\preceq \mathcal{C}(c_1.1)[h_1@\mathsf{cli}/c_1.1][pl_1@\mathsf{cli}/c_1.2]\\
(\Phi_1(3) \triangleright \mathcal{V}_1[\{pl_1\} \cup \emptyset]) &\preceq \mathcal{C}(c_1.2)[h_1@\mathsf{cli}/c_1.1][pl_1@\mathsf{cli}/c_1.2]
\end{aligned}
$$

It is not difficult to verify the validity of both conditions above. □

Example 7. For ${}^{2}\mathsf{recv}(c_1, h_1, pl_1)$ in the statement of fil, it can be established that $\mathcal{C}, \mathcal{V}_2 \vdash^{\Phi_2,\Lambda_2}_{\mathsf{fil}} (\star, \emptyset)\ {}^{2}\mathsf{recv}(c_1, h_1, pl_1)\ (\star, \emptyset) : 3$ using the typing rule for input. The first premise of the rule amounts to $\star \curlywedge (\Phi_2(2) \triangleright \mathcal{V}_2[\emptyset]) \preceq \star \preceq \star$. The validity of this condition can be verified using the definition of $\preceq$, irrespective of the value of $\Phi_2(2)$. The second premise of the rule amounts to

$$\begin{array}{c}\Phi_2(2) \triangleright \mathcal{V}_2[h_1 \mapsto \mathcal{C}(c_1.1) \curlywedge \mathcal{V}_2[\emptyset]][pl_1 \mapsto \mathcal{C}(c_1.2) \curlywedge \mathcal{V}_2[\emptyset]] \\ \preceq_{\Lambda_2(3) \cup \{h_1, pl_1\}} \mathcal{V}_2[c_1.1/h_1@\mathsf{fil}][c_1.2/pl_1@\mathsf{fil}]\end{array}$$

We have $\Lambda_2(3) = \{h_1, pl_1, s\}$. Hence, the condition can be turned into

$$\begin{array}{c}(\Phi_2(2) \triangleright \mathcal{C}(c_1.1) \curlywedge \star) \preceq \mathcal{V}_2(h_1)[c_1.1/h_1@\mathsf{fil}][c_1.2/pl_1@\mathsf{fil}] \\ (\Phi_2(2) \triangleright \mathcal{C}(c_1.2) \curlywedge \star) \preceq \mathcal{V}_2(pl_1)[c_1.1/h_1@\mathsf{fil}][c_1.2/pl_1@\mathsf{fil}] \\ (\Phi_2(2) \triangleright \mathcal{V}_2(s)) \preceq \mathcal{V}_2(s)[c_1.1/h_1@\mathsf{fil}][c_1.2/pl_1@\mathsf{fil}]\end{array}$$

It is trivial to check that the first and third conditions above are satisfied. The second condition can be turned into

$$\begin{pmatrix}(\Phi_2(2) \wedge c_1.1 = (h_2@\mathsf{srv} + 1)\%N \triangleright \{\mathsf{app}\}) \\ \curlywedge (\Phi_2(2) \wedge c_1.1 \neq (h_2@\mathsf{srv} + 1)\%N \triangleright \{\mathsf{au}\}) \\ \curlywedge (\Phi_2(2) \wedge \mathsf{true} \triangleright \star)\end{pmatrix} \preceq_{\text{INF}} \begin{pmatrix}(c_1.1 = (s@\mathsf{fil} + 1)\%N \triangleright \{\mathsf{app}\}) \\ \curlywedge (c_1.1 \neq (s@\mathsf{fil} + 1)\%N \triangleright \{\mathsf{au}\})\end{pmatrix}$$

We have $\Phi_2(2) \Rightarrow h_2@\mathsf{srv} = s@\mathsf{fil}$ (which can be found out using our analyses to be presented later in this article). Therefore, the validity of the condition above can be verified. □

Example 8. For the if statement whose conditional expression is at program point 3 in the statement of fil, we have

$$\begin{array}{l}(\mathcal{V}_2)^{\mathsf{fil}, h_1 = (s+1)\%N}_{\star, \emptyset, 7} = [\{s, h_1, h_2\} \mapsto \mathsf{true} \triangleright \star, pl_1 \mapsto \mathsf{true} \triangleright \{\mathsf{app}\}, pl_2 \mapsto \mathit{inf}(\mathcal{V}_2(pl_2))] \\ (\mathcal{V}_2)^{\mathsf{fil}, h_1 \neq (s+1)\%N}_{\star, \emptyset, 7} = [\{s, h_1, h_2\} \mapsto \mathsf{true} \triangleright \star, pl_1 \mapsto \mathsf{true} \triangleright \{\mathsf{au}\}, pl_2 \mapsto \mathit{inf}(\mathcal{V}_2(pl_2))]\end{array}$$

Here, the mapping of a set of variables is shorthand for the mapping of each individual variable in the set to the same image. Let the two type environments $(\mathcal{V}_2)^{\mathsf{fil}, h_1 = (s+1)\%N}_{\star, \emptyset, 7}$ and $(\mathcal{V}_2)^{\mathsf{fil}, h_1 \neq (s+1)\%N}_{\star, \emptyset, 7}$ be denoted by $\mathcal{V}_t$ and $\mathcal{V}_f$, respectively.

For the assignment $s := {}^{4}h_1$ in the statement of fil, it can be established that $\mathcal{C}, \mathcal{V}_t \vdash^{\Phi_2,\Lambda_2}_{\mathsf{fil}} (\star, \{s, h_1\})\ s := {}^{4}h_1\ (\star, \{s, h_1\}) : 4$. With $\Lambda_2(5) = \{s, h_1, pl_1\}$, the only premise of the rule for assignments amounts to

$$\Phi(4) \triangleright \mathcal{V}_t[s \mapsto \star \curlywedge \mathcal{V}_t[\{s, h_1\}]] \preceq_{\{s, h_1, pl_1\}} \mathcal{V}_t[[h_1]@\mathsf{fil}/s@\mathsf{fil}] \tag{3}$$

Analogously to the examples above, it is not difficult to verify the validity of this condition. □

Permissiveness Enabled by Scoped Specialization. Note that with the baseline security type $\mathcal{V}_2(pl_1)$ for pl_1, the assignment $s := {}^4h_1$ in Example 8 cannot be typed. After this assignment, $h_1@\mathsf{fil} = (s@\mathsf{fil} + 1)\%N$ does not hold, and $\mathcal{V}_2(pl_1)$ allows information disclosure to $\{\mathsf{au}\}$. This disclosure is not allowed by $\mathcal{V}_2(pl_1)$ before the assignment (when $h_1@\mathsf{fil} = (s@\mathsf{fil} + 1)\%N$ holds), and, hence, information leakage is caused. The scoped specialization of the security type $\mathcal{V}_2(pl_1)$ avoids this change in the allowed target of information disclosure. This scoped specialization is a key enabler of the typing of the if statement in Example 8.

Permissiveness Enabled by Liveness Information. Note also that without exploiting information about the live variables, the assignment $s := {}^4h_1$ in Example 8 cannot be typed. Replacing the set $\{s, h_1, pl_1\}$ with the set of all variables in (3), the following would be required.

$$(\Phi(4) \triangleright \mathcal{V}_t(pl_2)) \preceq \mathcal{V}_t(pl_2)[[h_1]@\mathsf{fil}/s@\mathsf{fil}]$$

We have $\mathcal{V}_t(pl_2) = \mathcal{V}_2(pl_2)$ (i.e., the type specialization at the if has no effect on the security type of pl_2). Plugging in the value of $\mathcal{V}_2(pl_2)$, it can be seen that the condition above does not hold. This demonstrates the additional permissiveness of security typing as enabled by the exploitation of liveness information.

Fact 1 *Let $\vec{\Phi}$ be such that $\Phi_2(2) \Rightarrow h_2@\mathsf{srv} = s@\mathsf{fil}$, and $\Phi_2(7) \Rightarrow h_1@\mathsf{cli} = s@\mathsf{fil}$. Let $\vec{\Lambda}$ be in accordance with the following tables.*

$\Lambda_1(1)$	$\Lambda_1(2)$	$\Lambda_1(3)$	$\Lambda_1(4)$	$\Lambda_1(5)$
$\emptyset$	$\emptyset$	$\{h_1, pl_1\}$	$\emptyset$	$\{h_2, pl_2\}$

$\Lambda_3(1)$	$\Lambda_3(2)$	$\Lambda_3(3)$	$\Lambda_3(4)$	$\Lambda_3(5)$
$\emptyset$	$\emptyset$	$\{h_1, pl_1\}$	$\emptyset$	$\{h_2, pl_2\}$

$\Lambda_2(1)$	$\Lambda_2(2)$	$\Lambda_2(3)$	$\Lambda_2(4)$	$\Lambda_2(5)$	$\Lambda_2(6)$
$\{s\}$	$\{s\}$	$\{h_1, pl_1, s\}$	$\{h_1, pl_1\}$	$\{h_1, pl_1, s\}$	$\{h_1, pl_1, s\}$
	$\Lambda_2(7)$	$\Lambda_2(8)$	$\Lambda_2(9)$	$\Lambda_2(10)$	$\Lambda_2(11)$
	$\{s\}$	$\{h_2, pl_2, s\}$	$\{h_2, pl_2\}$	$\{h_2, pl_2, s\}$	$\{h_2, pl_2, s\}$

Then, there exist $\mathcal{C}$ and $\vec{\mathcal{V}}$ such that $\mathcal{C}, \vec{\mathcal{V}} \vdash^{\vec{\Phi}}_{\vec{\Lambda}} CS_{\mathrm{SPF}}$ can be established. Moreover, it holds that $\mathcal{C}(c_{\mathrm{usr}}) = \mathcal{C}(c'_{\mathrm{usr}}) = \mathcal{C}(c_{\mathrm{app}}) = \mathcal{C}(c'_{\mathrm{app}}) = \star$, $\mathcal{C}(c_{\mathrm{usr}}.1) = \mathcal{C}(c'_{\mathrm{usr}}.1) = \mathcal{C}(c_{\mathrm{app}}.1) = \mathcal{C}(c'_{\mathrm{app}}.1) = \star$, $\mathcal{C}(c'_{\mathrm{app}}.2) = \{\mathsf{app}\}$, $\mathcal{C}(c'_{\mathrm{usr}}.2) = \{\mathsf{usr}\}$, and

$$\mathcal{C}(c_{\mathrm{usr}}.2) = (c_{\mathrm{usr}}.1 = (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{app}\}) \curlywedge (c_{\mathrm{usr}}.1 \neq (h_2@\mathsf{srv}+1)\%N \triangleright \{\mathsf{au}\})$$
$$\mathcal{C}(c_{\mathrm{app}}.2) = (c_{\mathrm{app}}.1 = (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{usr}\}) \curlywedge (c_{\mathrm{app}}.1 \neq (h_1@\mathsf{cli}+1)\%N \triangleright \{\mathsf{au}\})$$

We have now illustrated that the stateful packet filtering system is well-typed (under proper auxiliary information about values and live variables). We next articulate the security guarantees provided by well-typedness.

6 Soundness of Security Typing for Noninterference

We show that our security type system enforces information-flow security that is formulated as a noninterference-like property [13, 18]. More concretely, this

property takes the shape of nondeducibility on strategies [55, 46]. The key idea is, the confidential behaviors and data of the strategies of the environment cannot be distinguished by observing differences in the execution of the system.

In a series of definitions, we formulate the observation of executions, the indistinguishability of strategies, and the information-flow security of systems. We use the symbol $\circledcirc$ to represent the presence of some action that is unobservable. We use the symbol $\odot$ to represent some value that is unobservable.

Definition 8. *The observation of the element in the list $\vec{v}$ of values that is communicated over the j-th channel component of the polyadic channel c, by the group $\mathcal{G}$ of principals, in the global configuration $gcnf$, denoted by $\lfloor c, \vec{v}, j \rfloor^{\mathcal{G}}_{gcnf}$, is defined as*

$$\lfloor c, \vec{v}, j \rfloor^{\mathcal{G}}_{gcnf} \triangleq \begin{cases} v_j & \text{if } [\![\mathcal{C}(c.j)]\!]_{gcnf,[(c.j \mapsto v_j)_j]} \cap \mathcal{G} \neq \emptyset \\ \odot & \text{otherwise} \end{cases}$$

Hence, if the content policy of the j-th component of the channel c is evaluated into a set of principals some of which are in $\mathcal{G}$, then the observation of the value is the value itself. This captures that the value is properly observed by some principals in $\mathcal{G}$. On the other hand, if none of the principals in the set resulting from the evaluation is in $\mathcal{G}$, then the observation of the value ($\odot$) does not reveal any information about the value.

Definition 9. *The observation of the action α by the group $\mathcal{G}$ of principals in the global configuration $gcnf$, denoted by $\lfloor \alpha \rfloor^{\mathcal{G}}_{gcnf}$, is defined as*

$$\lfloor c\rho\vec{v} \rfloor^{\mathcal{G}}_{gcnf} \triangleq \begin{cases} c\rho\,[\lfloor c, \vec{v}, 1 \rfloor^{\mathcal{G}}_{gcnf}, \ldots, \lfloor c, \vec{v}, n \rfloor^{\mathcal{G}}_{gcnf}] & \text{if } \mathcal{C}(c) \cap \mathcal{G} \neq \emptyset \\ \circledcirc & \text{otherwise} \end{cases}$$

$$\lfloor \alpha \rfloor^{\mathcal{G}}_{gcnf} \triangleq \circledcirc \quad \text{if } \alpha \textit{ is not a communication}$$

Hence, for a communication over the polyadic channel c, if the presence of communication over c may be revealed to some principals in $\mathcal{G}$ according to the presence policy of c, then the observation of the communication consists of the polyadic channel c, the polarity of the communication, and the observation of the communicated values. This captures the proper observation of the presence of the communication by some of the principals in $\mathcal{G}$. On the other hand, if no principal in $\mathcal{G}$ is allowed to observe the presence of communication over c, then the observation ($\circledcirc$) does not reveal any information about the presence of the communication. In our model, the security interface of a system consists only of the communications performed between the system and its environment. Hence, the actions that are not communication actions are masked by $\circledcirc$.

Definition 10. *The observation of the trace tr by the group $\mathcal{G}$ of principals, denoted by $\lfloor tr \rfloor^{\mathcal{G}}$, is inductively defined as*

$$\lfloor gcnf \rfloor^{\mathcal{G}} \triangleq \epsilon$$

$$\lfloor tr'.\alpha.gcnf \rfloor^{\mathcal{G}} \triangleq \lfloor tr' \rfloor^{\mathcal{G}}.\lfloor \alpha \rfloor^{\mathcal{G}}_{last(tr')}$$

Hence, the observation of a trace consists of the observation of all the actions in the trace in the same order. Each action is observed in the global configuration where the action takes place.

Definition 11. *The observation of the communication intent ci by the group $\mathcal{G}$ of principals in the global configuration $gcnf$, denoted by $\lfloor ci \rfloor^{\mathcal{G}}_{gcnf}$, is defined as*

$$\lfloor ci \rfloor^{\mathcal{G}}_{gcnf} \triangleq \begin{cases} \lfloor c!\vec{v} \rfloor^{\mathcal{G}}_{gcnf} & \text{if } ci = c!\vec{v} \\ c?\cdot & \text{if } ci = c?\cdot \wedge \mathcal{C}(c) \cap \mathcal{G} \neq \emptyset \\ \circledcirc & \text{otherwise} \end{cases}$$

Hence, for a communication intent over the polyadic channel c, if the presence of communication over c may be revealed to some principals in $\mathcal{G}$, then the observation of the communication intent consists of the polyadic channel c, the polarity of the communication intent, and the observation of the values communicated by the environment in case the communication intent is an output (performed by the environment). On the other hand, if no principal in $\mathcal{G}$ may observe the presence of communication over c, then the observation ($\circledcirc$) does not reveal any information about the communication intent.

Definition 12. *The observation of the interaction intent ii by the group $\mathcal{G}$ of principals in the global configuration $gcnf$, denoted by $\lfloor ii \rfloor^{\mathcal{G}}_{gcnf}$, is defined as $\lfloor (p, ci) \rfloor^{\mathcal{G}}_{gcnf} \triangleq (p, \lfloor ci \rfloor^{\mathcal{G}}_{gcnf})$, and $\lfloor (p_1, p_2) \rfloor^{\mathcal{G}}_{gcnf} \triangleq (p_1, p_2)$.*

Hence, the observation of an interaction intent consists of the observed parts of the constituent communication intent, if any, and the principals in the interaction intent. This reflects that the scheduling decisions are known to the observer.

Definition 13. *Two strategies ξ_1 and ξ_2 are* indistinguishable *for the group $\mathcal{G}$ of principals, denoted by $\xi_1 \overset{\mathcal{G}}{=} \xi_2$, iff*

$$\forall tr_1, tr_2 : \lfloor tr_1 \rfloor^{\mathcal{G}} = \lfloor tr_2 \rfloor^{\mathcal{G}} \Rightarrow \lfloor \xi_1(tr_1) \rfloor^{\mathcal{G}}_{last(tr_1)} = \lfloor \xi_2(tr_2) \rfloor^{\mathcal{G}}_{last(tr_2)}$$

Two strategies are indistinguishable if the observable differences in any two traces do not result in observable differences in the behaviors of the two strategies. Hence, the observable parts of the interaction intents of the environment do not reveal any confidential information about the communications performed by the system.

Definition 14. *A concurrent statement CS is* information-flow secure *under the channel policy environment $\mathcal{C}$ and the set Ξ of strategies, denoted by $Sec^{\mathcal{C}}_{\Xi}(CS)$, iff*

$$\begin{aligned} \forall \xi_1, \xi_2 \in \Xi : \forall \mathcal{G} \in \mathcal{P}(Pr) : \xi_1 \overset{\mathcal{G}}{=} \xi_2 \Rightarrow \\ \forall tr_1 \in \textit{traces-of}_{\xi_1}(CS) : \\ \exists tr_2 \in \textit{traces-of}_{\xi_2}(CS) : \lfloor tr_1 \rfloor^{\mathcal{G}} = \lfloor tr_2 \rfloor^{\mathcal{G}} \end{aligned}$$

Hence, a concurrent statement is information flow secure, if under indistinguishable strategies, the observations of its traces are identical. That is, under an

environment that may contain confidential information, but does not leak confidential information by itself (as expressed by the indistinguishability of the two given strategies), the execution of the system in the environment does not leak the confidential information obtainable from the environment.

We have the following theorem on the soundness of our security type system.

Theorem 1. *For* $CS = p_1 : S_1 || \ldots || p_n : S_n$, $\vec{\Phi}$ *satisfying* $\forall k \in \{1, \ldots, n\} : asst(\Phi_k, CS, k)$, *and* $\vec{\Lambda}$ *satisfying* $\forall k \in \{1, \ldots, n\} : live(\Lambda_k, CS[k])$, *if* $\mathcal{C}, \vec{\mathcal{V}} \vdash^{\vec{\Phi}}_{\vec{\Lambda}} CS$ *can be derived, then we have* $Sec^{\mathcal{C}}_{\Xi}(CS)$ *for each set* Ξ *of strategies.*

The proof of this theorem can be found in Appendix C. The theorem says: If a concurrent statement is well-typed with proper knowledge of the values arising in the execution of the concurrent statement, and the live variables, then the concurrent statement is information-flow secure. That is, the execution of the concurrent statement does not leak information beyond what is permitted by the globally dependent flow policies for the communication lines.

7 Static Analyses of Values

We devise two static analyses that provide proper information about the values of program variables to our security type system. More concretely, the analyses provide the appropriate assertion environments for the security type system that guarantee the soundness of the type system:

- The first analysis gathers global information about the values of variables, and about the branching decisions that have most recently been taken.
- The second analysis generates local information associating the most recent branching decisions of a process with logical assertions about the values of variables of that process.
- An assertion environment is then obtained by combining the global information and the local information provided by the two analyses, respectively.

7.1 Global Analysis

The core of the global analysis computes two mappings $E : Pt^* \to \mathcal{P}(Eq_{CS})$ and $B : Pt^* \to K \to \mathcal{P}(\{tt, ff, ?\})$. The types of E and B are explained below.

The mapping E provides information about the set of equalities of the form $x@p = [e]@p'$ that must hold at each combination of program points of the different processes (as represented by a list of program points). An equality of this form represents that the variable x of the process running on behalf of principal p is equal to the expression e of the process running on behalf of principal p'. The set of all such equalities of concern, Eq_{CS}, consists of the equalities between expressions potentially sent in a communication and the corresponding receiving variables, as formalized in the middle part of Table 1.

The mapping B provides information about the branching decision that may have been made at each program point, when the processes are jointly at program

$$\frac{\vec{\imath} = fst(S_1) \ldots fst(S_n)}{(E(\vec{\imath}), B(\vec{\imath})) \sqsupseteq (Eq^{\star}_{CS},\ \lambda\kappa.\{?\})}$$

$$\frac{(\vec{\imath}, \{(i, c!\vec{e}), (j, c?\vec{x})\}, \vec{\imath'}) \in E_{CS}}{(E(\vec{\imath'}), B(\vec{\imath'})) \sqsupseteq ([E(\vec{\imath})]^{p_j}_{c?\vec{x}} \cup \bigcup_r \{x_r@p_j = [e_r]@p_i\},\ B(\vec{\imath}))}$$

$$\frac{(\vec{\imath}, \{(i, \mathsf{brt})\}, \vec{\imath'}) \in E_{CS}}{(E(\vec{\imath'}), B(\vec{\imath'})) \sqsupseteq (E(\vec{\imath}),\ B(\vec{\imath})[\imath_i@p_i \mapsto \{tt\}])} \qquad \frac{(\vec{\imath}, \{(i, \mathsf{brf})\}, \vec{\imath'}) \in E_{CS}}{(E(\vec{\imath'}), B(\vec{\imath'})) \sqsupseteq (E(\vec{\imath}),\ B(\vec{\imath})[\imath_i@p_i \mapsto \{ff\}])}$$

$$\frac{(\vec{\imath}, \{(i, sa)\}, \vec{\imath'}) \in E_{CS} \quad sa \notin \{\mathsf{brt}, \mathsf{brf}\}}{(E(\vec{\imath'}), B(\vec{\imath'})) \sqsupseteq ([E(\vec{\imath})]^{p_i}_{sa},\ B(\vec{\imath}))}$$

$$Eq_{CS} \triangleq \{x_k@p_j = [e_k]@p_i \mid 1 \leq k \leq |\vec{x}| \wedge \exists i, j, c, \vec{\imath}, \vec{\imath'} : (\vec{\imath}, \{(i, c!\vec{e}), (j, c?\vec{x})\}, \vec{\imath'}) \in E_{CS}\}$$
$$Eq^{\star}_{CS} \triangleq Eq_{CS} \cap \{x@p = [e]@p' \mid m_{\star}(x) = [\![e]\!]_{m_{\star}}\}$$

$$[Eq]^p_{sa} \triangleq \{\phi \in Eq \mid \forall x \in upd(sa) : x@p \notin fvs(\phi)\}$$
$$fvs(x@p = [e]@p') \triangleq \{x@p\} \cup \{x'@p' \mid x' \text{ is a variable in } e\}$$
$$upd(sa) \triangleq \begin{cases} \{x\} & \text{if } sa = x \leftarrow e \\ \{\vec{x}\} & \text{if } sa = c?\vec{x} \\ \emptyset & \text{otherwise} \end{cases}$$

Table 1. The Derivation of the Constraints for the Analysis of Available Equalities

points in a given list. The value tt represents a branching decision to the true branch of a conditional or looping statement. The value ff represents a branching decision to the false branch of a conditional or looping statement. The value ? represents that no branching decision is made at a program point because it has not been reached. The set $K \triangleq \{\imath@p \mid \imath \in Pt \wedge p \in Pr\}$ is the set of program points decorated with principals. These principals are used for the technical purpose of distinguishing between the program points of different statements.

The computation of the mappings E and B is performed by computing the least solution of a set of constraints – the least set of constraints that can be derived using the inference rules in the top part of Table 1. In Table 1, the relation $\sqsupseteq$ is defined on pairs in the set $\mathcal{P}(Eq_{CS}) \times (K \to \mathcal{P}(\{tt, ff, ?\}))$, as subset inclusion for the first component, and pointwise superset inclusion for the second component.

In Table 1, the first rule constrains the equalities that hold when all processes are in their initial program points to the equalities in Eq_{CS} that hold in the initial memory. This set of equalities is denoted by $Eq^{\star}_{CS}$. The second rule expresses that after an inter-process communication, the equalities containing potentially updated variables are killed, while the equalities between these variables and

the expressions whose values they newly receive are generated. In this rule, the notation $[Eq]^p_{sa}$ is used. This notation (as formally defined in the bottom part of Table 1) represents the subset of equalities in Eq that do not contain variables updated in the syntactical action sa. The two rules on the third row state that after the syntactical action brt (resp. brf) at the program point $\imath_i$, taking the true branch (resp. false branch) becomes the (only possible) most recent branching decision at $\imath_i$. The last rule expresses that after an action that is performed by one single process, the equalities containing variables that are potentially updated are killed. No equalities are generated, because the global analysis does not keep information about the new values of the updated variables (this is impossible, e.g., when the update is by receiving values from the environment). The lost precision is partly recovered by our local analysis in the next section, because detailed information about values is kept track of in the local analysis.

The mappings E and B are parameterized over lists of program points. We define the mapping $\mathcal{A}_{\mathrm{G}}^{CS,k}$ that gives information about values and most recent branching decisions at each single program point in the k-th statement.

$$\mathcal{A}_{\mathrm{G}}^{CS,k}(\imath) \triangleq \bigsqcup\{(E(\vec{\imath'}), B(\vec{\imath'})) \mid \imath'_k = \imath \wedge \mathit{may\text{-}step}(CS, \vec{\imath'}, k)\}$$

Hence, $\mathcal{A}_{\mathrm{G}}^{CS,k}(\imath)$ is defined as the least upper bound of all pairs $(E(\vec{\imath'}), B(\vec{\imath'}))$ such that the k-th program point in $\vec{\imath'}$ is $\imath$, and a further step can be taken by the k-th process, when the processes in an execution of CS are jointly at the program points in $\vec{\imath'}$.

For the formulation of a correctness result of the global analysis, we use the function $\mathit{last\text{-}at} : \mathit{Act}^* \times \mathit{Pr} \times \mathit{Pt} \to \{\mathit{tt}, \mathit{ff}, ?\}$ to retrieve the most recent branching decisions in a list of actions. We define $\mathit{last\text{-}at}(\alpha_1 \dots \alpha_n, p, \imath)$ as tt if $\exists k \in \{1, \dots, n\} : \alpha_k = \mathsf{brt}^{p,\imath} \wedge \forall i \in \{k+1, \dots, n\} : \alpha_i \notin \{\mathsf{brt}^{p,\imath}, \mathsf{brf}^{p,\imath}\}$, as ff if $\exists k \in \{1, \dots, n\} : \alpha_k = \mathsf{brf}^{p,\imath} \wedge \forall i \in \{k+1, \dots, n\} : \alpha_i \notin \{\mathsf{brt}^{p,\imath}, \mathsf{brf}^{p,\imath}\}$, and as ? if $\forall k \in \{1, \dots, n\} : \alpha_i \notin \{\mathsf{brt}^{p,\imath}, \mathsf{brf}^{p,\imath}\}$.

We have the following theorem about the correctness of the analysis.

Lemma 3. *If $CS = p_1 : S_1 || \dots || p_n : S_n$, $tr \in \mathit{traces\text{-}of}_\xi(CS)$ for some $\xi \in \Xi$, $\mathit{last}(tr) = \langle\langle p_1, S'_1, m'_1\rangle, \dots, \langle p_n, S'_n, m'_n\rangle\rangle$, $\mathit{may\text{-}step}(CS, \mathit{fst}(S'_1) \dots \mathit{fst}(S'_n), k)$, and $\mathcal{A}_{\mathrm{G}}^{CS,k}(\mathit{fst}(S'_k)) = (Eq, f)$, then*

- *if $(x@p_i = [e]@p_j) \in Eq$, then we have $m'_i(x) = [\![e]\!]_{m'_j}$, and*
- *$\forall \imath \in Pt : \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in f(\imath@p_k)$.*

This lemma says that each equality collected by the global analysis holds in the actual execution, and each (most recent) branching decision in the analysis result faithfully reflects the actual (most recent) branching decision. The proof of this lemma can be found in Appendix B.1.

7.2 Local Analysis

We define a local analysis that associates most recent branching decisions with assertions about the values of variables. In our analysis, a set of pairs (μ, φ) is

$$\Psi \vdash D, {}^{\iota}\mathsf{skip}, D \;\text{ if } \Psi(\iota) = D$$

$$\Psi \vdash \{(\mu, \varphi[e/x]) \mid (\mu, \varphi) \in D\}, x :=^{\iota} e, D \;\text{ if } \Psi(\iota) = \{(\mu, \varphi[e/x]) \mid (\mu, \varphi) \in D\}$$

$$\Psi \vdash D, {}^{\iota}\mathsf{send}(c, \vec{e}), D \;\text{ if } \Psi(\iota) = D$$

$$\Psi \vdash \{(\mu, \forall \vec{v}.\varphi[\vec{v}/\vec{x}]) \mid (\mu, \varphi) \in D\}, {}^{\iota}\mathsf{recv}(c, \vec{x}), D \;\text{ if } \Psi(\iota) = \{(\mu, \forall \vec{v}.\varphi[\vec{v}/\vec{x}]) \mid (\mu, \varphi) \in D\}$$

$$\frac{\Psi \vdash D, S_1, D'' \qquad \Psi \vdash D'', S_2, D'}{\Psi \vdash D, S_1; S_2, D'}$$

$$\frac{\begin{array}{l}\Psi \vdash \{(\mu[\iota \mapsto tt], \varphi \wedge e) \mid (\mu, \varphi) \in D\}, S_1, D_1 \\ \Psi \vdash \{(\mu[\iota \mapsto ff], \varphi \wedge \neg e) \mid (\mu, \varphi) \in D\}, S_2, D_2\end{array} \;\text{ if } \Psi(\iota) = D}{\Psi \vdash D, {}^{\iota}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, D_1 \cup D_2}$$

$$\frac{\Psi \vdash \{(\mu[\iota \mapsto tt], \varphi \wedge e) \mid (\mu, \varphi) \in D\}, S, D}{\Psi \vdash D, {}^{\iota}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}, \{(\mu[\iota \mapsto ff], \varphi \wedge \neg e) \mid (\mu, \varphi) \in D\}} \;\text{ if } \Psi(\iota) = D$$

$$\frac{\Psi \vdash D_1', S, D_2'}{\Psi \vdash D_1, S, D_2} \;\text{ if } D_1 \rightsquigarrow D_1' \wedge D_2' \rightsquigarrow D_2$$

where $D_1 \rightsquigarrow D_2$ iff

$$\{\mu \mid \exists \varphi : (\mu, \varphi) \in D_1\} \subseteq \{\mu \mid \exists \varphi : (\mu, \varphi) \in D_2\} \wedge$$
$$\forall \mu, \varphi_1 : ((\mu, \varphi_1) \in D_1 \Rightarrow \exists \varphi_2 : (\mu, \varphi_2) \in D_2 \wedge \varphi_1 \Rightarrow \varphi_2)$$

Fig. 10. Association of Branching Decisions with Value Assertions

identified at each program point. Here, $\mu : Pt \to \{tt, ff, ?\}$ maps each program point to a single branching decision. On the other hand, φ is an assertion about the values of program variables.

The syntax for assertions is given below.

$$\begin{aligned} \varphi &::= \mathsf{true} \mid tm\ cop\ tm \mid \neg\varphi \mid \varphi \wedge \varphi \\ tm &::= n \mid x \mid f(tm, ..., tm) \end{aligned}$$

An assertion is true for logical truth, a comparison of two terms $tm\ cop\ tm$, the negation of an assertion, or the conjunction of two assertions. A term is a numeral n, a variable x, or a function application on terms. We denote the set of all assertions by $Asst$.

The local analysis is specified by a set of inference rules (Fig. 10) for the judgment $\Psi \vdash D_1, S, D_2$. Here, $\Psi : Pt \to \mathcal{P}((Pt \to \{tt, ff, ?\}) \times Asst)$ is the environment recording the analysis data at each program point, S is a statement, and D_1 and D_2 are the analysis data before and after the statement S. In more detail, D_1 and D_2 are both sets of pairs (μ, φ).

In each rule of Fig. 10 whose conclusion contains an explicit program point ι, it is specified that the analysis data $\Psi(\iota)$ obtained from the environment Ψ is equal to the analysis data before the statement at ι. In the analysis data, the second component of each pair obeys standard Hoare logic rules. The first

component of each pair in the analysis data is updated in the beginning of each true branch of an if statement, mapping the program point of the if statement to tt. This records a most recent branching decision at $\imath$ of taking the true branch. Note that the recording of this branching decision is accompanied by a transformation of the assertion φ. In the beginning of each false branch of an if statement, an analogous treatment mapping the program point of the if to ff is applied. The first component of each pair in the analysis data is updated in the beginning of each loop body, mapping the program point of the loop (identifying the conditional test of the loop) to tt. Finally, the first component of each pair in the analysis data is updated in the end of each loop, mapping the program point of the loop to ff.

We say that the mapping $\mathcal{A}^S_{\mathrm{L}}$ is a result of the local analysis for the statement S if there exists some Ψ such that $\Psi \vdash \{(\lambda\imath.?, \varphi)\}, S, D$ is derivable, with φ holding in the initial memory $m_\star$, $\mathcal{A}^S_{\mathrm{L}}(\imath) = \Psi(\imath)$, and $\mathcal{A}^S_{\mathrm{L}}(\iota_{\mathrm{f}}) = D$.

Lemma 4. *If $\mathcal{A}^S_{\mathrm{L}}$ is a result of the local analysis for S, and for each $i \in \{1, \ldots, n\}$, it can be derived that $\langle p, S_{i-1}, m_{i-1}\rangle \xrightarrow{\alpha_{i-1}} \langle p, S_i, m_i\rangle$, where $S_0 = S$ and $m_0 = m_\star$, then there exist some μ, and some φ, such that $(\mu, \varphi) \in \mathcal{A}^S_{\mathrm{L}}(fst(S_n))$, $\forall \imath : \mu(\imath) = last\text{-}at(\alpha_0 \ldots \alpha_{n-1}, p, \imath)$, and the assertion φ holds in the memory m_n.*

This lemma says that the pairs (μ, φ) resulting from the local analysis at different program points is a sound over-approximation of the actual computation, and the association of both components of these pairs is correct. The proof of this lemma can be found in Appendix B.2.

7.3 Combining the Global and Local Analyses

We show that an appropriate assertion environment guaranteeing the soundness of our security type system (cf. Section 5) can be obtained by combining the results of the global analysis and the local analysis.

Theorem 2. *For all concurrent statements $CS = p_1 : S_1 || \ldots || p_n : S_n$, $k \in \{1, \ldots, n\}$, and $\Phi \in Pt \to Asst$, if for each $\imath \in Pt$, it holds that $\Phi(\imath) = (\bigwedge Eq) \wedge (\bigvee\{\, [\varphi]@p_k \mid \exists \mu : (\mu, \varphi) \in D \wedge \forall \imath' : \mu(\imath') \in f(\imath'@p_k)\})$, where Eq, f, and D satisfy $\mathcal{A}^{CS,k}_{\mathrm{G}}(\imath) = (Eq, f)$ and $\mathcal{A}^{S_k}_{\mathrm{L}}(\imath) = D$, then we have $asst(\Phi, CS, k)$.*

According to the theorem, an assertion environment guaranteeing the soundness of the security type system is one that asserts all the equalities known to hold at each program point, as well as the disjunction of all the local assertions paired with the branching decisions that are possible for the program point. The proof of this theorem can be found in Appendix B.3.

Hence, the analyses developed in this section can be used to infer correct information about the values of variables that supports sound security typing under globally dependent flow policies. It can be seen from $\Phi(\imath)$ in Theorem 2 that the information about the most recent branching decisions as obtained from the global analysis constrains the local assertions that are possible, thereby improving the precision of the overall analysis about values.

Fact 2 *For the concurrent statement CS_{SPF} of the stateful packet filtering system, the list $\vec{\Phi}$ (with $|\vec{\Phi}| = 3$) of assertion environments produced by our static analyses satisfies the condition required in Fact 1.*

Hence, our static analyses providing information about the values of variables are not only sound, but also sufficiently precise for supporting the security typing of the stateful packet filtering system. This results in an overall analysis of the system proving that it is information-flow secure under the globally dependent flow policies to be enforced.

8 Related Work

8.1 Dependent Information-Flow Policies

Numerous attempts have been made to enforce information-flow policies dependent on values, permissions, or other security parameters – to address the needs for security in diverse practical scenarios. In [53], a security type system is proposed to enforce policies in the Decentralized Label Model [38] that may depend on run-time principals. In [8–10], security policies depending on lock variables that reflect roles and access rights are enforced. In [3, 2], value-dependent flow policies are supported by a combination of value assertions and relational assertions. In [30, 31], type-based enforcement of value-dependent flow policies is developed for database-like systems. In [42, 29, 41], message-passing systems are considered, where the security policy of a message field may depend on the other fields of the same message, and the security policy of a variable may depend on the other variables of the same process. In [28], flow policies dependent on the future values of program variables are considered. In [34], a symbolic executor is developed to enforce flow policies depending on the computation history. In [37, 36], a security type system is developed for flow-sensitive, value-dependent policies in a shared-memory concurrent language, such that global value-dependency is permitted. In [26], a two-pass approach is developed to enforce flow and value-sensitive policies in a sequential language, where a transformation enables the treatment of flow-sensitive policies in a flow-insensitive manner, and increases the precision of specifying value-dependency. In [12], a security type system is developed to enforce information-flow policies dependent on permissions in a sequential language with function calls and explicit branching on permissions.

Most of the developments above have been carried out on sequential languages, with the exception of [37, 42, 29, 41]. The developments [42, 29, 41] are carried out for message-passing systems, without enabling global dependencies in the flow policies. The security type system of [37] enforces flow policies dependent on the values of any variables in a shared-memory concurrent program. In addition, flow-sensitive security types [22] are supported – the policies may vary at different program points in a sequential composition. In comparison with [37], we enforce globally dependent flow policies in message-passing concurrent systems. The policy of a message field may depend on the other fields of the message, as well as on variables of any process, and the policy of a variable may

depend on variables of any process. We differentiate between the presence and content of message-passing communication, and we obtain how variables across processes relate (due to communication) through dedicated static analyses. We do not support flow-sensitive policies (in the style of [22]) like [37] does. On the other hand, we enable the specialization of flow policies at conditional tests (which is termed the scoped specialization of flow policies), such that the flow policies may alter on entrance to a conditional branch or a loop. In addition, we permit a high level of precision in security type checking by exploiting the live variables information at each program point.

In [26], a form of liveness information is exploited in the information-flow analysis, increasing the precision of the analysis. Whereas the liveness analysis needed for our development is a standard live variables analysis (e.g., [40]), the liveness analysis required in [26] regards all variables in the policy of a live variable as live (which results in enlarged sets of live variables). Whereas our security typing rule for assignments explicitly requires the security policies of all live variables not to be weakened, the typing rule for assignments in [26] requires the variable being assigned not to be depended upon by the policies of any live variables. While the latter difference has roots in similar rationale, our use of liveness information seems to result in more overall permissiveness in security typing. However, we do not support flow-sensitive policies like [26] does.

We cater for interactive programs (e.g., [46]) by a seamless integration of environment strategies (that synchronously communicate with processes of the system) in our model of computation. Information-flow security under explicit environment strategies is not addressed in any of the aforementioned developments enforcing dependent flow policies.

8.2 Information-Flow Security for Concurrent Systems

One of the main features for which our development differs from most prior work on the enforcement of dependent flow policies is concurrency. For two decades, concurrency has been a crucial feature to be addressed in information-flow security research. The following discussion is non-exhaustive about prior work on information-flow security for concurrent systems.

In [51], it is observed that concurrent programs admit an additional class of information leakage (later known as internal timing leakage) than their sequential counterparts, and the first sound security type system for concurrent programs is developed. In [50], a probabilistic security property is proposed to cater for the scheduling of threads, and a possibilistic property is identified and enforced, to enforce the probabilistic property under all secure schedulers. In [7], syntactical scheduler models (expressed in program code) are considered. In [35], distributed systems are considered, where all threads at the same place share variables. In [33, 4], rely-guarantee-style reasoning is developed for the information-flow security of concurrent programs, resulting in sound compositional reasoning about information-flow security with high precision. In [23], a type-and-effect system is developed to secure concurrent programs against internal timing leakage, while permitting benign races on the public variables. Apart from the aforementioned

work that deals with the interleaving semantics of concurrency, treatment of true concurrency exists. For instance, in [6] and [5], expressive security policies and properties are studied for Petri nets and event structures, respectively.

8.3 Information-Flow Security for Message-Passing Systems

Our work deals with value-dependent flow policies in message-passing systems. Although value-dependency is rarely addressed for message-passing systems, the information-flow security of such systems in general has been extensively studied. The following discussion of related work in this regard is again non-exhaustive.

In [47, 15–17, 27], security properties are studied for confidentiality and integrity in process calculi. In [20, 45, 25, 21], security type systems are developed for variants of the π-calculus. In [11], a fine-grained information-flow type system is developed for a language with session-based communication [52]. In [48], a security property and a security type system are developed for a concurrent language with asynchronous message-passing communication, with support for compositional reasoning under arbitrary environments of systems. In [19], compositional reasoning techniques are developed for the information-flow security of component-based systems where the components communicate via synchronous message-passing. The potential environments of each component is explicitly considered in secure composition.

9 Conclusion

In answer to the challenge of securing concurrent, message-passing systems under information-flow policies with global dependencies, we develop a solution consisting of a security type system and two supportive static analyses. The security guarantee of our solution is articulated via a noninterference-like property for open systems, and established via formal proofs. The security type system and the static analyses are implemented in a proof-of-concept tool. Our verification technique is illustrated using the example of a stateful packet filtering system where the flow of information depends on how the headers of the messages from different modules of the system correlate.

Our solution targets a computation model rich enough to cover both interprocess communication à la concurrent programs and environmental interaction à la interactive programs. The solution features improvement in precision/permissiveness in multiple aspects. In the security type system, the strength of the security policies is required to be preserved only for the live variables (as opposed to all variables). The policies for variables are specialized when entering the branches of a `if` statement, or the body of a `while` loop, which reduces the potential changes of the policies that could negatively affect the permissiveness of the type system. Finally, the two supportive static analyses are coupled via information about the most recent branching decisions in the execution of a system, improving the precision of their combination.

Directions for future work include support for type inference with the use of globally dependent flow policies, and support for a richer programming language for the individual processes in a system.

Acknowledgement. This work was partially supported by the National Natural Science Foundation of China (61876111, 61572331, 61602325), and the National Key R&D Plan of China (2017YFC0806700).

References

1. The OCaml programming language. http://ocaml.ocamlpro.com/.
2. Torben Amtoft, Josiah Dodds, Zhi Zhang, Andrew W. Appel, Lennart Beringer, John Hatcliff, Xinming Ou, and Andrew Cousino. A certificate infrastructure for machine-checked proofs of conditional information flow. In *Proceedings of the First International Conference on Principles of Security and Trust (POST)*, pages 369–389, 2012.
3. Torben Amtoft, John Hatcliff, Edwin Rodríguez, Robby, Jonathan Hoag, and David A. Greve. Specification and checking of software contracts for conditional information flow. In *FM 2008: Formal Methods, 15th International Symposium on Formal Methods*, pages 229–245, 2008.
4. Aslan Askarov, Stephen Chong, and Heiko Mantel. Hybrid monitors for concurrent noninterference. In *Proceedings of IEEE 28th Computer Security Foundations Symposium (CSF)*, pages 137–151, 2015.
5. Paolo Baldan, Alessandro Beggiato, and Alberto Lluch-Lafuente. Many-to-many information flow policies. In *Coordination Models and Languages - 19th IFIP WG 6.1 International Conference (COORDINATION)*, pages 159–177, 2017.
6. Luca Bernardinello, Görkem Kilinç, and Lucia Pomello. Non-interference notions based on reveals and excludes relations for petri nets. *T. Petri Nets and Other Models of Concurrency*, 11:49–70, 2016.
7. Gérard Boudol and Ilaria Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Computer Science*, 281(1-2):109–130, 2002.
8. Niklas Broberg and David Sands. Flow locks: Towards a core calculus for dynamic flow policies. In *Programming Languages and Systems, 15th European Symposium on Programming (ESOP)*, pages 180–196, 2006.
9. Niklas Broberg and David Sands. Paralocks: Role-based information flow control and beyond. In *Proceedings of the 37th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 431–444, 2010.
10. Niklas Broberg, Bart van Delft, and David Sands. Paragon - practical programming with information flow control. *Journal of Computer Security*, 25(4-5):323–365, 2017.
11. Sara Capecchi, Ilaria Castellani, and Mariangiola Dezani-Ciancaglini. Typing access control and secure information flow in sessions. *Information and Computation*, 238:68–105, 2014.
12. Hongxu Chen, Alwen Tiu, Zhiwu Xu, and Yang Liu. A permission-dependent type system for secure information flow analysis. In *31st IEEE Computer Security Foundations Symposium (CSF)*, pages 218–232, 2018.
13. Ellis S. Cohen. Information transmission in computational systems. In *Proceedings of the Sixth Symposium on Operating System Principles (SOSP)*, pages 133–139, 1977.

14. Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: an efficient SMT solver. In *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference (TACAS)*, pages 337–340, 2008.
15. Riccardo Focardi and Roberto Gorrieri. Classification of security properties (part I: information flow). In *Foundations of Security Analysis and Design, Tutorial Lectures (FOSAD)*, pages 331–396, 2000.
16. Riccardo Focardi and Sabina Rossi. Information flow security in dynamic contexts. *Journal of Computer Security*, 14(1):65–110, 2006.
17. Simon N. Foley. A nonfunctional approach to system integrity. *IEEE Journal on Selected Areas in Communications*, 21(1):36–43, 2003.
18. Joseph A. Goguen and José Meseguer. Security policies and security models. In *1982 IEEE Symposium on Security and Privacy (S&P)*, pages 11–20, 1982.
19. Simon Greiner and Daniel Grahl. Non-interference with what-declassification in component-based systems. In *IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 253–267, 2016.
20. Matthew Hennessy and James Riely. Information flow vs. resource access in the asynchronous pi-calculus. *ACM Transactions on Programming Languages and Systems*, 24(5):566–591, 2002.
21. Kohei Honda and Nobuko Yoshida. A uniform type structure for secure information flow. *ACM Transactions on Programming Languages and Systems*, 29(6):31, 2007.
22. Sebastian Hunt and David Sands. On flow-sensitive security types. In *Proceedings of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 79–90, 2006.
23. Aleksandr Karbyshev, Kasper Svendsen, Aslan Askarov, and Lars Birkedal. Compositional non-interference for concurrent programs via separation and framing. In *Proceedings of the 7th International Conference on Principles of Security and Trust, (POST)*, pages 53–78, 2018.
24. George E. Karniadakis and Robert M. Kirby. *Parallel Scientific Computing in C++ and MPI - A Seamless Approach to Parallel Algorithms and their Implementation*. Cambridge University Press, 2003.
25. Naoki Kobayashi. Type-based information flow analysis for the pi-calculus. *Acta Informatica*, 42(4-5):291–347, 2005.
26. Peixuan Li and Danfeng Zhang. Towards a flow- and path-sensitive information flow analysis. In *30th IEEE Computer Security Foundations Symposium (CSF)*, pages 53–67, 2017.
27. Ximeng Li, Flemming Nielson, and Hanne Riis Nielson. Factorization of behavioral integrity. In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS)*, pages 500–519, 2015.
28. Ximeng Li, Flemming Nielson, and Hanne Riis Nielson. Future-dependent flow policies with prophetic variables. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS@CCS)*, pages 29–42, 2016.
29. Ximeng Li, Flemming Nielson, Hanne Riis Nielson, and Xinyu Feng. Disjunctive information flow for communicating processes. In *Proceedings of the 10th International Symposium on Trustworthy Global Computing (TGC)*, pages 95–111, 2015.
30. Luísa Lourenço and Luís Caires. Information flow analysis for valued-indexed data security compartments. In *Trustworthy Global Computing - 8th International Symposium (TGC)*, pages 180–198, 2013.
31. Luísa Lourenço and Luís Caires. Dependent information flow types. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 317–328. ACM, 2015.

32. Heiko Mantel. Information flow and noninterference. In *Encyclopedia of Cryptography and Security, 2nd Ed.*, pages 605–607. 2011.
33. Heiko Mantel, David Sands, and Henning Sudbrock. Assumptions and guarantees for compositional noninterference. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium (CSF)*, pages 218–232, 2011.
34. Kristopher K. Micinski, Jonathan Fetter-Degges, Jinseong Jeon, Jeffrey S. Foster, and Michael R. Clarkson. Checking interaction-based declassification policies for android using symbolic execution. In *Proceedings of the 20th European Symposium on Research in Computer Security (ESORICS)*, pages 520–538, 2015.
35. Stefan Muller and Stephen Chong. Towards a practical secure concurrent language. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 57–74, 2012.
36. Toby C. Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. Compositional security-preserving refinement for concurrent imperative programs. *Archive of Formal Proofs*, 2016.
37. Toby C. Murray, Robert Sison, Edward Pierzchalski, and Christine Rizkallah. Compositional verification and refinement of concurrent value-dependent noninterference. In *Proceedings of the IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 417–431, 2016.
38. Andrew C. Myers. *Mostly-static decentralized information flow control.* PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1999.
39. Flemming Nielson. Program transformations in a denotational setting. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 7(3):359–379, 1985.
40. Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. *Principles of program analysis.* Springer, 1999.
41. Hanne Riis Nielson and Flemming Nielson. Content dependent information flow control. *Journal of Logic and Algebraic Methods in Programming*, 87:6–32, 2017.
42. Hanne Riis Nielson, Flemming Nielson, and Ximeng Li. Hoare logic for disjunctive information flow. In *Programming Languages with Applications to Biology and Security*, pages 47–65, 2015.
43. Gordon D. Plotkin. A structural approach to operational semantics. Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981. Reprinted 1991.
44. Amir Pnueli and Willem P. de Roever. Rendezvous with ADA. Technical report, Rijksuniversiteit Utrecht, 07 1982.
45. François Pottier. A simple view of type-secure information flow in the pi-calculus. In *15th IEEE Computer Security Foundations Workshop (CSFW)*, pages 320–330, 2002.
46. Willard Rafnsson, Daniel Hedin, and Andrei Sabelfeld. Securing interactive programs. In *Proceedings of the 25th IEEE Computer Security Foundations Symposium (CSF)*, pages 293–307, 2012.
47. A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop, (CSFW)*, pages 228–238, 1999.
48. Andrei Sabelfeld and Heiko Mantel. Securing communication in a concurrent language. In *Proceedings of the 9th International Symposium on Static Analysis (SAS)*, pages 376–394, 2002.
49. Andrei Sabelfeld and Andrew C. Myers. Language-based information-flow security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, 2003.

50. Andrei Sabelfeld and David Sands. Probabilistic noninterference for multi-threaded programs. In *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, pages 200–214, 2000.
51. Geoffrey Smith and Dennis M. Volpano. Secure information flow in a multi-threaded imperative language. In *Proceedings of the 25th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 355–364, 1998.
52. Kaku Takeuchi, Kohei Honda, and Makoto Kubo. An interaction-based language and its typing system. In *PARLE '94: Parallel Architectures and Languages Europe, 6th International (PARLE)*, pages 398–413, 1994.
53. Stephen Tse and Steve Zdancewic. Run-time principals in information-flow type systems. In *Proceedings of the 2004 IEEE Symposium on Security and Privacy (S&P)*, pages 179–193, 2004.
54. Dennis M. Volpano, Cynthia E. Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *Journal of Computer Security*, 4(2/3):167–188, 1996.
55. J. Todd Wittbold and Dale M. Johnson. Information flow in nondeterministic systems. In *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy (S&P)*, pages 144–161, 1990.
56. Yongwang Zhao, David Sanán, Fuyuan Zhang, and Yang Liu. Reasoning about information flow security of separation kernels with channel-based communication. In *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 791–810, 2016.

A Additional Definitions and Specifications

Notation. In this article, we write $[a_1, \ldots, a_n]$ for a list whose elements are a_1, ..., a_n, and use $\vec{a}$ as shorthand notation for such a list. For a list $l = [a_1, \ldots, a_n]$ where $n \geq 1$, and $i \in \{1, \ldots, n\}$, we write $l[i \mapsto b]$ for the list $[a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n]$, write $last(l)$ for the element a_n, and write $|l|$ for the length n of l. For two lists l_1 and l_2, we write $l_1.l_2$ for the concatenation of l_1 and l_2. For an element a and a list l, we abuse notation to write $a.l$ and $l.a$ for $[a].l$ and $l.[a]$, respectively. For a function $f : A \to B$, we write $f[a \mapsto b]$, where $a \in A$ and $b \in B$, for the function f' with $f'(x) = f(x)$ whenever $x \neq a$, and $f'(a) = b$, and write $f[(a_i \mapsto b_i)_{i \in \{1,\ldots,n\}}]$, where $\forall i \in \{1, \ldots, n\} : a_i \in A \wedge b_i \in B$, for the function $((f[a_1 \mapsto b_1]) \ldots)[a_n \mapsto b_n]$, omitting the range of i in the subscript when this is unimportant for the discussion or clear from the context.

Definition of the Function *fst*. The function $fst : Stmt \to Pt$ (used e.g., in Fig. 4) is defined by $fst(\text{`}\mathsf{skip}) = \imath$, $fst(x := \text{`}e) = \imath$, $fst(\text{`}\mathsf{send}(c, \vec{e})) = \imath$, $fst(\text{`}\mathsf{recv}(c, \vec{x})) = \imath$, $fst(S_1; S_2) = fst(S_1)$, $fst(\text{`}\mathsf{if}\ e\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}) = \imath$, $fst(\text{`}\mathsf{while}\ e\ \mathsf{do}\ S\ \mathsf{od}) = \imath$, and $fst(\text{`}\mathsf{stop}) = \imath$.

Definition of the Successor Relation. In Fig. 11, we define the local successor relation $succ(S, \imath')$ used to generate the program graph of a concurrent statement (cf. Section 7 and Fig. 4), where S is a labeled statement, and $\imath'$ is the program point immediately after S.

$$
\begin{aligned}
succ(\text{`}\mathsf{skip}, \imath') &\triangleq \{(\imath, \mathsf{sk}, \imath')\} \\
succ(\text{`}x := e, \imath') &\triangleq \{(\imath, x \leftarrow e, \imath')\} \\
succ(\text{`}\mathsf{send}(c, \vec{e}), \imath') &\triangleq \{(\imath, c!\vec{e}, \imath')\} \\
succ(\text{`}\mathsf{recv}(c, \vec{x}), \imath') &\triangleq \{(\imath, c?\vec{x}, \imath')\} \\
succ(S_1; S_2, \imath') &\triangleq succ(S_1, fst(S_2)) \cup succ(S_2, \imath') \\
succ(\text{`}\mathsf{if}\ b\ \mathsf{then}\ S_1\ \mathsf{else}\ S_2\ \mathsf{fi}, \imath') &\triangleq \{(\imath, \mathsf{brt}, fst(S_1)), (\imath, \mathsf{brf}, fst(S_2))\} \cup \\
&\qquad succ(S_1, \imath') \cup succ(S_2, \imath') \\
succ(\text{`}\mathsf{while}\ b\ \mathsf{do}\ S\ \mathsf{od}, \imath') &\triangleq \{(\imath, \mathsf{brt}, fst(S))\} \cup \{(\imath, \mathsf{brf}, \imath')\} \cup succ(S, \imath)
\end{aligned}
$$

Fig. 11. The Definition of *succ*

Definition of the Interpretation of Globally Dependent Flow Policies. The complete definition of the interpretation of globally dependent flow policies is given in Fig. 12. Part of this definition is presented in Section 4. In Fig. 12, the semantics $[\![cop]\!]$ and $[\![g]\!]$ of comparative operators *cop* and functions g are unspecified. This allows for a good level of generality. We require, however, that $[\![cop]\!]$ and $[\![g]\!]$ should be total, for all policies to have their interpretation.

Results of the Local Analysis on the Filtering Module. The local analysis of Section 7.2 admits the results for the filtering module in the stateful packet filtering system as shown in Fig. 13.

$$
\begin{aligned}
[\![\{p_1,\ldots,p_2\}]\!]_{gcnf,\delta} &\triangleq \{p_1,\ldots,p_2\}\\
[\![\phi \triangleright P]\!]_{gcnf,\delta} &\triangleq \begin{cases} [\![P]\!]_{gcnf,\delta} & \text{if } [\![\phi]\!]_{gcnf,\delta} = tt \\ Pr & \text{otherwise} \end{cases}\\
[\![P_1 \curlywedge P_2]\!]_{gcnf,\delta} &\triangleq [\![P_1]\!]_{gcnf,\delta} \cap [\![P_2]\!]_{gcnf,\delta}
\end{aligned}
$$

$$
\begin{aligned}
[\![\mathsf{true}]\!]_{gcnf,\delta} &\triangleq tt\\
[\![t_1 \ cop \ t_2]\!]_{gcnf,\delta} &\triangleq [\![cop]\!]([\![t_1]\!]_{gcnf,\delta}, [\![t_2]\!]_{gcnf,\delta})\\
[\![\neg\phi]\!]_{gcnf,\delta} &\triangleq \begin{cases} ff & \text{if } [\![\phi]\!]_{gcnf,\delta} = tt \\ tt & \text{if } [\![\phi]\!]_{gcnf,\delta} = ff \end{cases}\\
[\![\phi_1 \wedge \phi_2]\!]_{gcnf,\delta} &\triangleq \begin{cases} tt & \text{if } [\![\phi_1]\!]_{gcnf,\delta} = tt \text{ and } [\![\phi_2]\!]_{gcnf,\delta} = tt \\ ff & \text{otherwise} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
[\![n]\!]_{gcnf,\delta} &\triangleq [\![n]\!]\\
[\![x@p_j]\!]_{\langle\ldots,\langle p_j,S,m\rangle,\ldots\rangle,\delta} &\triangleq m(x)\\
[\![c.j]\!]_{gcnf,\delta} &\triangleq \delta(j)\\
[\![g(t_1,\ldots,t_n)]\!]_{gcnf,\delta} &\triangleq [\![g]\!]([\![t_1]\!]_{gcnf,\delta},\ldots,[\![t_n]\!]_{gcnf,\delta})
\end{aligned}
$$

Fig. 12. The Semantics of Globally Dependent Flow Policies

Semantic Liveness and Live Variables Analysis.

Definition 15. *We define* $m_1 \overset{X}{=} m_2$ *to express* $\forall x \in X : m_1(x) = m_2(x)$.

Definition 16. *We define in-vars*(S) *by in-vars*$(\mathsf{recv}(c,\vec{x})) \triangleq \{\vec{x}\}$, *in-vars*$(S_1; S_2) \triangleq$ *in-vars*(S_1) *and in-vars*$(S) = \emptyset$ *otherwise.*

Definition 17 (Semantic Liveness). *We define live*(Λ, S) *to express if*

$$\langle p, S, m_\star\rangle \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} \langle p, S', m'\rangle \quad (n \geq 0)$$

$\langle p, S', m_1'\rangle \xrightarrow{\alpha} \langle p, S'', m_1''\rangle$, *and* $m_1' \overset{\Lambda(fst(S'))\cup in\text{-}vars(S')}{=} m_2'$, *then there exists some* m_2'' *such that* $\langle p, S', m_2'\rangle \xrightarrow{\alpha} \langle p, S'', m_2''\rangle$ *and* $m_1'' \overset{\Lambda(fst(S''))}{=} m_2''$.

We next define a live variables analysis for a given statement S via the smallest set of equations that can be derived using the following rule.

$$\frac{(\iota_1, sa, \iota_2) \in succ(S, \iota_{\mathrm{f}})}{\Lambda(\iota_1) = (\Lambda(\iota_2) \setminus kill(sa)) \cup gen(sa)} \tag{4}$$

The $kill(sa)$ and $gen(sa)$ in the above are defined below.

$$
\begin{aligned}
kill(\mathsf{sk}) &= \emptyset & gen(\mathsf{sk}) &= \emptyset\\
kill(x \leftarrow e) &= \{x\} & gen(x \leftarrow e) &= fvs(e)\\
kill(c!\vec{e}) &= \emptyset & gen(c!\vec{e}) &= fvs(e)\\
kill(c?\vec{x}) &= \{\vec{x}\} & gen(c?\vec{x}) &= \emptyset\\
kill(\mathsf{brt}) &= \emptyset & gen(\mathsf{brt}) &= \emptyset\\
kill(\mathsf{brf}) &= \emptyset & gen(\mathsf{brf}) &= \emptyset
\end{aligned}
$$

The analysis is such that if Λ is a solution of the set of equations derived using (4), then $live(\Lambda, S)$ holds.

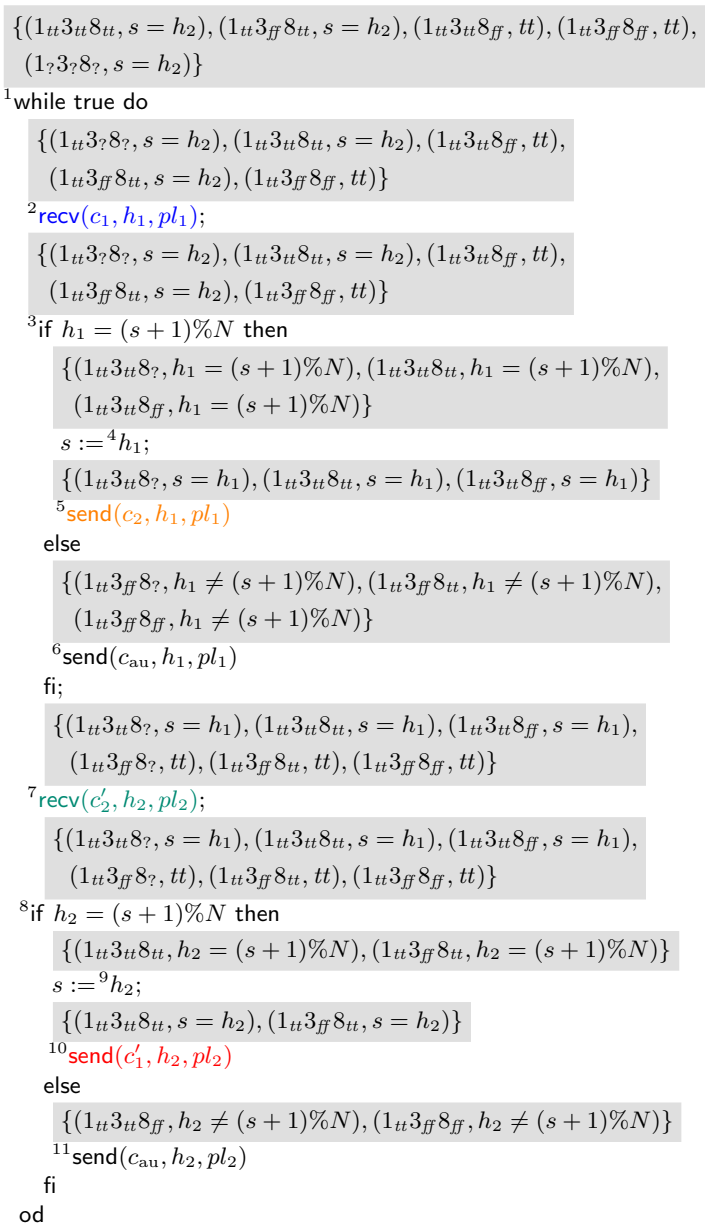

Fig. 13. Result of the Analysis Associating Branching Decisions with Value Assertions for the Filtering Statement

B Proof of Theorem 2

In this appendix, we present the proof of Theorem 2 on the correctness of the assertion environment defined using the results of the global analysis and the local analysis. In Appendix B.1 and Appendix B.2, we give the proofs of Lemma 3 and Lemma 4, respectively. In Appendix B.3, we give the proof of Theorem 2 using Lemma 3 and Lemma 4.

B.1 Proof of Lemma 3

Lemma 5. *If* $(\imath, sa, \imath') \in succ(S, \imath'')$*, and* $\imath' \neq \imath''$*, then for all* $\imath''' \notin pts(S)$*, it holds that* $(\imath, sa, \imath') \in succ(S, \imath''')$*.*

Lemma 6. *If* $(\imath, sa, \imath') \in succ(S, \imath')$*, and* $\imath' \notin pts(S)$*, then* $(\imath, sa, \imath'') \in succ(S, \imath'')$ *for all* $\imath'' \notin pts(S)$*.*

The proof of these two lemmas is by induction on the structure of S.

Lemma 7. *If* $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$*, and* $\langle p, S', m'\rangle \xrightarrow{\alpha'}$*, then for all* $\imath' \notin pts(S)$*, it holds that* $succ(S', \imath') \subseteq succ(S, \imath')$*.*

The proof of this lemma is by induction on the derivation of $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$.

Lemma 8. *If for all* $i \in \{1, \ldots, n+1\}$*, we have* $\langle p, S_{i-1}, m_{i-1}\rangle \xrightarrow{\alpha_i} \langle p, S_i, m_i\rangle$*, and* $\alpha = \alpha_{n+1}$*, then there exists some* sa *such that* $(fst(S_n), sa, fst(S_{n+1})) \in succ(S_0, \iota_{\mathrm{f}})$*, and it holds that*

$$
\begin{aligned}
&(\exists c, \vec{v}, \vec{e} : \alpha = c!\vec{v} \wedge sa = c!\vec{e} \wedge [\![\vec{e}]\!]_{m_n} = \vec{v})\\
&\vee\ (\exists c, \vec{v}, \vec{x} : \alpha = c?\vec{v} \wedge sa = c?\vec{x} \wedge m_{n+1}(\vec{x}) = \vec{v})\\
&\vee\ \alpha = \mathit{brt}^{p, fst(S_n)} \wedge sa = \mathsf{brt}\\
&\vee\ \alpha = \mathit{brf}^{p, fst(S_n)} \wedge sa = \mathsf{brf}\\
&\vee\ \alpha = \tau \wedge (sa = \mathsf{sk} \vee \exists v : \exists e : sa = (v \leftarrow e))
\end{aligned}
$$

Proof. We perform an induction on the number of the derivation steps.

- Base case – suppose the number of derivation steps is 1.
  We have $n = 0$, $i \in \{1\}$, and
  $$\langle p, S_0, m_0\rangle \xrightarrow{\alpha} \langle p, S_1, m_1\rangle \tag{5}$$
  We perform an induction on the derivation of (5). We only display the three representative cases below.
  - Case (5) is derived with the rule for an assignment $x :=^{\imath} e$. We have $\alpha = \tau$ and $sa = (x \leftarrow e)$ for which the conclusion holds.

- Case (5) is derived with the first rule for the sequential composition $S_{\mathrm{a}}; S_{\mathrm{b}}$, due to $\langle p, S_{\mathrm{a}}, m_0\rangle \xrightarrow{\alpha} \langle p, S'_{\mathrm{a}}, m_1\rangle$.
  By the induction hypothesis (for the inner induction), we have some sa with $(\mathit{fst}(S_{\mathrm{a}}), sa, \mathit{fst}(S'_{\mathrm{a}})) \in \mathit{succ}(S_{\mathrm{a}}, \iota_{\mathrm{f}})$ such that α, sa, m_0 and m_1 satisfy the final condition of the lemma. Since $\mathit{fst}(S'_{\mathrm{a}}) \neq \iota_{\mathrm{f}}$, and $\mathit{fst}(S_{\mathrm{b}}) \notin \mathit{pts}(S_{\mathrm{a}})$, we have
  $$(\mathit{fst}(S_{\mathrm{a}}), sa, \mathit{fst}(S'_{\mathrm{a}})) \in \mathit{succ}(S_{\mathrm{a}}, \mathit{fst}(S_{\mathrm{b}}))$$
  by Lemma 5. Hence, $(\mathit{fst}(S_{\mathrm{a}}; S_{\mathrm{b}}), sa, \mathit{fst}(S'_{\mathrm{a}}; S_{\mathrm{b}})) \in \mathit{succ}(S_{\mathrm{a}}; S_{\mathrm{b}}, \iota_{\mathrm{f}})$ with α, sa, m_0 and m_1 satisfying the final condition of the lemma.
- Case (5) is derived using the second rule for the sequential composition $S_{\mathrm{a}}; S_{\mathrm{b}}$, due to $\langle p, S_{\mathrm{a}}, m_0\rangle \xrightarrow{\alpha} \langle p, {}^{\iota_{\mathrm{f}}}\mathsf{stop}, m_1\rangle$.
  By the induction hypothesis (for the inner induction), we have some sa with $(\mathit{fst}(S_{\mathrm{a}}), sa, \iota_{\mathrm{f}}) \in \mathit{succ}(S_{\mathrm{a}}, \iota_{\mathrm{f}})$ such that α, sa, m_0 and m_1 satisfy the final condition of the lemma. We have $\iota_{\mathrm{f}} \notin \mathit{pts}(S_{\mathrm{a}})$, and $\mathit{fst}(S_{\mathrm{b}}) \notin \mathit{pts}(S_{\mathrm{a}})$. Hence, we have
  $$(\mathit{fst}(S_{\mathrm{a}}), sa, \mathit{fst}(S_{\mathrm{b}})) \in \mathit{succ}(S_{\mathrm{a}}, \mathit{fst}(S_{\mathrm{b}}))$$
  by Lemma 6. Therefore, we have $(\mathit{fst}(S_{\mathrm{a}}; S_{\mathrm{b}}), sa, \mathit{fst}(S_{\mathrm{b}})) \in \mathit{succ}(S_{\mathrm{a}}; S_{\mathrm{b}}, \iota_{\mathrm{f}})$ with α, sa, m_0 and m_1 satisfying the final condition of the lemma.

– Inductive case – suppose the number of derivation steps is k ($k \geq 2$).
  We have $n = k-1$. There are $k-1$ steps from $\langle p, S_1, m_1\rangle$ to $\langle p, S_{n+1}, m_{n+1}\rangle$.
  By the induction hypothesis, there is some sa such that
  $$(\mathit{fst}(S_n), sa, \mathit{fst}(S_{n+1})) \in \mathit{succ}(S_1, \iota_{\mathrm{f}})$$
  and α, sa, m_n, and m_{n+1} satisfy the final condition of the lemma.
  We have $\langle p, S_0, m_0\rangle \xrightarrow{\alpha_1} \langle p, S_1, m_1\rangle$ from the hypotheses of the lemma. From $\langle p, S_1, m_1\rangle$, it is known that there is a further step. Hence, we have $\mathit{succ}(S_1, \iota_{\mathrm{f}}) \subseteq \mathit{succ}(S_0, \iota_{\mathrm{f}})$ by Lemma 7.
  Therefore, we have $(\mathit{fst}(S_n), sa, \mathit{fst}(S_{n+1})) \in \mathit{succ}(S_0, \iota_{\mathrm{f}})$ with α, sa, m_n and m_{n+1} satisfying the final condition of the lemma.

The induction above completes the proof of this lemma. □

We next prove Lemma 3 from the main text as follows.

Proof (of Lemma 3). We have

$$\mathcal{A}_{\mathrm{G}}^{CS,k}(\mathit{fst}(S'_k)) = (Eq, f) = \bigsqcup\{(E(\vec{\imath}), B(\vec{\imath})) \mid \imath'_k = \mathit{fst}(S'_k) \wedge \mathit{may\text{-}step}(CS, \vec{\imath}, k)\}$$

We perform an induction on the length of tr to show if

$$\mathit{last}(tr) = \langle\langle p_1, S'_1, m'_1\rangle, \ldots, \langle p_n, S'_n, m'_n\rangle\rangle$$
$$\mathit{may\text{-}step}(CS, \mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n), k)$$

then it holds that

– if $(x@p_i = [e]@p_j) \in E(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))$, then $m'_i(x) = [\![e]\!]_{m'_j}$, and
– $\forall \imath \in Pt : \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))(\imath@p_k)$.

The conclusion of this lemma directly follows from the conditions above due to the definition of $\mathcal{A}_{\mathrm{G}}^{CS,k}(\mathit{fst}(S'_k))$.

The induction proof is as follows.

1. Suppose $tr = \mathit{gcnf}_0$ where $\mathit{gcnf}_0 = \langle\langle p_1, S_1, m_\star\rangle, \ldots, \langle p_n, S_n, m_\star\rangle\rangle$. We have $S'_1 = S_1, \ldots, S'_n = S_n$, and $m'_1 = m_\star, \ldots, m'_n = m_\star$.
   From the constraints of the global analysis, we have $E(\mathit{fst}(S_1)\ldots\mathit{fst}(S_n)) \subseteq Eq^\star_{CS}$. Hence, for each equality $(x@p_i = [e]@p_j) \in E(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))$, we have $(x@p_i = [e]@p_j) \in Eq^\star_{CS}$. Hence, we have $m_\star(x) = [\![e]\!]_{m_\star}$ by the definition of $Eq^\star_{CS}$. Therefore, we have $m'_i(x) = [\![e]\!]_{m'_j}$.
   From the constraints of the global analysis, we have for all κ,
   $$? \in B(\mathit{fst}(S_1)\ldots\mathit{fst}(S_n))(\kappa)$$
   We have $\mathit{last\text{-}at}(\mathit{acts\text{-}of}(\mathit{gcnf}_0), p_k, \imath) = ?$. Hence, we have
   $$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(\mathit{gcnf}_0), p_k, \imath) \in B(\mathit{fst}(S_1)\ldots\mathit{fst}(S_n))(\kappa)$$
   Therefore, for all $\imath$, it holds that
   $$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))(\imath@p_k)$$
2. Suppose $tr = tr'.\alpha.\mathit{gcnf}$, where $\mathit{last}(tr') = \langle\langle p_1, S''_1, m''_1\rangle, \ldots, \langle p_n, S''_n, m''_n\rangle\rangle$. By the induction hypothesis, we have
   $$\forall i, j, x, e : (x@p_i = [e]@p_j) \in E(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n)) \;\Rightarrow\; m''_i(x) = [\![e]\!]_{m''_j} \tag{6}$$
   $$\forall \imath \in Pt : \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr'), p_k, \imath) \in B(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n))(\imath@p_k) \tag{7}$$
   We make a case analysis on the last derivation step for tr (cf. Fig. 2).
   - The case where $\alpha = \diamond$ is straightforward.
   - Suppose tr is derived from tr' due to $\langle p_i, S''_i, m''_i\rangle \xrightarrow{c!\vec{v}} \langle p_i, S'_i, m'_i\rangle$ and $\langle p_j, S''_j, m''_j\rangle \xrightarrow{c?\vec{v}} \langle p_j, S'_j, m'_j\rangle$, with $\alpha = \tau$. Using Lemma 8, we have $(\mathit{fst}(S''_i), c!\vec{e'}, \mathit{fst}(S'_i)) \in \mathit{succ}(S_i)$ for some $\vec{e'}$, and $(\mathit{fst}(S''_j), c?\vec{x'}, \mathit{fst}(S'_j)) \in \mathit{succ}(S_j)$ for some $\vec{x'}$. It is not difficult to derive $\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n) \in V_{CS}$, because $tr' \in \mathit{traces\text{-}of}_\xi(CS)$. Hence, we have
     $$(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n), \{(i, c!\vec{e'}), (j, c?\vec{x'})\}, \mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n)) \in E_{CS}$$
     We therefore have the constraints
     $$E(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n)) \subseteq [E(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n))]^{p_j}_{c?\vec{x'}} \cup \bigcup_r \{x'_r@p_j = [e'_r]@p_i\} \tag{8}$$
     $$\forall \kappa : B(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))(\kappa) \supseteq B(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n))(\kappa) \tag{9}$$
     Pick arbitrary equality $(x@p_h = [e]@p_s) \in E(\mathit{fst}(S'_1)\ldots\mathit{fst}(S'_n))$.
     If $(x@p_h = [e]@p_s) \in [E(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n))]^{p_j}_{c?\vec{x'}}$, then we have $(x@p_h = [e]@p_s) \in E(\mathit{fst}(S''_1)\ldots\mathit{fst}(S''_n))$ and no variable in $x@p_h = [e]@p_s$ is updated by $c?\vec{x'}$. Using (6), it can be deduced that $m''_h(x) = [\![e]\!]_{m''_s}$. By the non-updatedness of $x@p_h = [e]@p_s$, we have $m'_h(x) = [\![e]\!]_{m'_s}$.

If $(x@p_h = [e]@p_s) \in \bigcup_r \{x'_r@p_j = [e'_r]@p_i\}$, then we have $m'_h(x) = [\![e]\!]_{m'_s}$ by the semantics.
Pick arbitrary $\imath$. We have

$$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr'), p_k, \imath)$$

because $\alpha = \tau$. Hence, we have

$$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n))(\imath@p_k)$$

by (7). Therefore, we have

$$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S'_1) \ldots \mathit{fst}(S'_n))(\imath@p_k)$$

by (9).

– Suppose tr is derived from tr' due to $\langle p_i, S''_i, m''_i\rangle \xrightarrow{\alpha} \langle p_i, S'_i, m'_i\rangle$ for some i. Using Lemma 8, we obtain $(\mathit{fst}(S''_i), sa, \mathit{fst}(S'_i)) \in \mathit{succ}(S_i)$ for some syntactical action sa, such that $\alpha = c!\vec{v}$ and $sa = c!\vec{e'}$ for some $\vec{e'}$, $\alpha = c?\vec{v}$ and $sa = c?\vec{x'}$ for some $\vec{x'}$, $\alpha = \mathsf{brt}^{p_i, \mathit{fst}(S''_i)}$ and $sa = \mathsf{brt}$, $\alpha = \mathsf{brf}^{p_i, \mathit{fst}(S''_i)}$ and $sa = \mathsf{brf}$, or $\alpha = \tau$ and $sa = \tau \vee \exists x' : \exists e' : sa = x' \leftarrow e'$. It is not difficult to deduce that $\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n) \in V_{CS}$. By $\langle p_i, S''_i, m''_i\rangle \xrightarrow{\alpha} \langle p_i, S'_i, m'_i\rangle$ and the correspondence between α and sa, we have that if sa is a (syntactical) communication over a polyadic channel c, then $c \in \mathit{EPCh}$. Hence, we have

$$((\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n)), (i, sa), (\mathit{fst}(S'_1) \ldots \mathit{fst}(S'_n))) \in E_{CS}$$

We make a case analysis on sa.

- Case $sa = \mathsf{brt}$. We have $\alpha = \mathsf{brt}^{p_i, \mathit{fst}(S''_i)}$.
  In the global analysis, we have the constraints

$$E(\mathit{fst}(S'_1) \ldots \mathit{fst}(S'_n)) \subseteq E(\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n)) \tag{10}$$

$$\forall \kappa : B(\mathit{fst}(S'_1) \ldots \mathit{fst}(S'_n))(\kappa) \supseteq B(\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n))[\mathit{fst}(S''_i)@p_i \mapsto \{tt\}](\kappa) \tag{11}$$

  It is straightforward, using (10) and (6), to show that for each equality $x@p_h = [e]@p_s$, it holds that $m'_h(x) = [\![e]\!]_{m'_s}$.
  Pick an arbitrary $\imath$. We make a case analysis on whether $\imath@p_k = \mathit{fst}(S''_i)@p_i$.
  * Case $\imath@p_k \neq \mathit{fst}(S''_i)@p_i$.
    We have $\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr'), p_k, \imath)$. Hence, we have

$$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S''_1) \ldots \mathit{fst}(S''_n))(\imath@p_k)$$

    by (7). Hence, we obtain

$$\mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p_k, \imath) \in B(\mathit{fst}(S'_1) \ldots \mathit{fst}(S'_n))(\imath@p_k)$$

    using (11).

* Case $\imath@p_k = fst(S''_i)@p_i$.
  We have $B(fst(S''_1)\ldots fst(S''_n))[fst(S''_i)@p_i \mapsto \{tt\}](\imath@p_k) = \{tt\}$.
  Hence, $tt \in B(fst(S'_1)\ldots fst(S'_n))(\imath@p_k)$ by (11). We have

$$last\text{-}at(acts\text{-}of(tr), p_k, \imath) = tt$$

  Hence, we have

$$last\text{-}at(acts\text{-}of(tr), p_k, \imath) \in B(fst(S'_1)\ldots fst(S'_n))(\imath@p_k)$$

- Case $sa = \mathsf{brf}$.
  The reasoning is analogous to that of the previous case.
- Case $sa \notin \{\mathsf{brt}, \mathsf{brf}\}$.
  In the global analysis, we have

$$E(fst(S'_1)\ldots fst(S'_n)) \subseteq [E(fst(S''_1)\ldots fst(S''_n))]^{p_i}_{sa} \quad (12)$$
$$\forall \kappa : B(fst(S'_1)\ldots fst(S'_n))(\kappa) \supseteq B(fst(S''_1)\ldots fst(S''_n))(\kappa) \quad (13)$$

  It is not difficult to establish the two goals of the induction proof with reasoning analogous to that of the case of internal communication.

This completes the induction proof, and in turn, the proof of this lemma. □

B.2 Proof of Lemma 4

Lemma 9. *If $\Psi \vdash D_1, S, D_2$ and $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$ can be derived, $tr \in Tr$, and there exist μ and φ such that $(\mu, \varphi) \in D_1$, $\forall \imath : \mu(\imath) = last\text{-}at(acts\text{-}of(tr), p, \imath)$, and φ holds at m, then*

- *if $S' \neq {}^{\iota_f}\mathsf{stop}$, then there exists some D'_1 such that $\Psi \vdash D'_1, S', D_2$ can be derived, and there exist μ' and φ' such that $(\mu', \varphi') \in D'_1$, $\forall \imath : \mu'(\imath) = last\text{-}at(acts\text{-}of(tr).\alpha, p, \imath)$, and φ' holds at m', and*
- *if $S' = {}^{\iota_f}\mathsf{stop}$, then there exist some μ' and φ' such that $(\mu', \varphi') \in D_2$, $\forall \imath : \mu'(\imath) = last\text{-}at(acts\text{-}of(tr).\alpha, p, \imath)$, and φ' holds at m'.*

Proof. The proof is by induction on the derivation of $\Psi \vdash D_1, S, D_2$. Only selected cases are presented.

- Case ${}^{\imath}\mathsf{recv}(c, \vec{x})$. We have $S' = {}^{\iota_f}\mathsf{stop}$. Assume for μ_0 and $\forall \vec{v}.\varphi_0[\vec{v}/\vec{x}]$ with $(\mu_0, \varphi_0) \in D_2$ we have $\forall \imath : \mu_0(\imath) = last\text{-}at(acts\text{-}of(tr), p, \imath)$ and $\forall \vec{v}.\varphi_0[\vec{v}/\vec{x}]$ holds at m. Hence, we have $\forall \imath : \mu_0(\imath) = last\text{-}at(acts\text{-}of(tr).\alpha, p, \imath)$ because $\alpha = c?\vec{v'}$ for some $\vec{v'}$. We have $\varphi_0[\vec{v}/\vec{x}]$ holds at $m[(v_j \mapsto a_j)_j]$ for all $a_1, \ldots, a_{|\vec{x}|}$ because $\forall \vec{v}.\varphi_0[\vec{v}/\vec{x}]$ holds at m. Hence, φ_0 holds at $m[(v_j \mapsto a_j)_j][(x_j \mapsto m[(v_j \mapsto a_j)_j](v_j))_j]$. Hence, φ_0 holds at $m[(x_j \mapsto a_j)_j]$ for all $a_1, \ldots, a_{|\vec{x}|}$. Hence, φ_0 holds at m'.
- Case $S_1; S_2$. We have $\langle p, S_1, m\rangle \xrightarrow{\alpha} \langle p, S'_1, m'\rangle$ from $\langle p, S_1; S_2, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$. We have $\Psi \vdash D_1, S_1, D'$ and $\Psi \vdash D', S_2, D_2$ from $\Psi \vdash D_1, S_1; S_2, D_2$. We make a case analysis on S'_1.

- Suppose $S_1' \neq {}^{\iota_f}\mathsf{stop}$. By the induction hypothesis, there exist D_1' such that $\Psi \vdash D_1', S_1', D'$ and there exist μ' and φ' such that $(\mu', \varphi') \in D_1'$, $\forall \imath : \mu'(\imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr).\alpha, p, \imath)$, and φ' holds at m'. Hence, we have $\Psi \vdash D_1', S_1'; S_2, D_2$ with D_1', m' and α satisfying the required condition in the conclusion of the lemma.
- Suppose $S_1' = {}^{\iota_f}\mathsf{stop}$. By the induction hypothesis, there exist some μ' and φ' such that $(\mu', \varphi') \in D'$, $\forall \imath : \mu'(\imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr).\alpha, p, \imath)$, and φ' holds at m'. This directly gives the desired conclusion because $S' = S_2$.

– Case ${}^{\imath_0}\mathsf{if}\ e\ \mathsf{then}\ S_\mathrm{a}\ \mathsf{else}\ S_\mathrm{b}\ \mathsf{fi}$. We make a case analysis on $[\![e]\!]_m$.
- Case $[\![e]\!]_m = tt$. We have $S' = S_\mathrm{a}$, $m' = m$, $\alpha = \mathsf{brt}^{p,\imath_0}$, and $D_1' = \{(\mu[\imath_0 \mapsto tt], \varphi \wedge e) \mid (\mu, \varphi) \in D_1\}$.
  Assume for some μ_0 and φ_0, it holds that $(\mu_0, \varphi_0) \in D_1$, $\forall \imath : \mu_0(\imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p, \imath)$, and φ_0 holds in m.
  Pick $\mu' = \mu_0[\imath_0 \mapsto tt]$, and $\varphi' = \varphi_0 \wedge e$. We have $(\mu', \varphi') \in D_2$. Pick an arbitrary $\imath$. If $\imath = \imath_0$, then we have $\mu'(\imath_0) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr).\alpha, p, \imath_0) = tt$. If $\imath \neq \imath_0$, then we have $\mu'(\imath) = \mu_0(\imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr), p, \imath) = \mathit{last\text{-}at}(\mathit{acts\text{-}of}(tr).\alpha, p, \imath)$. We have φ' holds in m' because φ_0 holds in m, $[\![e]\!]_m = tt$, and $m' = m$.
- Case $[\![e]\!]_m = ff$. The reasoning is analogous to that of the case $[\![e]\!]_m = tt$.

– Case subsumption. The reasoning is by a cases analysis on whether S' is ${}^{\iota_f}\mathsf{stop}$, and using the induction hypothesis in each of the two cases.

This induction completes the proof of this lemma. □

Lemma 10. *If $\Psi \vdash D_1, S, D_2$, then $D_1 \rightsquigarrow \Psi(\mathit{fst}(S))$.*

Proof. The proof is by a straightforward induction on the derivation of $\Psi \vdash D_1, S, D_2$, using $\mathit{fst}(S_1; S_2) = \mathit{fst}(S_1)$, and the transitivity of $\rightsquigarrow$. □

Lemma 11. *If $\Psi \vdash \{(\lambda\imath.?, \varphi_0)\}, S, D$ for some φ_0 that holds in $m_\star$, and for each $i \in \{1, \ldots, n\}$, it can be derived that $\langle p, S_{i-1}, m_{i-1}\rangle \xrightarrow{\alpha_{i-1}} \langle p, S_i, m_i\rangle$, where $S_0 = S$ and $m_0 = m_\star$, then*

– *if $S_n \neq {}^{\iota_f}\mathsf{stop}$, then there exists some D' such that $\Psi \vdash D', S_n, D$ can be derived, and there exist some μ and φ, such that $(\mu, \varphi) \in D'$, $\forall \imath : \mu(\imath) = \mathit{last\text{-}at}(\alpha_0 \ldots \alpha_{n-1}, p, \imath)$, and φ holds in m_n, and*
– *if $S_n = {}^{\iota_f}\mathsf{stop}$, then there exist some μ and φ, such that $(\mu, \varphi) \in D$, $\forall \imath : \mu(\imath) = \mathit{last\text{-}at}(\alpha_0 \ldots \alpha_{n-1}, p, \imath)$, and φ holds in m_n.*

Proof. We perform an induction on n.

– Case $n = 0$. We have $S \neq {}^{\iota_f}\mathsf{stop}$ because $\Psi \vdash \{(\lambda\imath.?, \varphi_0)\}, S, D$ can be derived. We have $D' = \{(\lambda\imath.?, \varphi_0)\}$. Pick an arbitrary $\imath$, we have $(\lambda\imath.?)(\imath) = \mathit{last\text{-}at}(\epsilon, p, \imath) =?$. We have φ_0 holds in $m_\star$, which is m_0.
– Case $n = k$ ($k > 0$). We have $S_{k-1} \neq {}^{\iota_f}\mathsf{stop}$. Hence, using the induction hypothesis, we obtain that there exists some D' such that $\Psi \vdash D', S_{k-1}, D$ can be derived, and there exist some μ and φ such that $(\mu, \varphi) \in D'$, $\forall \imath :$

$\mu(\imath) = \mathit{last\text{-}at}(\alpha_0 \dots \alpha_{k-2}, p, \imath)$, and φ holds in m_{k-1}. We have the local step $\langle p, S_{k-1}, m_{k-1}\rangle \xrightarrow{\alpha_{k-1}} \langle p, S_k, m_k\rangle$. The conclusion of this lemma can now be derived based on this step, using Lemma 9.

The induction above completes the proof of this lemma. □

The proof of Lemma 4 is as follows.

Proof (of Lemma 4). By the definition of $\mathcal{A}^S_{\mathrm{L}}$, we have $\Psi \vdash \{(\lambda\imath.?, \varphi_0)\}, S, D$ for some φ_0 that holds in $m_\star$, such that for all $\imath \neq \iota_{\mathrm{f}}$, it holds that $\mathcal{A}^S_{\mathrm{L}}(\imath) = \Psi(\imath)$, and $\mathcal{A}^{\iota_{\mathrm{f}}}_{\mathrm{L}} = D$.

We make a case analysis on S_n.

- Case $S_n \neq {}^{\iota_{\mathrm{f}}}\mathsf{stop}$. We have some D' such that $\Psi \vdash D', S_n, D$ and some μ and φ, such that $(\mu, \varphi) \in D'$, $\forall \imath : \mu(\imath) = \mathit{last\text{-}at}(\alpha_0 \dots \alpha_{n-1}, p, \imath)$, and φ holds in m_n, by Lemma 11. We have $D' \rightsquigarrow \Psi(\mathit{fst}(S_n))$ by Lemma 10. Hence, $D' \rightsquigarrow \mathcal{A}^S_{\mathrm{L}}(\mathit{fst}(S_n))$. The conclusion of the lemma can now be derived using the definition of $\rightsquigarrow$.
- Case $S_n = {}^{\iota_{\mathrm{f}}}\mathsf{stop}$. The conclusion of the lemma can be directly deduced using the definition of $\mathcal{A}^S_{\mathrm{L}}$ and Lemma 11.

This completes the proof. □

We next give the proof of Theorem 2 based on Lemma 3 and Lemma 4.

B.3 Proof of Theorem 2 Using Lemma 3 and Lemma 4

Proof. Assume $tr \in \mathit{traces\text{-}of}_\xi(CS)$,

$$\mathit{last}(tr) = \mathit{gcnf} = \langle\langle p_1, S'_1, m_1\rangle, \dots, \langle p_n, S'_n, m'_n\rangle\rangle$$

and $\mathit{may\text{-}step}(CS, \mathit{fst}(S'_1) \dots \mathit{fst}(S'_n), k)$ holds.

Assume that $\mathcal{A}^{CS,k}_{\mathrm{G}}(\mathit{fst}(S'_k)) = (Eq, f)$ for some Eq and f, and $\mathcal{A}^{S_k}_{\mathrm{L}}(\mathit{fst}(S'_k)) = D$ for some D.

By Lemma 3, we have for each equality in Eq, say denoted by ϕ_{EQ}, that $[\![\phi_{\mathrm{EQ}}]\!]_{\mathit{gcnf}} = tt$. Hence, we have $[\![\bigwedge Eq]\!]_{\mathit{gcnf}} = tt$.

Suppose the derivation of tr is attributed to the following sequence of actions performed by the k-th process (excl. the actions resulting in an overall $\diamond$ at the system level): $\alpha^1_k. \dots . \alpha^h_k$. There exist some μ and φ such that $(\mu, \varphi) \in \mathcal{A}^{S_k}_{\mathrm{L}}(\mathit{fst}(S'_k))$, $\forall \imath : \mu(\imath) = \mathit{last\text{-}at}(\alpha^1_k. \dots . \alpha^h_k, p_k, \imath)$, and φ holds in m'_k, according to Lemma 4. We have $\forall \imath : \mathit{last\text{-}at}(\alpha^1_k. \dots . \alpha^h_k, p_k, \imath) = \mathit{last\text{-}at}(tr, p_k, \imath)$. We have $\forall \imath : \mathit{last\text{-}at}(tr, p_k, \imath) \in f(\imath@p_k)$ by Lemma 3. Hence, we have $\forall \imath : \mu(\imath) \in f(\imath@p_k)$. We have $[\![[\varphi]@p_k]\!]_{\mathit{gcnf}} = tt$ because φ holds in m'_k. Hence, we have $[\![\bigvee\{\, [\varphi]@p_k \mid \exists \mu : (\mu, \varphi) \in D \wedge \forall \imath' : \mu(\imath') \in f(\imath'@p_k)\}]\!]_{\mathit{gcnf}} = tt$.

Therefore, it holds that $[\![\Phi(\mathit{fst}(S'_k))]\!]_{\mathit{gcnf}} = tt$. This completes the proof of the theorem. □

C Proof of Theorem 1

Our proof of Theorem 1 is outlined as follows. An augmented programming language is first defined where a statement can be annotated with the conditional expressions whose scopes cover the statement. A security type system is then defined for this augmented language (with most parts analogous to our security type system in Section 5). This makes it possible to keep track of the conditional expressions used in specializing the security types at different points in a statement. Using this security type system, a notion of syntactical highness for a process, a notion of low-equivalence for two processes, and a notion of low-equivalence for two systems are then defined. In these definitions, "high" and "low" are conceptualized wrt. a group $\mathcal{G}$ of principals, and wrt. the specialized security types for variables. On this basis, a two-run property for systems composed of augmented statements is proven (Lemma 29). Finally, this two-run property is shown to give rise to the soundness of our security type system (for the non-augmented language). The augmented language and its security type system are presented in Appendix C.1, and the remaining part of the proof of Theorem 1 is presented in Appendix C.2.

C.1 Augmented Language, and Security Type System

Syntax of the Augmented Language. To keep track of the specialization of globally dependent flow policies with conditional expressions under scoped specialization, we augment the syntactical category of statements in our programming language with statements annotated with $\imath\{e\}$. Here, e is a conditional expression, and the annotation is called a *conditional block*. We thus obtain the syntax for *augmented statements* AS as shown in Fig. 14.

$$
\begin{aligned}
AS ::= {}& \imath\mathsf{skip} \mid x := \imath e \mid \imath\mathsf{send}(c, \vec{e}) \mid \imath\mathsf{recv}(c, \vec{x}) \mid AS; AS \mid \\
& \imath\mathsf{if}\ e\ \mathsf{then}\ AS\ \mathsf{else}\ AS\ \mathsf{fi} \mid \imath\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od} \mid \imath\mathsf{stop} \mid C\ AS \\
C ::= {}& \imath\{e\}
\end{aligned}
$$

Fig. 14. The Syntax of Augmented Statements

We next define a series of utility functions and predicates on augmented statements. We define the predicate $afst : AStmt \to Pt$ by $afst(\imath\mathsf{skip}) = \imath$, $afst(\imath\mathsf{stop}) \triangleq \imath$, $afst(x := \imath e) \triangleq \imath$, $afst(\imath\mathsf{send}(c, \vec{e})) \triangleq \imath$, $afst(\imath\mathsf{recv}(c, \vec{x})) \triangleq \imath$, $afst(\imath\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}) \triangleq \imath$, $afst(\imath\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}) \triangleq \imath$, $afst(\imath\{e\}AS) \triangleq afst(AS)$, and $afst(AS_1; AS_2) \triangleq afst(AS_1)$.

We define $\mathit{fvs\text{-}cond} : \mathit{AStmt} \to \mathcal{P}(\mathit{Var})$ to retrieve the set of free variables of all conditional blocks and conditional expressions in an augmented statement.

$$\begin{aligned}
\mathit{fvs\text{-}cond}(AS_1; AS_2) &\triangleq \mathit{fvs\text{-}cond}(AS_1) \cup \mathit{fvs\text{-}cond}(AS_2) \\
\mathit{fvs\text{-}cond}(\text{`}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}) &\triangleq \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_1) \cup \mathit{fvs\text{-}cond}(AS_2) \\
\mathit{fvs\text{-}cond}(\text{`}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}) &\triangleq \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS) \\
\mathit{fvs\text{-}cond}(\text{`}\{e\}AS) &\triangleq \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS) \\
\mathit{fvs\text{-}cond}(AS) &\triangleq \emptyset \quad \text{if } AS \text{ is not one of the above}
\end{aligned}$$

We define the predicate $\mathit{atoms} : \mathit{AStmt} \to \mathcal{P}(\mathit{Stmt})$ by

$$\begin{aligned}
\mathit{atoms}(\text{`}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}) &\triangleq \mathit{atoms}(AS_1) \cup \mathit{atoms}(AS_2) \\
\mathit{atoms}(\text{`}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}) &\triangleq \mathit{atoms}(AS) \\
\mathit{atoms}(AS_1; AS_2) &\triangleq \mathit{atoms}(AS_1) \cup \mathit{atoms}(AS_2) \\
\mathit{atoms}(\text{`}\{e\}AS) &\triangleq \mathit{atoms}(AS) \\
\mathit{atoms}(AS) &\triangleq \{AS\} \quad \text{if } AS \text{ is not one of the above}
\end{aligned}$$

We define the function $\mathit{upd\text{-}next} : \mathit{AStmt} \to \mathcal{P}(\mathit{Var})$ by

$$\begin{aligned}
\mathit{upd\text{-}next}(x :=\text{`}a) &\triangleq \{x\} \\
\mathit{upd\text{-}next}(\text{`}\mathsf{recv}(c, \vec{x})) &\triangleq \{\vec{x}\} \\
\mathit{upd\text{-}next}(AS_1; AS_2) &\triangleq \mathit{upd\text{-}next}(AS_1) \\
\mathit{upd\text{-}next}(\text{`}\{e\}AS_1) &\triangleq \mathit{upd\text{-}next}(AS_1) \\
\mathit{upd\text{-}next}(AS) &\triangleq \emptyset \quad \text{if } AS \text{ is not one of the above}
\end{aligned}$$

We inductively define $\mathit{strip\text{-}stmt}(AS)$ to remove the conditional blocks in augmented statements.

$$\begin{aligned}
\mathit{strip\text{-}stmt}(\text{`}\mathsf{skip}) &\triangleq \text{`}\mathsf{skip} \\
\mathit{strip\text{-}stmt}(x :=\text{`}e) &\triangleq x :=\text{`}e \\
\mathit{strip\text{-}stmt}(\text{`}\mathsf{send}(c, \vec{e})) &\triangleq \text{`}\mathsf{send}(c, \vec{e}) \\
\mathit{strip\text{-}stmt}(\text{`}\mathsf{recv}(c, \vec{x})) &\triangleq \text{`}\mathsf{recv}(c, \vec{x}) \\
\mathit{strip\text{-}stmt}(AS_1; AS_2) &\triangleq \mathit{strip\text{-}stmt}(AS_1); \mathit{strip\text{-}stmt}(AS_2) \\
\mathit{strip\text{-}stmt}(\text{`}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}) &\triangleq \text{`}\mathsf{if}\ e\ \mathsf{then}\ \mathit{strip\text{-}stmt}(AS_1) \\
&\qquad\qquad \mathsf{else}\ \mathit{strip\text{-}stmt}(AS_2) \\
\mathit{strip\text{-}stmt}(\text{`}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}) &\triangleq \text{`}\mathsf{while}\ e\ \mathsf{do}\ \mathit{strip\text{-}stmt}(AS)\ \mathsf{od} \\
\mathit{strip\text{-}stmt}(\text{`}\{e\}AS) &\triangleq \mathit{strip\text{-}stmt}(AS)
\end{aligned}$$

Replacing each statement by a corresponding augmented statement in a concurrent statement, we obtain an *augmented concurrent statement* $ACS = p_1 : AS_1 || \ldots || p_n : AS_n$.

$$\langle p, {}^{\iota}\mathsf{skip}, m\rangle \xrightarrow{\tau} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle$$

$$\langle p, x :=^{\iota} e, m\rangle \xrightarrow{\tau} \langle p, {}^{\iota_f}\mathsf{stop}, m[x \mapsto [\![e]\!]_m]\rangle$$

$$\frac{\langle p, AS_1, m\rangle \xrightarrow{\alpha} \langle p, AS_1', m'\rangle}{\langle p, AS_1; AS_2, m\rangle \xrightarrow{\alpha} \langle p, AS_1'; AS_2, m'\rangle} \quad \text{if } \neg(\exists \vec{C} : AS_1' = \mathit{blks}(\vec{C}, {}^{\iota_f}\mathsf{stop}))$$

$$\frac{\exists \vec{C} : \langle p, AS_1, m\rangle \xrightarrow{\alpha} \langle p, \mathit{blks}(\vec{C}, {}^{\iota_f}\mathsf{stop}), m'\rangle}{\langle p, AS_1; AS_2, m\rangle \xrightarrow{\alpha} \langle p, AS_2, m'\rangle}$$

$$\langle p, {}^{\iota}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}, m\rangle \xrightarrow{\mathsf{brt}^{p,\iota}} \langle p, {}^{\iota}\{e\}AS_1, m\rangle \quad \text{if } [\![e]\!]_m = \mathit{tt}$$

$$\langle p, {}^{\iota}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}, m\rangle \xrightarrow{\mathsf{brf}^{p,\iota}} \langle p, {}^{\iota}\{\neg e\}AS_2, m\rangle \quad \text{if } [\![e]\!]_m = \mathit{ff}$$

$$\langle p, {}^{\iota}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brt}^{p,\iota}} \langle p, ({}^{\iota}\{e\}AS); {}^{\iota}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}, m\rangle \quad \text{if } [\![e]\!]_m = \mathit{tt}$$

$$\langle p, {}^{\iota}\mathsf{while}\ e\ \mathsf{do}\ AS\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brf}^{p,\iota}} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle \quad \text{if } [\![e]\!]_m = \mathit{ff}$$

$$\langle p, {}^{\iota}\mathsf{send}(c, \vec{e}), m\rangle \xrightarrow{c!\vec{v}} \langle p, {}^{\iota_f}\mathsf{stop}, m\rangle \quad \text{if } [\![\vec{e}]\!]_m = \vec{v}$$

$$\langle p, {}^{\iota}\mathsf{recv}(c, \vec{x}), m\rangle \xrightarrow{c?\vec{v}} \langle p, {}^{\iota_f}\mathsf{stop}, m[(x_j \mapsto v_j)_j]\rangle$$

$$\frac{\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle}{\langle p, {}^{\iota}\{e\}AS, m\rangle \xrightarrow{\alpha} \langle p, {}^{\iota}\{e\}AS', m'\rangle}$$

Fig. 15. The Semantics of Augmented Statements

Semantics of the Augmented Language. We call configurations of the form $\langle p, AS, m\rangle$ *augmented local configurations*. They are like local configurations except that they contain augmented statements instead of statements. Analogously, if a configuration is like a global configuration except that it contains augmented statements instead of statements, it is called an *augmented global configuration*. Furthermore, if a sequence is like a trace except that it contains exclusively augmented global configurations instead of global configurations, the sequence is called an *augmented trace*. We denote the set of augmented traces by ATr.

For an augmented local configuration $\mathit{alcnf} = \langle p, AS, m\rangle$, we write $\mathit{astmt\text{-}of}(\mathit{alcnf})$ for the augmented statement AS, and $\mathit{prin\text{-}of}(\mathit{alcnf})$ for the principal p. For an augmented global configuration $\mathit{agcnf} = \langle\langle p_1, AS_1, m_1\rangle, \ldots, \langle p_n, AS_n, m_n\rangle\rangle$, we write $\mathit{acstmt\text{-}of}(\mathit{agcnf})$ for the augmented concurrent statement $p_1 : AS_1 || \ldots || p_n : AS_n$.

To remove the conditional blocks in augmented global configurations, we define $\mathit{strip\text{-}gcnf}(\mathit{agcnf})$ by

$$\mathit{strip\text{-}gcnf}(\langle\langle p_1, AS_1, m_1\rangle, \ldots, \langle p_n, AS_n, m_n\rangle\rangle) \triangleq \langle\langle p_1, \mathit{strip\text{-}stmt}(AS_1), m_1\rangle, \ldots, \langle p_n, \mathit{strip\text{-}stmt}(AS_n), m_n\rangle\rangle$$

To remove the conditional blocks in augmented traces, we inductively define $\mathit{strip\text{-}tr}(\mathit{atr})$ by

$$\mathit{strip\text{-}tr}(\mathit{agcnf}) \triangleq \mathit{strip\text{-}gcnf}(\mathit{agcnf})$$

$$\mathit{strip\text{-}tr}(\mathit{atr}.\alpha.\mathit{agcnf}) \triangleq \mathit{strip\text{-}tr}(\mathit{atr}).\alpha.\mathit{strip\text{-}gcnf}(\mathit{agcnf})$$

$$\frac{last(atr) = \langle \ldots, alcnf_1, \ldots, alcnf_2, \ldots \rangle \quad alcnf_1 \xrightarrow{c!\vec{v}} alcnf_1' \quad alcnf_2 \xrightarrow{c?\vec{v}} alcnf_2'}{atr \rightarrow_\xi atr.\tau.\langle \ldots, alcnf_1', \ldots, alcnf_2', \ldots \rangle} \text{ if } \xi(strip\text{-}tr(atr)) = (prin\text{-}of(alcnf_1), prin\text{-}of(alcnf_2))$$

$$\frac{last(atr) = \langle \ldots, alcnf_1, \ldots, alcnf_2, \ldots \rangle \quad alcnf_1 \xrightarrow{c?\vec{v}} alcnf_1' \quad alcnf_2 \xrightarrow{c!\vec{v}} alcnf_2'}{atr \rightarrow_\xi atr.\tau.\langle \ldots, alcnf_1', \ldots, alcnf_2', \ldots \rangle} \text{ if } \xi(strip\text{-}tr(atr)) = (prin\text{-}of(alcnf_1), prin\text{-}of(alcnf_2))$$

$$\frac{last(atr) = \langle \ldots, alcnf, \ldots \rangle \quad alcnf \xrightarrow{\alpha} alcnf'}{atr \rightarrow_\xi atr.\alpha.\langle \ldots, alcnf', \ldots \rangle} \text{ if } \exists p, ci : \xi(strip\text{-}tr(atr)) = (p, ci) \wedge p = prin\text{-}of(alcnf) \wedge match(\alpha, ci)$$

$$\frac{\neg(\exists agcnf, \alpha : \alpha \neq \diamond \wedge atr \rightarrow_\xi atr.\alpha.agcnf)}{atr \rightarrow_\xi atr.\diamond.last(atr)}$$

Fig. 16. The Semantics for Augmented Concurrent Statements

For the augmented statements, we formulate a semantics with the judgment $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$, and with the calculus rules in Fig. 15. In this figure, $blks(\vec{C}, AS)$ is written for the augmented statement

$$C_1(C_2(\ldots(C_n\ AS)\ldots))$$

We introduce the judgment $atr \rightarrow_\xi atr'$ for an execution step of an augmented concurrent statement (with information about conditional blocks recorded). The calculus rules for this judgment are given in Fig. 16.

Given an augmented concurrent statement $ACS = p_1 : AS_1 || \ldots || p_n : AS_n$, we write $atraces\text{-}of_\xi(ACS)$ for the set of augmented traces of ACS under the strategy ξ. That is,

$$atraces\text{-}of_\xi(ACS) \triangleq \{atr \in ATr \mid \langle\langle p_1, AS_1, m_\star\rangle, \ldots, \langle p_n, AS_n, m_\star\rangle\rangle(\rightarrow_\xi)^* atr\}$$

We have the following two lemmas on the transparency of the augmentation of statements.

Lemma 12. *If $S = strip\text{-}stmt(AS)$, then the following two statements hold*

- *If $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$, then there exists some AS' such that $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$, and $S' = strip\text{-}stmt(AS')$.*
- *If $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$, then $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, strip\text{-}stmt(AS'), m'\rangle$.*

The proof of this lemma is straightforward via induction on the derivation of $\langle p, S, m\rangle \xrightarrow{\alpha} \langle p, S', m'\rangle$ (for the first direction) and of $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$ (for the second direction).

Lemma 13. *If $tr = strip\text{-}tr(atr)$, then the following two statements hold*

- *If $tr \rightarrow_\xi tr'$, then there exists some atr' such that $atr \rightarrow_\xi atr'$ and $tr' = strip\text{-}tr(atr')$.*
- *If $atr \rightarrow_\xi atr'$, then $tr \rightarrow_\xi strip\text{-}tr(atr')$.*

Proof. Assume arbitrary tr and atr such that $tr = \mathit{strip}\text{-}tr(atr)$ holds.

If $tr \to_\xi tr'$, then according to the semantic rules, it must be the case that $tr' = tr.\alpha.gcnf$ with some α and some $gcnf$. The case where $\alpha \neq \diamond$ can be resolved by case analysis on the derivation of $tr \to_\xi tr'$, using Lemma 12. Due to the involvement of negation in the semantic rule for $\alpha = \diamond$, we defer the proof of this case.

If $atr \to_\xi atr'$, then according to the semantic rules, it must be the case that $atr' = atr.\alpha.agcnf$ with some α and $agcnf$. The case where $\alpha \neq \diamond$ can be resolved by case analysis on the derivation of $atr \to_\xi atr'$, using Lemma 12. We defer the case where $\alpha = \diamond$ to the end.

We turn to the case $\alpha = \diamond$ in the first direction. We show that there does not exist any $agcnf$, $\alpha \neq \diamond$ such that $atr \to_\xi atr.\alpha.agcnf$. Assume per absurdum that such $agcnf$ and $\alpha \neq \diamond$ exist. Then from what has been shown, there exist $gcnf$ and $\alpha \neq \diamond$ such that $tr \to_\xi tr.\alpha.gcnf$. But this means that $tr \to_\xi tr'$ is impossible and we have a contradiction with the hypothesis.

The case $\alpha = \diamond$ can be resolved analogously with a contradiction. □

We define the set of well-formed augmented statements $AStmt_{\mathrm{WF}}$ as the least set satisfying the following rules. The intuition is that all augmented statements reachable in the execution of a (plain) statement must be well-formed.

$$\frac{AS \in AStmt_{\mathrm{WF}} \quad S \in Stmt \quad \forall \imath : {}^{\imath}\mathsf{stop} \notin atoms(AS) \cup atoms(S)}{AS; S \in AStmt_{\mathrm{WF}}}$$

$$\frac{AS \in AStmt_{\mathrm{WF}} \quad \imath \neq \iota_{\mathrm{f}}}{{}^{\imath}\{e\}AS \in AStmt_{\mathrm{WF}}}$$

$$\frac{S \in Stmt \quad (\exists \imath : {}^{\imath}\mathsf{stop} \in atoms(S)) \Rightarrow S = {}^{\iota_{\mathrm{f}}}\mathsf{stop}}{S \in AStmt_{\mathrm{WF}}}$$

It can be deduced that $AStmt_{\mathrm{WF}} \subseteq AStmt$.

Lemma 14. *If $AS \in AStmt_{\mathrm{WF}}$, and ${}^{\imath}\mathsf{stop} \in atoms(AS)$ for some $\imath$, then it holds that $\exists \vec{C} : AS = blks(\vec{C}, {}^{\iota_{\mathrm{f}}}\mathsf{stop})$.*

Proof. The proof of this lemma is by induction on the derivation of $AS \in AStmt_{\mathrm{WF}}$.

- Suppose $AS \in AStmt_{\mathrm{WF}}$ is derived with the first rule applied last. We have $AS = AS'; S$ for some AS' and S. Assume ${}^{\imath}\mathsf{stop} \in atoms(AS)$. Then, ${}^{\imath}\mathsf{stop} \in atoms(AS') \cup atoms(S)$. We have a contradiction with the premise.
- Suppose $AS \in AStmt_{\mathrm{WF}}$ is derived with the second rule applied last. We have $AS = {}^{\imath}\{e\}AS'$ for some $\imath$, e, and AS'. Assume ${}^{\imath}\mathsf{stop} \in atoms(AS)$. Then ${}^{\imath}\mathsf{stop} \in atoms(AS')$. By the induction hypothesis, there exists some $\vec{C}$ such that $AS' = blks(\vec{C}, {}^{\iota_{\mathrm{f}}}\mathsf{stop})$. Hence, we have $\vec{C}' = {}^{\imath}\{e\}\vec{C}$ such that $AS = blks(\vec{C}', {}^{\iota_{\mathrm{f}}}\mathsf{stop})$.
- Suppose $AS \in AStmt_{\mathrm{WF}}$ is derived with the third rule applied last. This case can be resolved by using the premise of the rule directly.

This completes the proof. □

Lemma 15. *If $AS \in AStmt_{\mathrm{WF}}$, and $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$ can be derived, then $AS' \in AStmt_{\mathrm{WF}}$.*

Proof. The proof is by induction on the derivation of $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$. Only selected cases of this induction are presented.

- Case $\langle p, AS_1; AS_2, m\rangle \xrightarrow{\alpha} \langle p, AS_1'; AS_2, m'\rangle$ due to $\langle p, AS_1, m\rangle \xrightarrow{\alpha} \langle p, AS_1', m'\rangle$ and $\neg(\exists \vec{C} : AS_1' = blks(\vec{C}, {}^{\iota_{\mathrm{f}}}\mathsf{stop}))$. We make a case analysis on how $AS_1; AS_2 \in AStmt_{\mathrm{WF}}$ is derived.
  - Suppose the derivation of $AS_1; AS_2 \in AStmt_{\mathrm{WF}}$ is by the first rule. We have $AS_1 \in AStmt_{\mathrm{WF}}$, $AS_2 \in Stmt$, and $\forall \imath : {}^{\imath}\mathsf{stop} \notin atoms(AS_1) \cup atoms(AS_2)$. By the induction hypothesis, we have $AS_1' \in AStmt_{\mathrm{WF}}$. By Lemma 14, for all $\imath'$, if ${}^{\imath'}\mathsf{stop} \in atoms(AS_1')$ then $AS_1' = blks(\vec{C}, {}^{\iota_{\mathrm{f}}}\mathsf{stop})$ for some $\vec{C}$. By the side condition of the semantic rule, this is not the case. Hence, ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1')$ for any $\imath'$. It can be derived that ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1') \cup atoms(AS_2)$ for any $\imath'$. Hence, $AS_1'; AS_2 \in AStmt_{\mathrm{WF}}$.
  - Suppose the derivation of $AS_1; AS_2 \in AStmt_{\mathrm{WF}}$ is by the third rule. Then, $AS_1 \in Stmt$, and $AS_2 \in Stmt$. Using the premise of the rule, it can be deduced that ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1; AS_2)$ for any $\imath'$. Hence, ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1)$. We have $AS_1 \in AStmt_{\mathrm{WF}}$. We also have ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1) \cup atoms(AS_2)$. Hence, the reasoning from here can be performed analogously to that of the previous case.
- Case $\langle p, {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brt}^{p,\imath}} \langle p, ({}^{\imath}\{e\}AS_1); {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}, m\rangle$. For brevity, we use the shorthand notation $\underline{\mathsf{wh}}$ for ${}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}$. From $\underline{\mathsf{wh}} \in AStmt_{\mathrm{WF}}$, we have $\underline{\mathsf{wh}} \in Stmt$ and ${}^{\imath'}\mathsf{stop} \notin \underline{\mathsf{wh}}$ for any $\imath'$. Hence, $AS_1 \in Stmt$ and ${}^{\imath'}\mathsf{stop} \notin AS_1$ for any $\imath'$. Hence, $AS_1 \in AStmt_{\mathrm{WF}}$ can be deduced using the third rule. We obtain ${}^{\imath}\{e\}AS_1 \in AStmt_{\mathrm{WF}}$ using the second rule. That ${}^{\imath}\{e\}AS_1; \underline{\mathsf{wh}} \in AStmt_{\mathrm{WF}}$ can now be deduced using the first rule.
- Case $\langle p, {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}, m\rangle \xrightarrow{\mathsf{brf}^{p,\imath}} \langle p, {}^{\iota_{\mathrm{f}}}\mathsf{stop}, m\rangle$. Trivial.
- Case $\langle p, {}^{\imath}\{e\}AS_1, m\rangle \xrightarrow{\alpha} \langle p, {}^{\imath}\{e\}AS_1', m'\rangle$ due to $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$. From how ${}^{\imath}\{e\}AS_1$ could have been derived, we have $AS_1 \in AStmt_{\mathrm{WF}}$ and $\imath \neq \iota_{\mathrm{f}}$. By the induction hypothesis, we have $AS_1' \in AStmt_{\mathrm{WF}}$. Hence, ${}^{\imath}\{e\}AS_1' \in AStmt_{\mathrm{WF}}$ can be established using the second rule.

This induction completes the proof of the lemma. □

Security Type System for the Augmented Language. We devise a security type system for augmented statements. The typing judgment is

$$\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath'$$

Here, Ctx is a context such that for each program point $\imath$, if the augmented statement AS' at $\imath$ in AS is a conditional branch or a loop, then $Ctx(\imath) = (l', X', \imath')$ where $\imath'$ is the program point right after the branches or the loop body of AS', and l' and X' constitute the post-context at $\imath'$. If the augmented statement at $\imath$ is not a conditional branch or a loop, then $Ctx(\imath) = \bot$. If $\imath$ is not

$$\frac{}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{skip}\ (l, X) : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = \bot$$

$$\frac{\Phi(\imath) \triangleright \mathcal{V}[x \mapsto l \curlywedge \mathcal{V}[X \cup \mathit{fvs}(e)]] \preceq_{\Lambda(\imath') \cup \{x\}} \mathcal{V}[[e]@p/x@p]}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ x := {}^{\imath}e\ (l, X) : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = \bot$$

$$\frac{\begin{array}{c} l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}[X]) \preceq \mathcal{C}(c) \preceq l' \\ \Phi(\imath) \triangleright \mathcal{V}[(c.j \mapsto \mathcal{V}[\mathit{fvs}(e_j) \cup X])_j] \preceq_{\Lambda(\imath') \cup \{c.1, \ldots, c._{|c|}\}} (\mathcal{V} \uplus \mathcal{C})[([e_j]@p/c.j)_j] \end{array}}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{send}(c, \vec{e})\ (l', X) : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = \bot$$

$$\frac{\begin{array}{c} l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}[X]) \preceq \mathcal{C}(c) \preceq l' \\ \Phi(\imath) \triangleright \mathcal{V}[(x_j \mapsto \mathcal{C}(c.j) \curlywedge \mathcal{V}[X])_j] \preceq_{\Lambda(\imath') \cup \{\vec{x}\}} \mathcal{V}[(c.j/x_j@p)_j] \end{array}}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{recv}(c, \vec{x})\ (l', X) : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = \bot$$

$$\frac{\begin{array}{c} \mathcal{C}, \mathcal{V}^{p,e}_{l',X',\imath'}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ \mathit{AS}_1\ (l', X') : \imath' \\ \mathcal{C}, \mathcal{V}^{p,\neg e}_{l',X',\imath'}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ \mathit{AS}_2\ (l', X') : \imath' \end{array}}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{if}\ e\ \mathsf{then}\ \mathit{AS}_1\ \mathsf{else}\ \mathit{AS}_2\ \mathsf{fi}\ (l', X') : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = (l', X', \imath')$$

$$\frac{X \cup \mathit{fvs}(e) \subseteq X' \quad \mathcal{C}, \mathcal{V}^{p,e}_{l,X',\imath}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X')\ \mathit{AS}\ (l, X') : \imath}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\mathsf{while}\ e\ \mathsf{do}\ \mathit{AS}\ \mathsf{od}\ (l, X') : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = (l, X', \imath)$$

$$\frac{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ \mathit{AS}_1\ (l'', X'') : \mathit{fst}(\mathit{AS}_2) \quad \mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l'', X'')\ \mathit{AS}_2\ (l', X') : \imath'}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ \mathit{AS}_1; \mathit{AS}_2\ (l', X') : \imath'}$$

$$\frac{\mathcal{C}, \mathcal{V}^{p,e}_{l',X',\imath'}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ \mathit{AS}\ (l', X') : \imath'}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l, X)\ {}^{\imath}\{e\}\mathit{AS}\ (l', X') : \imath'}\ \text{if}\ \mathit{Ctx}(\imath) = (l', X', \imath')$$

$$\frac{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l'_1, X'_1)\ \mathit{AS}\ (l'_2, X'_2) : \imath' \quad l_1 \preceq l'_1 \quad l'_2 \preceq l_2 \quad X_1 \subseteq X'_1 \quad X'_2 \subseteq X_2}{\mathcal{C}, \mathcal{V}, \mathit{Ctx} \vdash_p^{\Phi,\Lambda} (l_1, X_1)\ \mathit{AS}\ (l_2, X_2) : \imath'}$$

Fig. 17. Selected Rules of the Type System that Records Context Information

in AS, then Ctx is unconstrained at $\imath$. The remaining elements in the judgment have analogous intuition to that of the corresponding elements of our typing judgment for (plain) statements in Section 5.

The typing rules for augmented statements are shown in Fig. 17. Each rule with an explicit program point $\imath$ in the conclusion contains a side condition that specifies the value of $\mathit{Ctx}(\imath)$. The rule for the augmented statement $\{e\}\mathit{AS}$ says: If AS can be typed with the specialized type environment $\mathcal{V}^{p,e}_{l',X',\imath'}$, and the set $X \cup \mathit{fvs}(e)$ of variables as part of the pre-context, then the augmented statement $\{e\}\mathit{AS}$ can be typed with the original type environment $\mathcal{V}$, and the set X of variables as part of the pre-context. The remaining elements of the typing rules are analogous to those of the typing rules presented in Section 5.

We establish Lemma 16 and Lemma 17 about the security type system for the augmented programming language.

Lemma 16. *If* $\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath'$, *then* $X \cup \mathit{fvs\text{-}cond}(AS) \subseteq X'$.

Proof. The proof is by induction on the derivation of

$$\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath'$$

In the cases where AS is $^{\imath}\mathsf{skip}$, $x := {}^{\imath}e$, $^{\imath}\mathsf{send}(c, \vec{e})$, or $^{\imath}\mathsf{recv}(c, \vec{x})$, we have $\mathit{fvs\text{-}cond}(AS) = \emptyset$, $X' = X$. Hence, it holds that $X \cup \mathit{fvs\text{-}cond}(AS) \subseteq X'$. The remaining cases are resolved as follows.

- Case $^{\imath}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$. We refer to $^{\imath}\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$ as $\underline{\mathsf{if}}$. By the typing of $\underline{\mathsf{if}}$, we have $\mathcal{C}, \mathcal{V}^{p,e}_{l',X',\imath'}, Ctx \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ AS_1\ (l', X') : \imath'$, and $\mathcal{C}, \mathcal{V}^{p,\neg e}_{l',X',\imath'}, Ctx \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ AS_2\ (l', X') : \imath'$. Hence, we have $X \cup \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X'$ and $X \cup \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_2) \subseteq X'$ using the induction hypothesis. Hence, we have $X \cup \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_1) \cup \mathit{fvs\text{-}cond}(AS_2) \subseteq X'$. Hence, we have $X \cup \mathit{fvs\text{-}cond}(\underline{\mathsf{if}}) \subseteq X'$.
- Case $^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}$. We refer to $^{\imath}\mathsf{while}\ e\ \mathsf{do}\ AS_1\ \mathsf{od}$ as $\underline{\mathsf{wh}}$. From the typing of $\underline{\mathsf{wh}}$ and the induction hypothesis, we obtain $X \cup \mathit{fvs}(e) \subseteq X'$, and $X' \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X'$. Hence, we have $X \cup \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X'$. Hence, we have $X \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X'$.
- Case $AS_1; AS_2$. From the typing of $AS_1; AS_2$ we have

  $$\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS_1\ (l'', X'') : \mathit{fst}(AS_2)$$
  $$\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l'', X'')\ AS_2\ (l', X') : \imath'$$

  Hence, we have $X \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X''$, and $X'' \cup \mathit{fvs\text{-}cond}(AS_2) \subseteq X'$ using the induction hypothesis. Hence, we have $X \cup \mathit{fvs\text{-}cond}(AS_1) \cup \mathit{fvs\text{-}cond}(AS_2) \subseteq X'$. Hence, we have $X \cup \mathit{fvs\text{-}cond}(AS_1; AS_2) \subseteq X'$.
- Case $^{\imath}\{e\}AS_1$. From the typing of $^{\imath}\{e\}AS_1$, we have $\mathcal{C}, \mathcal{V}^{p,e}_{l',X',\imath'}, Ctx \vdash_p^{\Phi,\Lambda} (l, X \cup \mathit{fvs}(e))\ AS_1\ (l', X') : \imath'$. Hence, we have $X \cup \mathit{fvs}(e) \cup \mathit{fvs\text{-}cond}(AS_1) \subseteq X'$. Hence, we have $X \cup \mathit{fvs\text{-}cond}(^{\imath}\{e\}AS_1) \subseteq X'$.
- Case sub-typing. It is straightforward to establish the desired result from the typing of AS and using the induction hypothesis.

The induction above completes the proof of this lemma. □

Lemma 17. *If* $\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath'$, $\imath \in \mathit{pts}(AS)$, *and* $Ctx(\imath) = (l_1, X_1, \imath_1)$, *then* $l \preceq l_1$ *and* $X \subseteq X_1$.

The proof of this lemma is by induction on the derivation of $\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath'$.

C.2 Soundness of Security Typing

To lighten the presentation of the remaining part of the soundness proof, we adopt the notational convention described as follows. For a formula ϕ, we write $[\![\phi]\!]_{agcnf}$ for $[\![\phi]\!]_{\mathit{strip\text{-}gcnf}(agcnf)}$. For a policy $P \in \mathit{Pol}$, we write $[\![P]\!]_{agcnf}$ for $[\![P]\!]_{\mathit{strip\text{-}gcnf}(agcnf)}$. In a similar spirit, we write $\lfloor\alpha\rfloor^{\mathcal{G}}_{agcnf}$ for $\lfloor\alpha\rfloor^{\mathcal{G}}_{\mathit{strip\text{-}gcnf}(agcnf)}$,

$\lfloor ci \rfloor^{\mathcal{G}}_{agcnf}$ for $\lfloor ci \rfloor^{\mathcal{G}}_{strip\text{-}gcnf(agcnf)}$, and $\lfloor atr \rfloor^{\mathcal{G}}$ for $\lfloor strip\text{-}tr(atr) \rfloor^{\mathcal{G}}$. Moreover, we take the channel policy environment $\mathcal{C}$ to be a global constant, and we avoid explicit parameterization of further definitions over $\mathcal{C}$.

Lemma 18. *Let agcnf be $\langle \ldots, \langle p, AS, m \rangle, \ldots \rangle$, and P be a policy in the implication normal form. The following two statements hold*

- *If $[\![e]\!]_m = tt$, then $[\![P^{p,e}]\!]_{agcnf} = [\![P]\!]_{agcnf}$.*
- *If $[\![e]\!]_m = ff$, then $[\![P^{p,\neg e}]\!]_{agcnf} = [\![P]\!]_{agcnf}$.*

Proof. We prove the first statement, and the proof of the second statement is analogous.

Let P be $\bigwedge_j (\phi_j \triangleright R_j)$. Then, $P^{p,e}$ is $\bigwedge_j (\phi'_j \triangleright R_j)$ where $\phi'_j = \mathsf{true}$ if $[e]@p \Rightarrow \phi_j$, $\phi'_j = \mathsf{false}$ if $unsat([e]@p \wedge \phi_j)$ and $\phi'_j = \phi_j$ otherwise.

Pick arbitrary i. With a case analysis, we show $[\![\phi_i]\!]_{agcnf} = [\![\phi'_i]\!]_{agcnf}$, which directly establishes the validity of the first statement in the lemma.

- Suppose $[e]@p \Rightarrow \phi_i$, and $\phi'_i = \mathsf{true}$. We have $[\![[e]@p]\!]_{agcnf} = tt$. Hence, $[\![\phi_i]\!]_{agcnf} = tt$. Hence, $[\![\phi_i]\!]_{agcnf} = [\![\phi'_i]\!]_{agcnf}$.
- Suppose $unsat([e]@p \wedge \phi_i)$, and $\phi'_i = \mathsf{false}$. Because $[\![[e]@p]\!]_{agcnf} = tt$, we have $[\![\phi_i]\!]_{agcnf} = ff$. Hence, $[\![\phi_i]\!]_{agcnf} = [\![\phi'_i]\!]_{agcnf}$.
- The case where $\phi'_i = \phi_i$ is trivial.

This completes the proof. □

We define the specialization of a policy with a given augmented statement and a given context as follows.

$$\begin{aligned}
\mathcal{V}^{p,AS_1;AS_2}_{Ctx} &\triangleq \mathcal{V}^{p,AS_1}_{Ctx} \\
\mathcal{V}^{p,\imath\{e\}AS}_{Ctx} &\triangleq (\mathcal{V}^{p,e}_{Ctx(\imath)})^{p,AS}_{Ctx} && \text{if } \neg(\exists \imath' : {}^{\imath'}\mathsf{stop} \in atoms(AS)) \\
\mathcal{V}^{p,AS}_{Ctx} &\triangleq \mathcal{V} && \text{if } AS \text{ is not one of the above}
\end{aligned}$$

We define the syntactical "highness" for a process.

Definition 18 (Syntactically high processes).

- *$high^{\mathcal{V},Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i, X, l', X', \imath')$ if and only if for $p = prin\text{-}of(agcnf[i])$ and $AS = astmt\text{-}of(agcnf[i])$, it holds that*

$$\begin{aligned}
X \subseteq \mathcal{X} \wedge \exists l : \mathcal{C}, \mathcal{V}, Ctx \vdash^{\Phi,\Lambda}_{p} (l, X)\ AS\ (l', X') : \imath' \wedge \\
(l \cap \mathcal{G} = \emptyset \vee \exists x \in X : [\![\mathcal{V}^{p,AS}_{Ctx}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset)
\end{aligned}$$

- *$high^{\mathcal{V},Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i, l', X', \imath')$ if and only if $\exists X : high^{\mathcal{V},Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i, X, l', X', \imath')$.*
- *$high^{\mathcal{V},Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i)$ if and only if $\exists l', X', \imath' : high^{\mathcal{V},Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i, l', X', \imath')$*

Intuitively, a process is syntactically high wrt. the group $\mathcal{G}$ of principals if the augmented statement of the process can be typed with a pre-context that captures unobservability by any principal in $\mathcal{G}$.

We next define the *low-equality* of the states of two processes, the *low-equivalence* of two processes, and the *low-equivalence* of two systems. We write $lvars(\mathcal{V}, \mathcal{G})$ for the set $\{x \in Var \mid frs(\mathcal{V}(x)) \cap \mathcal{G} \neq \emptyset\}$. Hence, $lvars(\mathcal{V}, \mathcal{G})$ is the set of all variables that have a security type in INF where all the reader sets contain at least one principal in $\mathcal{G}$. A variable in $lvars(\mathcal{V}, \mathcal{G})$ is always observable by some principal in $\mathcal{G}$.

Definition 19 (Low-equality of process states). *For augmented global configurations* $agcnf_1$ *and* $agcnf_2$, $agcnf_1 \overset{\mathcal{V}, Ctx, \mathcal{G}}{\underset{\Phi, \Lambda, i}{=}} agcnf_2$ *holds if and only if there exist* n *with* $i \in \{1, \ldots, n\}$, $p_1, \ldots, p_n$, $AS_{11}, \ldots, AS_{n1}$, $AS_{12}, \ldots, AS_{n2}$, $m_{11}, \ldots, m_{n1}$, $m_{12}, \ldots, m_{n2}$, *such that* $agcnf_1 = \langle\langle p_1, AS_{11}, m_{11}\rangle, \ldots, \langle p_n, AS_{n1}, m_{n1}\rangle\rangle$, $agcnf_2 = \langle\langle p_1, AS_{12}, m_{12}\rangle, \ldots, \langle p_n, AS_{n2}, m_{n2}\rangle\rangle$, *and*

$$\forall x \in Var : \left([\![\mathcal{V}_{Ctx}^{p_i, AS_{i1}}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![\mathcal{V}_{Ctx}^{p_i, AS_{i2}}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset \right)$$
$$\wedge \left(\left(\begin{array}{l} x \in \Lambda(afst(AS_{i1})) \cap \Lambda(afst(AS_{i2})) \\ \quad \cup\, lvars(\mathcal{V}, \mathcal{G}) \\ \wedge\ [\![\mathcal{V}_{Ctx}^{p_i, AS_{i1}}(x)]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset \end{array} \right) \Rightarrow m_{i1}(x) = m_{i2}(x) \right)$$

Hence, the states of the i-th processes in the augmented global configurations $agcnf_1$ and $agcnf_2$ are low-equal wrt. the group $\mathcal{G}$ of principals if each variable has the same confidentiality level wrt. $\mathcal{G}$ in $agcnf_1$ and $agcnf_2$, and each variable that is low and live, or that is in $lvars(\mathcal{V}, \mathcal{G})$, has the same value in $agcnf_1$ and in $agcnf_2$.

Definition 20 (Low-equivalence of processes).

$$\frac{\begin{array}{c} agcnf_1 \overset{\mathcal{V}, Ctx, \mathcal{G}}{\underset{\Phi, \Lambda, i}{=}} agcnf_2 \\ prin\text{-}of(agcnf_1[i]) = p = prin\text{-}of(agcnf_2[i]) \\ high_{\Phi, \Lambda, \mathcal{X}}^{\mathcal{V}, Ctx, \mathcal{G}}(agcnf_1, i, l_1', X_1', \iota_{\mathrm{f}}) \wedge X_1' \subseteq \mathcal{X} \vee \iota_{\mathrm{f}} \mathsf{stop} \in atoms(astmt\text{-}of(agcnf_1[i])) \\ high_{\Phi, \Lambda, \mathcal{X}}^{\mathcal{V}, Ctx, \mathcal{G}}(agcnf_2, i, l_2', X_2', \iota_{\mathrm{f}}) \wedge X_2' \subseteq \mathcal{X} \vee \iota_{\mathrm{f}} \mathsf{stop} \in atoms(astmt\text{-}of(agcnf_2[i])) \end{array}}{agcnf_1 \overset{\mathcal{V}, Ctx, \mathcal{G}}{\underset{\Phi, \Lambda, \mathcal{X}, i}{\approx}} agcnf_2}$$

$$\frac{\begin{array}{c} agcnf_1 \overset{\mathcal{V}, Ctx, \mathcal{G}}{\underset{\Phi, \Lambda, i}{=}} agcnf_2 \\ prin\text{-}of(agcnf_1[i]) = p = prin\text{-}of(agcnf_2[i]) \\ astmt\text{-}of(agcnf_1[i]) = AS = astmt\text{-}of(agcnf_2[i]) \\ \mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi, \Lambda} (l, X)\ AS\ (l', X') : \iota_{\mathrm{f}} \wedge X' \subseteq \mathcal{X} \vee \iota_{\mathrm{f}} \mathsf{stop} \in atoms(AS) \\ \neg high_{\Phi, \Lambda, \mathcal{X}}^{\mathcal{V}, Ctx, \mathcal{G}}(agcnf_1, i, l', X', \iota_{\mathrm{f}}) \qquad \neg high_{\Phi, \Lambda, \mathcal{X}}^{\mathcal{V}, Ctx, \mathcal{G}}(agcnf_2, i, l', X', \iota_{\mathrm{f}}) \end{array}}{agcnf_1 \overset{\mathcal{V}, Ctx, \mathcal{G}}{\underset{\Phi, \Lambda, \mathcal{X}, i}{\approx}} agcnf_2}$$

Hence, the i-th processes in the augmented global configurations $agcnf_1$ and $agcnf_2$ are low-equivalent wrt. the group $\mathcal{G}$ of principals if the following two conditions are met:

1. the states of the i-th processes in $agcnf_1$ and $agcnf_2$ are low-equal wrt. $\mathcal{G}$,
2. both processes either are high or have terminated, or neither process is high and both processes have the same augmented statement.

Definition 21 (Low-equivalence of systems).

$$\frac{\forall i : agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2 \qquad nip(\mathcal{C}, \vec{\mathcal{V}}) \quad no\text{-}upd(cstmt\text{-}of(strip\text{-}gcnf(agcnf_1)), \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}) \quad \underline{no\text{-}upd(cstmt\text{-}of(strip\text{-}gcnf(agcnf_2)), \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})}}{agcnf_1 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\simeq}} agcnf_2}$$

Hence, two systems represented by the augmented global configurations $agcnf_1$ and $agcnf_2$ are low-equivalent if the i-th processes in $agcnf_1$ and $agcnf_2$ are low-equivalent for each i, and the conditions *nip* and *no-upd* are satisfied. In the above, $cstmt\text{-}of(gcnf)$ is the concurrent statement of $gcnf$, for any global configuration $gcnf$.

Lemma 19. *If* $agcnf[i] = alcnf$, $alcnf \xrightarrow{\alpha} alcnf'$, $\alpha \in \{c!\vec{v}, c?\vec{v} \mid \mathcal{C}(c) \cap \mathcal{G} = \emptyset \wedge \vec{v} \in Val^*\}$, $\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)$ $astmt\text{-}of(alcnf)$ $(l', X') : \imath'$, *and* $X' \subseteq \mathcal{X}$, *then we have* $\exists X_0 \supseteq X : high^{\mathcal{V}, Ctx, \mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf, i, X_0, l', X', \imath')$.

The proof of this lemma is by induction on the derivation of $\mathcal{C}, \mathcal{V}, Ctx \vdash_p^{\Phi,\Lambda} (l, X)$ $astmt\text{-}of(alcnf)$ $(l', X') : \imath'$.

Lemma 20. *For all* $agcnf_1$ *and* $agcnf_2$ *such that* $agcnf_1[j] = \langle p_j, AS_1, m_1\rangle$ *and* $agcnf_2[j] = \langle p_j, AS_2, m_2\rangle$, *if* $nip(\mathcal{C}, \vec{\mathcal{V}})$ *holds, for all* k, *and* $x \in lvars(\mathcal{V}_k, \mathcal{G})$, *it holds that* $[\![x@p_k]\!]_{agcnf_1} = [\![x@p_k]\!]_{agcnf_2}$, *and* $(\mathcal{V}_j)^{p_j, AS_1}_{Ctx}(x) = (\mathcal{V}_j)^{p_j, AS_2}_{Ctx}(x)$, *then* $\forall x \in Var : [\![(\mathcal{V}_j)^{p_j, AS_1}_{Ctx}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_j)^{p_j, AS_2}_{Ctx}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset$.

Proof. If $\mathcal{G} = \emptyset$, then the conclusion of the lemma obviously holds. In the following, we assume that $\mathcal{G} \neq \emptyset$.

Pick an arbitrary $x \in Var$. Let $inf(\mathcal{V}_j(x))$ be denoted by P. Let $(\mathcal{V}_j)^{p_j, AS_1}_{Ctx}(x)$ be denoted by P'. Then, $P = (\phi_1 \triangleright R_1) \curlywedge \ldots \curlywedge (\phi_n \triangleright R_n)$ for some n, $\phi_1, \ldots, \phi_n$, $R_1, \ldots, R_n$. Furthermore, P' is of the form $(\phi'_1 \triangleright R_1) \curlywedge \ldots \curlywedge (\phi'_n \triangleright R_n)$, where $\phi'_i \in \{\phi_i, \mathsf{true}, \mathsf{false}\}$ for each $i \in \{1, \ldots, n\}$.

Assume per absurdum, that $[\![P']\!]_{agcnf_1} \cap \mathcal{G}$ and $[\![P']\!]_{agcnf_2} \cap \mathcal{G}$ differ in emptiness. Further assume, w.l.o.g., that $[\![P']\!]_{agcnf_1} \cap \mathcal{G} = \emptyset$. Then, there is an index set I, such that $(\bigcap_{i \in I} R_i) \cap \mathcal{G} = \emptyset$. Since $\mathcal{G} \neq \emptyset$, there exists some principal p such that $p \in \mathcal{G}$, and $p \in Pr \setminus (\bigcap_{i \in I} R_i)$. Hence, $p \in Pr \setminus (\bigcap_{1 \leq i \leq n} R_i)$. Hence, we have $p \in \bigcap_{1 \leq j \leq n} \bigcap_{u \in fvs(\phi_j)} frs\text{-}cv(\mathcal{C}, \vec{\mathcal{V}}, u)$. Hence, for each $j \in \{1, \ldots, n\}$, and each $y@p_h \in fvs(\phi_j)$, we have $frs(\mathcal{V}_h(y)) \cap \mathcal{G} \neq \emptyset$. Hence, for each $j \in \{1, \ldots, n\}$, and each $y@p_h \in fvs(\phi'_j)$, we have $y \in lvars(\mathcal{V}_h, \mathcal{G})$. Hence, for each $y@p_h$ in the conditions of P', we have $[\![y@p_h]\!]_{agcnf_1} = [\![y@p_h]\!]_{agcnf_2}$ according to the hypothesis of the lemma. Since policies of variables depend only on variables (but not on channel components, as specified in Section 5.1), we must have $[\![P']\!]_{agcnf_1} \cap \mathcal{G} = [\![P']\!]_{agcnf_2} \cap \mathcal{G}$. This contradicts the differing emptiness of $[\![P']\!]_{agcnf_1} \cap \mathcal{G}$ and $[\![P']\!]_{agcnf_2} \cap \mathcal{G}$. This completes the proof of the lemma. □

Lemma 21. *The following two statements hold for all $\mathcal{G} \neq \emptyset$.*

1. *If $l \cap \mathcal{G} = \emptyset \vee \exists x \in X : [\![\mathcal{V}_{Ctx}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$, then $\mathcal{V}_{l,X,\imath}^{p,e} = \mathcal{V}$ for all e.*
2. *If $high_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V},Ctx,\mathcal{G}}(agcnf, i)$, $AS = astmt\text{-}of(agcnf[i])$, then $\mathcal{V}_{Ctx}^{p,AS} = \mathcal{V}$.*

Proof. We first show item 1 of the lemma. If $l \cap \mathcal{G} = \emptyset$, and $\mathcal{G} \neq \emptyset$, then we have $l \neq Pr$, and thus $l \cap \bigcap_{x \in X} frs(\mathcal{V}(x)) \neq Pr$. On the other hand, if there exists some $x_0 \in X$ such that $[\![\mathcal{V}_{Ctx}^{p,AS}(x_0)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$, and $\mathcal{G} \neq \emptyset$, then we have $frs(\mathcal{V}(x_0)) \neq Pr$. Hence, we also have $l \cap \bigcap_{x \in X} frs(\mathcal{V}(x)) \neq Pr$. Hence, for each variable y, $\mathcal{V}_{l,X,\imath}^{p,e}(y) = \mathcal{V}(y)$. Hence, we have $\mathcal{V}_{l,X,\imath}^{p,e} = \mathcal{V}$.

We next show item 2 of the lemma. Assume per absurdum that $\mathcal{V}_{Ctx}^{p,AS} \neq \mathcal{V}$. Then, there exists some program point $\imath \in pts(AS)$, and some expression e in AS, such that $Ctx(\imath) = (l', X', \imath')$, and $\mathcal{V}_{l',X',\imath'}^{p,e} \neq \mathcal{V}$. We have $l' \cap \mathcal{G} = \emptyset \vee \exists x \in X' : [\![\mathcal{V}_{Ctx}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$, because of $high_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V},Ctx,\mathcal{G}}(agcnf, i)$, $\imath \in pts(AS)$, $Ctx(\imath) = (l', X', \imath')$, and Lemma 17. Then, we have $\mathcal{V}_{l',X',\imath'}^{p,e} = \mathcal{V}$ due to item 1 of the lemma, and we have a contradiction. □

Lemma 22. *If $AS \in AStmt_{\mathrm{WF}}$, $[\![\Phi(afst(AS))]\!]_{agcnf} = tt$, $nip(\mathcal{C}, \vec{\mathcal{V}})$, $X' \subseteq \mathcal{X}$, $high_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V}_i,Ctx,\mathcal{G}}(agcnf, i, X, l', X', \imath')$, $agcnf[i] = \langle p, AS, m\rangle$, $\langle p, AS, m\rangle \xrightarrow{\alpha} \langle p, AS', m'\rangle$, and $agcnf' = agcnf[i \mapsto \langle p, AS', m'\rangle]$, then*

1. *$\forall x \in Var : [\![(\mathcal{V}_i)_{Ctx}^{p,AS'}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset \;\Leftrightarrow\; [\![(\mathcal{V}_i)_{Ctx}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$,*
2. *if $\alpha = c!\vec{v}$ or $\alpha = c?\vec{v}$ for some c and $\vec{v}$, then $\mathcal{C}(c) \cap \mathcal{G} = \emptyset$,*
3. *$\forall x \in upd\text{-}next(AS) : [\![(\mathcal{V}_i)_{Ctx}^{p,AS'}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$, and*
4. *$\imath_{\mathsf{f}}\mathsf{stop} \in atoms(AS')$ or there exists some $X_0 \supseteq X$ for which it holds that $high_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V}_i,Ctx,\mathcal{G}}(agcnf', i, X_0, l', X', \imath')$.*

Proof. By $high_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V}_i,Ctx,\mathcal{G}}(agcnf, i, X, l', X', \imath')$, there exists some l such that

$$X \subseteq \mathcal{X} \tag{14}$$

$$\mathcal{C}, \mathcal{V}_i, Ctx \vdash_p^{\Phi,\Lambda} (l, X)\ AS\ (l', X') : \imath' \tag{15}$$

$$l \cap \mathcal{G} = \emptyset \vee \exists x \in X : [\![(\mathcal{V}_i)_{Ctx}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset \tag{16}$$

We perform an induction on the derivation of (15). Only selected cases of this induction are presented below.

– Case $^{\imath}\mathsf{recv}(c, \vec{x})$. From (15), we have

$$l \curlywedge (\Phi(\imath) \triangleright \mathcal{V}_i[X]) \preceq \mathcal{C}(c) \tag{17}$$

$$\forall j : \Phi(\imath) \triangleright \mathcal{C}(c.j) \curlywedge \mathcal{V}_i[X] \preceq \mathcal{V}_i(x_j)[(c.j/x_j@p)_j] \tag{18}$$

We have $\mathcal{C}(c) \cap \mathcal{G} = \emptyset$ by (16), (17), and $[\![\Phi(\imath)]\!]_{agcnf} = tt$, thereby establishing condition 2 of the lemma. Hence, $[\![\mathcal{C}(c.j)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$ because the content of communication is no less confidential than the presence of communication for each polyadic channel. Hence, we have $\forall j : [\![(\mathcal{V}_i)_{Ctx}^{p,\imath_{\mathsf{f}}\mathsf{stop}}(x_j)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$ because of (18). This establishes condition 3 of the lemma.

Pick an arbitrary $y \in \mathit{lvars}(\mathcal{V}_i, \mathcal{G})$. We have $[\![(\mathcal{V}_i)^{p,\iota\mathsf{f}\,\mathsf{stop}}_{Ctx}(y)]\!]_{agcnf'} \cap \mathcal{G} \neq \emptyset$. Hence, we have $y \notin \{\vec{x}\}$, and it holds that $[\![y@p]\!]_{agcnf} = [\![y@p]\!]_{agcnf'}$. We also have $(\mathcal{V}_i)^{p,\imath\,\mathsf{recv}(c,\vec{x})}_{Ctx} = (\mathcal{V}_i)^{p,\iota\mathsf{f}\,\mathsf{stop}}_{Ctx}$. Hence, we can obtain

$$\forall x' \in \mathit{Var} : [\![(\mathcal{V}_i)^{p,\iota\mathsf{f}\,\mathsf{stop}}_{Ctx}(x')]\!]_{agcnf'} \cap \mathcal{G} = \emptyset \;\Leftrightarrow\; [\![(\mathcal{V}_i)^{p,\imath\,\mathsf{recv}(c,\vec{x})}_{Ctx}(x')]\!]_{agcnf} \cap \mathcal{G} = \emptyset$$

using Lemma 20, thereby establishing condition 1 of the lemma.
Condition 4 of the lemma trivially holds.

– Case $\imath\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$.
Condition 2 and condition 3 of the lemma vacuously hold.
In the following, we write $\underline{\mathsf{if}}$ as a shorthand notation for

$$\imath\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$$

We have $AS_1 \in \mathit{Stmt}$ and $AS_2 \in \mathit{Stmt}$ because $\underline{\mathsf{if}} \in \mathit{AStmt}_{\mathrm{WF}}$.
Assume w.l.o.g. $[\![e]\!]_m = tt$. We have $AS' = \imath\{e\}AS_1$.
Pick an arbitrary $x \in \mathit{Var}$. We show $[\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} = [\![(\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)]\!]_{agcnf}$ with a case analysis on whether $(\mathcal{V}_i)^{p,AS'}_{Ctx}(x) = (\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)$.

- Suppose $(\mathcal{V}_i)^{p,AS'}_{Ctx}(x) = (\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)$ holds. Because $agcnf'$ has the same memory states as those of $agcnf$, we directly have $[\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} = [\![(\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)]\!]_{agcnf}$.
- Suppose $(\mathcal{V}_i)^{p,AS'}_{Ctx}(x) \neq (\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)$. Let $P \triangleq \mathit{inf}((\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x))$. We have $P = \mathit{inf}(\mathcal{V}_i(x))$. Hence, $(\mathcal{V}_i)^{p,AS'}_{Ctx}(x) = P^{p,e}$. We have $[\![\mathcal{V}_i{}^{p,\underline{\mathsf{if}}}_{Ctx}(x)]\!]_{agcnf} = [\![P]\!]_{agcnf}$ by Lemma 1. We have $[\![P]\!]_{agcnf} = [\![P^{p,e}]\!]_{agcnf}$ by Lemma 18 and $[\![e]\!]_m = tt$. We have $[\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf} = [\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'}$ because $agcnf$ and $agcnf'$ have the same memory states. Hence, we have $[\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} = [\![(\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)]\!]_{agcnf}$.

The reasoning above shows

$$\forall x \in \mathit{Var} : [\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p,\underline{\mathsf{if}}}_{Ctx}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset \tag{19}$$

From (15), we have $\mathcal{C}, (\mathcal{V}_i)^{p,e}_{l',X',\imath'}, Ctx \vdash^{\Phi,\Lambda}_p (l, X \cup \mathit{fvs}(e))\ AS_1\ (l', X') : \imath'$. Hence, we have $\mathcal{C}, \mathcal{V}_i, Ctx \vdash^{\Phi,\Lambda}_p (l, X)\ AS'\ (l', X') : \imath'$. We have $l \cap \mathcal{G} = \emptyset \vee \exists x \in X : [\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$ because of (16) and (19). Hence, we have $\mathit{high}^{\mathcal{V}_i,Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf', i, X, l', X', \imath')$.

– Case $AS_1; AS_2$.
From (15), we have

$$\mathcal{C}, \mathcal{V}_i, Ctx \vdash^{\Phi,\Lambda}_p (l, X)\ AS_1\ (l'', X'') : \mathit{afst}(AS_2) \tag{20}$$

$$\mathcal{C}, \mathcal{V}_i, Ctx \vdash^{\Phi,\Lambda}_p (l'', X'')\ AS_2\ (l', X') : \imath' \tag{21}$$

From $AS_1; AS_2 \in \mathit{AStmt}_{\mathrm{WF}}$, we have $AS_1 \in \mathit{AStmt}_{\mathrm{WF}}$. We have $X'' \subseteq X'$ by (21) and Lemma 16. Hence, we have $X'' \subseteq \mathcal{X}$. Let $agcnf_1 = agcnf[i \mapsto AS_1]$. Then, $agcnf_1$ and $agcnf$ have the same memory states. We have $\mathit{high}^{\mathcal{V}_i,Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf_1, i, X, l'', X'', \mathit{afst}(AS_2))$ because of (20), (14), (16), and

$$[\![(\mathcal{V}_i)^{p,AS}_{Ctx}]\!]_{agcnf} = [\![(\mathcal{V}_i)^{p,AS_1}_{Ctx}]\!]_{agcnf_1}$$

We have $\langle p, AS_1, m\rangle \xrightarrow{\alpha} \langle p, AS_1', m'\rangle$.
Let $agcnf_1' = agcnf_1[i \mapsto \langle p, AS_1', m'\rangle]$. By the induction hypothesis, we have

$$\forall x \in \mathit{Var} : [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \;\Leftrightarrow\; [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \tag{22}$$

$$(\exists c, \vec{v} : \alpha = c!\vec{v} \vee \alpha = c?\vec{v}) \Rightarrow \mathcal{C}(c) \cap \mathcal{G} = \emptyset \tag{23}$$

$$\forall x \in \mathit{upd\text{-}next}(AS_1) : [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \tag{24}$$

$$\iota_{\mathrm{f}}\mathsf{stop} \in \mathit{atoms}(AS_1') \vee \exists X_1 \supseteq X \wedge \mathit{high}_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V}_i,\mathit{Ctx},\mathcal{G}}(agcnf_1', i, X_1, l'', X'', \mathit{afst}(AS_2)) \tag{25}$$

Condition 2 of the lemma is established using (23).
To show conditions 1, 3, and 4 of the lemma, we make a case analysis on whether $\exists \vec{C} : AS_1' = \mathit{blks}(\vec{C}, \iota_{\mathrm{f}}\mathsf{stop})$.

- Suppose $\neg(\exists \vec{C} : AS_1' = \mathit{blks}(\vec{C}, \iota_{\mathrm{f}}\mathsf{stop}))$.
  We have $AS' = AS_1'; AS_2$, and the reasoning is straightforward by using the conditions (22), (24), and (25).
- Suppose $\exists \vec{C} : AS_1' = \mathit{blks}(\vec{C}, \iota_{\mathrm{f}}\mathsf{stop})$.
  We have $AS' = AS_2$. It holds that $AS_2 \in \mathit{Stmt}$ because $AS_1; AS_2 \in \mathit{AStmt}_{\mathrm{WF}}$.
  In case $\mathcal{G} = \emptyset$, it is straightforward to establish conditions 1, 3, and 4 of the lemma. Hence, we assume $\mathcal{G} \neq \emptyset$ below.
  We have $(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'} = \mathcal{V}_i$ because $\exists \vec{C} : AS_1' = \mathit{blks}(\vec{C}, \iota_{\mathrm{f}}\mathsf{stop})$.
  Hence, for an arbitrary variable $x \in \mathit{Var}$, we have

$$\begin{aligned}
& [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS'}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset \\
\Leftrightarrow\; & [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS'}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \\
\Leftrightarrow\; & [\![\mathcal{V}_i(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset && \text{(because } AS' = AS_2 \in \mathit{Stmt}\text{)} \\
\Leftrightarrow\; & [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset && \text{(because of } (\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'} = \mathcal{V}_i\text{)} \\
\Leftrightarrow\; & [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset && \text{(because of (22))} \\
\Leftrightarrow\; & [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \\
\Leftrightarrow\; & [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset
\end{aligned}$$

  This establishes condition 1 of the lemma.
  Pick an arbitrary $x \in \mathit{upd\text{-}next}(AS)$. We have $x \in \mathit{upd\text{-}next}(AS_1)$. We have $[\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS_1'}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset$ because of (24). From the reasoning above that establishes condition 1, we have $[\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS'}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$, establishing condition 3 of the lemma.
  We have $l \preceq l''$ and $X \subseteq X''$ by (20) and Lemma 16.
  We have previously shown $X'' \subseteq X'$. Hence, we have $X'' \subseteq \mathcal{X}$ because of $\mathcal{X}' \subseteq \mathcal{X}$. From (16), either $l \cap \mathcal{G} = \emptyset$, in which case we have $l'' \cap \mathcal{G} = \emptyset$, or $\exists x \in X : [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS}(x)]\!]_{agcnf} \cap \mathcal{G} = \emptyset$, in which case we have $\exists x \in X'' : [\![(\mathcal{V}_i)_{\mathit{Ctx}}^{p,AS'}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$ because of $X \subseteq X''$ and condition 1 of the lemma. Hence, we can establish $\mathit{high}_{\Phi,\Lambda,\mathcal{X}}^{\mathcal{V}_i,\mathit{Ctx},\mathcal{G}}(agcnf', i, X'', l', X', \iota')$ based on (21).

– Case ${}^{\imath}\{e\}AS_1$. We have $\langle p, AS_1, m\rangle \xrightarrow{\alpha} \langle p, AS_1', m'\rangle$, and $AS' = {}^{\imath}\{e\}AS_1'$. Let $agcnf_1 = agcnf[i \mapsto \langle p, AS_1, m\rangle]$. By (15), we have $\mathcal{C}, (\mathcal{V}_i)^{p,e}_{l',X',\imath'}, Ctx \vdash^{\Phi,\Lambda}_p (l, X \cup fvs(e))\ AS_1\ (l', X') : \imath'$. Hence, we have $X \cup fvs(e) \subseteq X'$, because of Lemma 16. Hence, we have $X \cup fvs(e) \subseteq \mathcal{X}$ because of $X' \subseteq \mathcal{X}$. Let $\mathcal{V}' = (\mathcal{V}_i)^{p,e}_{l',X',\imath'}$. We have $l \cap \mathcal{G} = \emptyset \vee \exists x \in X \cup fvs(e) : [\![(\mathcal{V}')^{p,AS_1}_{Ctx}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset$ because of (16), the fact that $agcnf_1$ and $agcnf$ have the same memory states, and the definition of $(\mathcal{V}_i)^{p,AS}_{Ctx}$. Hence, we have $high^{\mathcal{V}',Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf_1, i, X \cup fvs(e), l', X', \imath')$. We have $[\![\Phi(afst(AS_1))]\!]_{agcnf_1} = tt$ because of $[\![\Phi(afst(AS))]\!]_{agcnf} = tt$, $AS = {}^{\imath}\{e\}AS_1$, the definition of $afst$, and the fact that $agcnf_1$ and $agcnf$ have the same memory states. Hence, by the induction hypothesis, we have

$$\forall x \in Var : [\![(\mathcal{V}')^{p,AS_1'}_{Ctx}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \ \Leftrightarrow\ [\![(\mathcal{V}')^{p,AS_1}_{Ctx}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \tag{26}$$

$$(\exists c, \vec{v} : \alpha = c!\vec{v} \vee \alpha = c?\vec{v}) \Rightarrow \mathcal{C}(c) \cap \mathcal{G} = \emptyset \tag{27}$$

$$\forall x \in upd\text{-}next(AS_1) : [\![(\mathcal{V}')^{p,AS_1'}_{Ctx}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \tag{28}$$

$${}^{\iota_f}\mathsf{stop} \in atoms(AS_1') \vee \exists X_1 \supseteq X \cup fvs(e) : high^{\mathcal{V}',Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf_1', i, X_1, l', X', \imath') \tag{29}$$

where $agcnf_1' = agcnf'[i \mapsto \langle p, AS_1', m'\rangle]$.
We have $(\mathcal{V}')^{p,AS_1'}_{Ctx} = (\mathcal{V}_i)^{p,{}^{\imath}\{e\}AS_1'}_{Ctx}$. If ${}^{\iota_f}\mathsf{stop} \notin atoms(AS_1')$, then this equality can be established directly. Otherwise, we can obtain $\mathcal{V}' = \mathcal{V}_i$ by using Lemma 21, and this equality can be shown by observing that $AS_1' = blks(\vec{C}', {}^{\iota_f}\mathsf{stop})$ for some $\vec{C}'$ (since it can be shown that ${}^{\imath}\{e\}AS_1' \in AStmt_{\mathrm{WF}}$). Condition 1 of the lemma can be established by the use of (26), $(\mathcal{V}')^{p,AS_1'}_{Ctx} = (\mathcal{V}_i)^{p,{}^{\imath}\{e\}AS_1'}_{Ctx}$, $(\mathcal{V}')^{p,AS_1}_{Ctx} = (\mathcal{V}_i)^{p,{}^{\imath}\{e\}AS_1}_{Ctx}$, and the fact that $agcnf$ (resp. $agcnf'$) and $agcnf_1$ (resp. $agcnf_1'$) have the same memory states. Condition 2 of the lemma can be established using (27). Condition 3 of the lemma can be established using (28), $upd\text{-}next(AS) = upd\text{-}next(AS_1)$, $(\mathcal{V}')^{p,AS_1'}_{Ctx} = (\mathcal{V}_i)^{p,{}^{\imath}\{e\}AS_1'}_{Ctx}$, and the fact that $agcnf'$ and $agcnf_1'$ have the same memory states.
From (29), either ${}^{\iota_f}\mathsf{stop} \in atoms(AS_1')$ or there exists some $X_1 \supseteq X \cup fvs(e)$, such that $high^{\mathcal{V}_i,Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf_1', i, X_1, l', X', \imath')$. In the former case, we have ${}^{\iota_f}\mathsf{stop} \in atoms(AS')$. In the latter case, there exists some l_1 such that

$$\begin{aligned}
&X_1 \subseteq \mathcal{X}\\
&\mathcal{C}, \mathcal{V}', Ctx \vdash^{\Phi,\Lambda}_p (l_1, X_1)\ AS_1'\ (l', X') : \imath'\\
&l_1 \cap \mathcal{G} = \emptyset \vee \exists x \in X_1 : [\![(\mathcal{V}')^{p,AS_1'}_{Ctx}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset
\end{aligned}$$

Hence, we have $\mathcal{C}, \mathcal{V}_i, Ctx \vdash^{\Phi,\Lambda}_p (l_1, X_1)\ AS'\ (l', X') : \imath'$, and $l_1 \cap \mathcal{G} = \emptyset \vee \exists x \in X_1 : [\![(\mathcal{V}_i)^{p,AS'}_{Ctx}(x)]\!]_{agcnf'} \cap \mathcal{G} = \emptyset$. Hence, we have
$high^{\mathcal{V}_i,Ctx,\mathcal{G}}_{\Phi,\Lambda,\mathcal{X}}(agcnf', i, X_1, l', X', \imath')$
where $X_1 \supseteq X$. Condition 4 of the lemma is now established.
– Case subtyping. The reasoning for this case is straightforward using the induction hypothesis.

This induction completes the proof of the lemma. □

Lemma 23. *For* $agcnf_1$ *and* $agcnf_2$*, if* $agcnf_1[i] = \langle p_i, AS_1, m_1\rangle$*,* $agcnf_2[i] = \langle p_i, AS_2, m_2\rangle$*,* $agcnf'_1 = agcnf_1[i \mapsto \langle p_i, AS'_1, m_1\rangle]$*,* $agcnf'_2 = agcnf_2[i \mapsto \langle p_i, AS'_2, m_2\rangle]$*,* $j \neq i$*, and* $agcnf_1 \overset{\mathcal{V},Ctx,\mathcal{G}}{\underset{\Phi,\Lambda,\mathcal{X},j}{\approx}} agcnf_2$*, then* $agcnf'_1 \overset{\mathcal{V},Ctx,\mathcal{G}}{\underset{\Phi,\Lambda,\mathcal{X},j}{\approx}} agcnf'_2$*.*

Proof. By Definition 19 and Definition 20, the i-th augmented statement is unused in deciding $\cdot \overset{\mathcal{V},Ctx,\mathcal{G}}{\underset{\Phi,\Lambda,\mathcal{X},j}{\approx}} \cdot$ for two arbitrary augmented global configurations, if $j \neq i$. Hence, for $agcnf'_1$ and $agcnf'_2$ that differ from $agcnf_1$ and $agcnf_2$, respectively, only in the i-th augmented statement, relatedness in $\cdot \overset{\mathcal{V},Ctx,\mathcal{G}}{\underset{\Phi,\Lambda,\mathcal{X},j}{\approx}} \cdot$ is preserved. □

Lemma 24. *If* $nip(\mathcal{C}, \vec{\mathcal{V}})$ *holds,* $\forall k : agcnf_1 \overset{\mathcal{V}_k,Ctx_k,\mathcal{G}}{\underset{\Phi_k,\Lambda_k,k}{=}} agcnf_2$*,* $agcnf_1[i] \xrightarrow{\alpha} \langle p_i, AS'_{i1}, m'_{i1}\rangle$*,* $\forall x \in upd\text{-}next(AS_{i1}) : [\![(\mathcal{V}_i)^{p_i,AS'_{i1}}_{Ctx}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset$*,* $agcnf'_1 = agcnf_1[i \mapsto \langle p_i, AS'_{i1}, m'_{i1}\rangle]$*, then* $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j,Ctx_j,\mathcal{G}}{\underset{\Phi_j,\Lambda_j,j}{=}} agcnf_2$*.*

Proof. Pick an arbitrary k, and $x \in lvars(\mathcal{V}_k, \mathcal{G})$. If $k \neq i$, then we have $[\![x@p_k]\!]_{agcnf'_1} = [\![x@p_k]\!]_{agcnf_1}$ because $agcnf'_1$ and $agcnf_1$ have the same memory states for the k-th process. Assume $k = i$. We have $[\![(\mathcal{V}_k)^{p_k,AS'_{k1}}_{Ctx}(x)]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$, because $x \in lvars(\mathcal{V}_k, \mathcal{G})$. Hence, we have $x \neq upd\text{-}next(AS_{i1})$. Hence, we have $[\![x@p_k]\!]_{agcnf'_1} = [\![x@p_k]\!]_{agcnf_1}$. Hence, we have $\forall x \in lvars(\mathcal{V}_k, \mathcal{G}) : [\![x@p_k]\!]_{agcnf'_1} = [\![x@p_k]\!]_{agcnf_1}$ for all k.

Pick an arbitrary $j \neq i$, we have

$$\forall x \in Var : [\![(\mathcal{V}_j)^{p_j,AS'_{j1}}_{Ctx_j}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_j)^{p_j,AS_{j1}}_{Ctx_j}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \tag{30}$$

where $AS'_{j1} = astmt\text{-}of(agcnf'_1[j])$, $AS_{j1} = astmt\text{-}of(agcnf_1[j])$, by Lemma 20. We have $AS'_{j1} = AS_{j1}$. By $\forall k : agcnf_1 \overset{\mathcal{V}_k,Ctx_k,\mathcal{G}}{\underset{\Phi_k,\Lambda_k,k}{=}} agcnf_2$, we have

$$\forall x \in Var : [\![(\mathcal{V}_j)^{p_j,AS_{j1}}_{Ctx_j}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_j)^{p_j,AS_{j2}}_{Ctx_j}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset \tag{31}$$

where $AS_{j2} = astmt\text{-}of(agcnf_2[j])$. By (30) and (31), we have

$$\forall x \in Var : [\![(\mathcal{V}_j)^{p_j,AS'_{j1}}_{Ctx_j}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_j)^{p_j,AS_{j2}}_{Ctx_j}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset$$

Pick an arbitrary $x \in \Lambda_j(afst(AS'_{j1})) \cap \Lambda_j(afst(AS_{j2})) \cup lvars(\mathcal{V}_j, \mathcal{G})$ and assume $[\![(\mathcal{V}_j)^{p_j,AS'_{j1}}_{Ctx_j}(x)]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. We have $x \in \Lambda_j(afst(AS_{j1})) \cap \Lambda_j(afst(AS_{j2})) \cup lvars(\mathcal{V}_j, \mathcal{G})$ because $AS'_{j1} = AS_{j1}$. We deduce $[\![(\mathcal{V}_j)^{p_j,AS_{j1}}_{Ctx_j}(x)]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$ using (30). Hence, we have $m_{j1}(x) = m_{j2}(x)$, where m_{j1} and m_{j2} are the memory states of the j-th processes in $agcnf_1$ and $agcnf_2$, respectively. Let m'_{j1} and m'_{j2} be the memory states of the j-th process in $agcnf'_1$ and $agcnf_2$, respectively. We have $m'_{j1}(x) = m'_{j2}(x)$ because $m'_{j1} = m_{j1}$, and $m'_{j2} = m_{j2}$.

This completes the proof of this lemma. □

Definition 22. *The predicate no-upd-next is defined as*

$$\textit{no-upd-next}(ACS, i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{X}) \triangleq$$
$$\forall j \neq i : \forall x, x' : \left(\begin{pmatrix} x \in \Lambda_j(\textit{afst}(ACS[j])) \vee \\ x \in X_j \wedge \textit{frs}(\mathcal{V}_j(x)) \neq \textit{Pr} \end{pmatrix} \wedge x'@p_i \in \textit{fvs}(\mathcal{V}_j(x)) \right) \Rightarrow x' \notin \textit{upd-next}(ACS[i])$$

Lemma 25. *If* $\textit{nip}(\mathcal{C}, \vec{\mathcal{V}})$, $\forall j \neq i : \textit{agcnf}'_1[j] = \textit{agcnf}_1[j]$, $\forall j \neq i : \textit{agcnf}'_2[j] = \textit{agcnf}_2[j]$, $\forall j : \textit{astmt-of}(\textit{agcnf}_1[j]) \in \textit{AStmt}_{\mathrm{WF}}$, $\forall j : \textit{astmt-of}(\textit{agcnf}_2[j]) \in \textit{AStmt}_{\mathrm{WF}}$,

$$\textit{no-upd-next}(\textit{acstmt-of}(\textit{agcnf}_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}),$$
$$\textit{no-upd-next}(\textit{acstmt-of}(\textit{agcnf}_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}),$$

$\forall k : \textit{agcnf}_1 \overset{\mathcal{V}_k, \mathit{Ctx}_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} \textit{agcnf}_2$, $\forall y \in \textit{lvars}(\mathcal{V}_i, \mathcal{G}) : [\![y@p_i]\!]_{\textit{agcnf}'_1} = [\![y@p_i]\!]_{\textit{agcnf}'_2}$, *then it holds that* $\forall j \neq i : \textit{agcnf}'_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} \textit{agcnf}'_2$.

Proof. Pick an arbitrary $j \neq i$, and assume $\textit{agcnf}_1[j] = \langle p_j, \mathit{AS}_{j1}, m_{j1} \rangle$ and $\textit{agcnf}_2[j] = \langle p_j, \mathit{AS}_{j2}, m_{j2} \rangle$ for some p_j, AS_{j1}, AS_{j2}, m_{j1} and m_{j2}.

We make a case analysis on how $\textit{agcnf}_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} \textit{agcnf}_2$ is established.

– Suppose $\textit{agcnf}_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} \textit{agcnf}_2$ is established by the first rule for the relation. If $\mathcal{G} = \emptyset$, the proof is straightforward. Hence, we assume $\mathcal{G} \neq \emptyset$ below. We have for some l'_1, l'_2, X'_1, and X'_2, that

$$\textit{agcnf}_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, j}{=}} \textit{agcnf}_2 \tag{32}$$
$$\textit{high}^{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(\textit{agcnf}_1, j, l'_1, X'_1, \iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_j \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in \textit{atoms}(\mathit{AS}_{j1}) \tag{33}$$
$$\textit{high}^{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(\textit{agcnf}_2, j, l'_2, X'_2, \iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_j \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in \textit{atoms}(\mathit{AS}_{j2}) \tag{34}$$

We have $(\mathcal{V}_j)^{p_j, \mathit{AS}_{j1}}_{\mathit{Ctx}_j} = \mathcal{V}_j$ from (33), $\mathit{AS}_{j1} \in \textit{AStmt}_{\mathrm{WF}}$, $\mathcal{G} \neq \emptyset$, and Lemma 21.
We have $(\mathcal{V}_j)^{p_j, \mathit{AS}_{j2}}_{\mathit{Ctx}_j} = \mathcal{V}_j$ from (34), $\mathit{AS}_{j2} \in \textit{AStmt}_{\mathrm{WF}}$, $\mathcal{G} \neq \emptyset$, and Lemma 21.
Hence, we have $(\mathcal{V}_j)^{p_j, \mathit{AS}_{j1}}_{\mathit{Ctx}_j} = (\mathcal{V}_j)^{p_j, \mathit{AS}_{j2}}_{\mathit{Ctx}_j}$.
We next show $\textit{agcnf}'_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, j}{=}} \textit{agcnf}'_2$.
Pick an arbitrary $x \in \mathit{Var}$. We have for all $j \neq i$, and $y \in \textit{lvars}(\mathcal{V}_j, \mathcal{G})$, that $[\![y@p_j]\!]_{\textit{agcnf}'_1} = [\![y@p_j]\!]_{\textit{agcnf}'_2}$, because $\textit{agcnf}_1 \overset{\mathcal{V}_j, \mathit{Ctx}_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} \textit{agcnf}_2$, $\textit{agcnf}'_1[j] = \textit{agcnf}_1[j]$, and $\textit{agcnf}'_2[j] = \textit{agcnf}_2[j]$. Hence, for all k, and $y \in \textit{lvars}(\mathcal{V}_k, \mathcal{G})$, we have $[\![y@p_k]\!]_{\textit{agcnf}'_1} = [\![y@p_k]\!]_{\textit{agcnf}'_2}$. Hence, we have $[\![(\mathcal{V}_j)^{p_j, \mathit{AS}_{j1}}_{\mathit{Ctx}_j}(x)]\!]_{\textit{agcnf}'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_j)^{p_j, \mathit{AS}_{j2}}_{\mathit{Ctx}_j}(x)]\!]_{\textit{agcnf}'_2} \cap \mathcal{G} = \emptyset$ by Lemma 20, $\textit{nip}(\mathcal{C}, \vec{\mathcal{V}})$, and $(\mathcal{V}_j)^{p_j, \mathit{AS}_{j1}}_{\mathit{Ctx}_j} = (\mathcal{V}_j)^{p_j, \mathit{AS}_{j2}}_{\mathit{Ctx}_j}$. Next assume $x \in \Lambda_j(\textit{afst}(\mathit{AS}_{j1})) \cap \Lambda_j(\textit{afst}(\mathit{AS}_{j2})) \cup$

$lvars(\mathcal{V}_j, \mathcal{G})$, and it holds that $[\![(\mathcal{V}_j)^{p_j, AS_{j1}}_{Ctx_j}(x)]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. We then have $[\![(\mathcal{V}_j)^{p_j, AS_{j1}}_{Ctx_j}(x)]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$ using $no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$ and x is either in $\Lambda_j(afst(AS_{j1}))$ or in $lvars(\mathcal{V}_j, \mathcal{G})$. Hence, we have $m_{j1}(x) = m_{j2}(x)$ using (32). It has thus been established that $agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, j}{=}} agcnf'_2$.

We have

$$high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf'_1, j, l'_1, X'_1, \iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_j \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS_{j1})$$
$$high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf'_2, j, l'_2, X'_2, \iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_j \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS_{j2})$$

using (33), (34), and

$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$
$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$

Hence, we can establish $agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$.

– Suppose $agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$ is established by the second rule.

We have $AS_{j1} = AS_{j2}$. Hence, $(\mathcal{V}_j)^{p_j, AS_{j1}}_{Ctx_j} = (\mathcal{V}_j)^{p_j, AS_{j2}}_{Ctx_j}$. By reasoning analogous to that of the previous case, it can be shown that $agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, j}{=}} agcnf'_2$. We have $\neg high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf'_1, j)$ and $\neg high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf'_2, j)$, because of $\neg high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf_1, j)$, $\neg high^{\mathcal{V}_j, Ctx_j, \mathcal{G}}_{\Phi_j, \Lambda_j, \mathcal{X}_j}(agcnf_2, j)$, and

$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$
$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$

Hence, we can establish $agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$.

The case analysis above completes the proof of the lemma. □

Definition 23. *We define* $asst\text{-}now(\Phi, agcnf, i)$ *to express* $[\![\Phi(afst(AS))]\!]_{agcnf} = tt$*, where* $AS = astmt\text{-}of(agcnf[i])$*.*

Definition 24. *We define* $lv\text{-}now(\Lambda, AS, \iota')$ *to express if*

$$\langle p, AS, m_1\rangle \xrightarrow{\alpha} \langle p, AS', m'_1\rangle$$

and $m_1 \overset{\Lambda(afst(AS)) \cup in\text{-}vars(strip\text{-}stmt(AS))}{=} m_2$*, then there exists some* m'_2 *such that*

$$\langle p, AS, m_2\rangle \xrightarrow{\alpha} \langle p, AS', m'_2\rangle$$

and $m'_1 \overset{\Lambda(afst(AS'[\iota'/\iota_{\mathrm{f}}]))}{=} m'_2$.

Let RCI be the set $\{c!\vec{v}, c?\cdot \mid c \in PCh \wedge \vec{v} \in Val\}$ of *relaxed communication intents*. The difference with the set CI of communication intents is in that both external and internal polyadic channels can be used in the relaxed communication intents. The intention is to enable compositional proof for the case of synchronous communication, where the output and the input are regarded as separate communication actions with the (imaginary) environment.

Definition 25. *We define the predicate rmatch by*

$$\begin{aligned} rmatch(\alpha, rci) \triangleq\ & \exists c \in PCh : \exists \vec{v} \in Val^* : \alpha = c!\vec{v} \wedge rci = c?\cdot \vee \\ & \exists c \in PCh : \exists \vec{v} \in Val^* : \alpha = c?\vec{v} \wedge rci = c!\vec{v} \vee \\ & \alpha \notin \{c!\vec{v}, c?\vec{v} \mid c \in PCh \wedge \vec{v} \in Val^*\} \end{aligned}$$

Definition 26. *We define the following notion of a* modified step of a process

$$\begin{aligned} & alcnf \xrightarrow[rci]{\alpha} alcnf' : \imath' \\ \triangleq\ & \left(\begin{array}{l} \left(\begin{array}{l} \exists alcnf'' : alcnf \xrightarrow{\alpha} alcnf'' \wedge rmatch(\alpha, rci) \wedge \\ \qquad \left(\begin{array}{l} afst(astmt\text{-}of(alcnf'')) \neq \iota_{\mathrm{f}} \wedge alcnf' = alcnf'' \vee \\ afst(astmt\text{-}of(alcnf'')) = \iota_{\mathrm{f}} \wedge alcnf' = alcnf''[\imath'/\iota_{\mathrm{f}}] \end{array} \right) \end{array} \right) \\ \vee\ (\neg(\exists \alpha' : alcnf \xrightarrow{\alpha'} \wedge rmatch(\alpha', rci))) \wedge alcnf' = alcnf \wedge \alpha = \diamond \end{array} \right) \end{aligned}$$

The lemma below gives the conditions under which a modified step of a process that is not syntactically high from an augmented global configuration can be simulated by a step of a process executing the same augmented statement from a different augmented global configuration, while preserving the low-equivalence of the two processes.

Lemma 26. *If all of the following statements hold*

1. $nip(\mathcal{C}, \vec{\mathcal{V}})$,
2. $\lfloor rci_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor rci_2 \rfloor^{\mathcal{G}}_{agcnf_2}$,
3. *for each* j, $astmt\text{-}of(agcnf_1[j]) \in AStmt_{\mathrm{WF}}$,
4. *for each* j, $astmt\text{-}of(agcnf_2[j]) \in AStmt_{\mathrm{WF}}$,
5. *for each* j, $fvs\text{-}cond(astmt\text{-}of(agcnf_1[j])) \subseteq \mathcal{X}_j$,
6. *for each* j, $fvs\text{-}cond(astmt\text{-}of(agcnf_2[j])) \subseteq \mathcal{X}_j$,
7. $(\alpha_1 \neq \diamond \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}))$,
8. $(\exists \alpha_2 \neq \diamond : agcnf_2[i] \xrightarrow[rci_2]{\alpha_2}) \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$,
9. $prin\text{-}of(agcnf_1[i]) = p_i = prin\text{-}of(agcnf_2[i])$,
10. $astmt\text{-}of(agcnf_1[i]) = AS = astmt\text{-}of(agcnf_2[i])$,
11. $(\alpha_1 \neq \diamond \Rightarrow asst\text{-}now(\Phi_i, agcnf_1, i))$,
12. $(\exists \alpha_2 \neq \diamond : agcnf_2[i] \xrightarrow[rci_2]{\alpha_2}) \Rightarrow asst\text{-}now(\Phi_i, agcnf_2, i)$,
13. $lv\text{-}now(\Lambda_i, AS, \imath')$,
14. $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X)\ AS\ (l', X') : \imath'$, *where* $X' \subseteq \mathcal{X}_i$,
15. $\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l', X', \imath')$ *and* $\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_2, i, l', X', \imath')$,
16. $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$, $\forall j \neq i : agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$,
17. $agcnf_1[i] \xrightarrow[rci_1]{\alpha_1} alcnf'_1 : \imath'$, $agcnf'_1 = agcnf_1[i \mapsto alcnf'_1]$,

then there exist some α_2, $alcnf'_2$, *and* $agcnf'_2 = agcnf_2[i \mapsto alcnf'_2]$, *such that* $agcnf_2[i] \xrightarrow[rci_2]{\alpha_2} alcnf'_2 : \imath'$, $\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf_2}$, $\alpha_1 = \diamond \Leftrightarrow \alpha_2 = \diamond$, $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$, $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$, *and there exist*

some AS', l'', *such that* $\textit{astmt-of}(\textit{alcnf}'_1) = AS' = \textit{astmt-of}(\textit{alcnf}'_2)$, *and either* $\imath'\mathsf{stop} \in \textit{atoms}(AS')$ *or it can be derived that*

$$\mathcal{C}, \mathcal{V}_i, \mathit{Ctx}_i \vdash_{p_i}^{\Phi_i,\Lambda_i} (l'', X)\ AS'\ (l', X') : \imath'$$

Proof. The proof is by induction on the derivation of

$$\mathcal{C}, \mathcal{V}_i, \mathit{Ctx}_i \vdash_{p_i}^{\Phi_i,\Lambda_i} (l, X)\ AS\ (l', X') : \imath'$$

Only selected cases of this induction are shown.

– Case $\imath\mathsf{send}(c, \vec{e})$. Using $\neg \mathit{high}_{\Phi_i,\Lambda_i,\mathcal{X}_i}^{\mathcal{V}_i,\mathit{Ctx}_i,\mathcal{G}}(\mathit{agcnf}_1, i, l', X', \imath')$, we can obtain $\mathcal{C}(c) \cap \mathcal{G} \neq \emptyset$, by Lemma 19. Hence, we have $\alpha_1 = \diamond = \alpha_2$ or $\alpha_1 = c!\vec{v}$ and $\alpha_2 = c!\vec{v}'$ for some $\vec{v}$ and $\vec{v}'$, using $\lfloor \mathit{rci}_1 \rfloor_{\mathit{agcnf}_1}^{\mathcal{G}} = \lfloor \mathit{rci}_2 \rfloor_{\mathit{agcnf}_2}^{\mathcal{G}}$. In the first case, it is straightforward to establish the conclusion of the lemma. In the following, we assume $\alpha_1 = c!\vec{v}$ and $\alpha_2 = c!\vec{v}'$ for some $\vec{v}$ and $\vec{v}'$. We have $\vec{v} = [\![\vec{e}]\!]_{m_1}$ and $\vec{v}' = [\![\vec{e}]\!]_{m_2}$, where m_1 and m_2 are the memory states of $\mathit{agcnf}_1[i]$ and $\mathit{agcnf}_2[i]$, respectively.
We next show $\lfloor c!\vec{v} \rfloor_{\mathit{agcnf}_1}^{\mathcal{G}} = \lfloor c!\vec{v}' \rfloor_{\mathit{agcnf}_2}^{\mathcal{G}}$. As a first step, we show

$$[\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_1,[(c.k \mapsto v_k)_k]} \cap \mathcal{G} \neq \emptyset \Leftrightarrow [\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_2,[(c.k \mapsto v'_k)_k]} \cap \mathcal{G} \neq \emptyset$$

We consider the case where $\mathcal{G} \neq \emptyset$, since the equivalence above obviously holds in the other case. Assume per absurdum that the equivalence above does not hold. Let $\mathit{inf}(\mathcal{C}(c.j)) = P = \bigwedge_s (\phi_s \triangleright R_s)$. Then, there is an index set $I \neq \emptyset$, such that $(\bigcap_{s \in I} R_s) \cap \mathcal{G} = \emptyset$. Hence, $\mathcal{G} \subseteq \mathit{Pr} \setminus (\bigcap_{s \in I} R_s) \subseteq \mathit{Pr} \setminus (\bigcap_s R_s)$. Hence, using $\mathit{nip}(\mathcal{C}, \vec{\mathcal{V}})$, we can obtain that each principal in $\mathcal{G}$ is in $\mathit{frs}(\mathcal{V}_k(x'))$ for each $x'@p_k$ in each ϕ_s and in $\mathit{frs}(\mathcal{C}(c.k))$ for each $c.k$ in each ϕ_s.
Pick an arbitrary $x'@p_k$ in ϕ_s, we have $x' \in \mathit{lvars}(\mathcal{V}_k, \mathcal{G})$. Hence, we have $[\![x'@p_k]\!]_{\mathit{agcnf}_1} = [\![x'@p_k]\!]_{\mathit{agcnf}_2}$ by $\mathit{agcnf}_1 \overset{\mathcal{V}_i,\mathit{Ctx}_i,\mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} \mathit{agcnf}_2$.
Pick an arbitrary $c.k$ in ϕ_s. We have

$$\Phi_i(\imath) \Rightarrow \mathcal{V}_i[\mathit{fvs}(e_k) \cup X] \preceq \mathcal{C}(c.k)[(e_\ell / c.\ell)_\ell]$$

from the typing of $\imath\mathsf{send}(c, \vec{e})$. Hence, for each variable y in e_k, we have

$$[\![\mathcal{V}_i(y)]\!]_{\mathit{agcnf}_1,[(c.\ell \mapsto [\![e_\ell]\!]_{\mathit{agcnf}_1})_\ell]} \supseteq [\![\mathcal{C}(c.k)]\!]_{\mathit{agcnf}_1,[(c.\ell \mapsto [\![e_\ell]\!]_{\mathit{agcnf}_1})_\ell]}$$

We have $[\![\mathcal{V}_i(y)]\!]_{\mathit{agcnf}_1} \cap \mathcal{G} \neq \emptyset$ because $\mathcal{G} \subseteq \mathit{frs}(\mathcal{C}(c.k))$, and $\mathcal{V}_i(y)$ does not contain channel components in its conditions. Hence, if $y \in \Lambda_i(\imath)$, then we have $[\![y@p_i]\!]_{\mathit{agcnf}_1} = [\![y@p_i]\!]_{\mathit{agcnf}_2}$ by $\mathit{agcnf}_1 \overset{\mathcal{V}_i,\mathit{Ctx}_i,\mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} \mathit{agcnf}_2$. Hence, we have $[\![e_k]\!]_{m_1} = [\![e_k]\!]_{m_2}$. Hence, $v_k = v'_k$. Hence, $[\![c.k]\!]_{\mathit{agcnf}_1,[(c.k \mapsto v_k)_k]} = [\![c.k]\!]_{\mathit{agcnf}_2,[(c.k \mapsto v'_k)_k]}$.
We therefore have $[\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_1,[(c.k \mapsto v_k)_k]} = [\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_2,[(c.k \mapsto v_k)_k]}$, contradicting the assumption. We have thus established that

$$[\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_1,[(c.k \mapsto v_k)_k]} \cap \mathcal{G} \neq \emptyset \Leftrightarrow [\![\mathcal{C}(c.j)]\!]_{\mathit{agcnf}_2,[(c.k \mapsto v'_k)_k]} \cap \mathcal{G} \neq \emptyset$$

We have $\mathit{alcnf}'_1 = \langle p_i, \imath'\mathsf{stop}, m_1 \rangle$, and there exists $\mathit{alcnf}'_2 = \langle p_i, \imath'\mathsf{stop}, m_2 \rangle$, such that $\mathit{agcnf}_2[i] \xrightarrow{\alpha_2}_{\mathit{rci}_2} \mathit{alcnf}'_2 : \imath'$.

Again using the typing of $^{\iota}\mathsf{send}(c,\vec{e})$, it can be shown that $v_j = v'_j$ holds if $[\![\mathcal{C}(c.j)]\!]_{alcnf_1,[(c.\ell\mapsto v_\ell)_\ell]} \cap \mathcal{G} \neq \emptyset$. This completes the proof that $\lfloor c!\vec{v}\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor c!\vec{v}'\rfloor^{\mathcal{G}}_{agcnf_2}$.

Next, $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf'_2$ can be shown straightforwardly, noting that

$$(\mathcal{V}_i)^{p_i,\ ^{\iota'}\mathsf{stop}}_{Ctx_i} = (\mathcal{V}_i)^{p_i,\ ^{\iota}\mathsf{send}(c,\vec{e})}_{Ctx_i}$$

We have $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf'_2$ using $agcnf'_1 = agcnf_1[i \mapsto \langle p_i,\ ^{\iota'}\mathsf{stop}, m_1\rangle]$, $agcnf'_2 = agcnf_2[i \mapsto \langle p_i,\ ^{\iota}\{e\}AS_1, m_2\rangle]$, $agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf_2$ and Lemma 23.

There exist $AS' = {}^{\iota'}\mathsf{stop}$ such that $astmt\text{-}of(alcnf'_1) = AS' = astmt\text{-}of(alcnf'_2)$.

– Case $^{\iota}\mathsf{recv}(c,\vec{x})$. From $\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_1, i, l', X', \iota')$, we can obtain $\mathcal{C}(c) \cap \mathcal{G} \neq \emptyset$ using Lemma 19. We have $\alpha_1 = \diamond$ or $\alpha_1 = c?\vec{v}$ for some $\vec{v}$ with $rci_1 = c!\vec{v}$. In the first case, $\alpha_2 = \diamond$ (because $\lfloor rci_1\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor rci_2\rfloor^{\mathcal{G}}_{agcnf_2}$) and it is straightforward to establish the conclusion of the lemma. Hence, we assume $\alpha_1 = c?\vec{v}$ for some $\vec{v}$ with $rci_1 = c!\vec{v}$ below.

Then, $alcnf'_1 = \langle p_i,\ ^{\iota'}\mathsf{stop}, m'_1\rangle$, where $m'_1 = m_1[\vec{x} \mapsto \vec{v}]$. There exist $\alpha_2 = c?\vec{v}'$ with $rci_2 = c!\vec{v}'$, $alcnf'_2 = \langle p_i,\ ^{\iota'}\mathsf{stop}, m'_2\rangle$, where $m'_2 = m_2[\vec{x} \mapsto \vec{v}']$, by $\lfloor c_1!\vec{v}\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor c!\vec{v}'\rfloor^{\mathcal{G}}_{agcnf_2}$. Again from $\lfloor c_1!\vec{v}\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor c!\vec{v}'\rfloor^{\mathcal{G}}_{agcnf_2}$, we can obtain $\lfloor \alpha_1\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2\rfloor^{\mathcal{G}}_{agcnf_2}$.

We proceed to show $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf'_2$. We first show

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i,\ ^{\iota'}\mathsf{stop}}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i,\ ^{\iota'}\mathsf{stop}}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset$$

Pick an arbitrary k and $y \in lvars(\mathcal{V}_k, \mathcal{G})$. We show

$$[\![y@p_k]\!]_{agcnf'_1} = [\![y@p_k]\!]_{agcnf'_2}$$

by a case distinction on y and k.

- Suppose $y@p_k \notin \{x_j@p_i \mid j \in \{1,\ldots,|\vec{x}|\}\}$. We have $[\![y@p_k]\!]_{agcnf_1} = [\![y@p_k]\!]_{agcnf_2}$ because of $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k,\Lambda_k,k}{=}} agcnf_2$, and $y \in lvars(\mathcal{V}_k, \mathcal{G})$. By the semantics, we also have $[\![y@p_k]\!]_{agcnf_1} = [\![y@p_k]\!]_{agcnf'_1}$, and $[\![y@p_k]\!]_{agcnf_2} = [\![y@p_k]\!]_{agcnf'_2}$. Hence, we have $[\![y@p_k]\!]_{agcnf'_1} = [\![y@p_k]\!]_{agcnf'_2}$.
- Suppose $y@p_k \in \{x_j@p_i \mid j \in \{1,\ldots,|\vec{x}|\}\}$. We have $y = x_j$ for some $j \in \{1,\ldots,|\vec{x}|\}$, and $p_k = p_i$. From the typing of $^{\iota}\mathsf{recv}(c,\vec{x})$, we have

  $$\Phi(\iota) \triangleright \mathcal{C}(c.j) \preceq \mathcal{V}_i(x_j)[(c.k/(x_k@p_k))_k]$$

  Hence, we have

  $$[\![\mathcal{C}(c.j)]\!]_{agcnf_1,[(c.k\mapsto v_k)_k]} \supseteq [\![\mathcal{V}_i(x_j)]\!]_{agcnf_1[(x_k\mapsto v_k)_k],[(c.k\mapsto v_k)_k]}$$

  because of $(\alpha_1 \neq \diamond \Rightarrow asst\text{-}now(\Phi_i, agcnf_1, i))$ and $\alpha_1 \neq \diamond$. Hence, we can obtain $v_j = v'_j$ for each $j \in \{1,\ldots,|\vec{v}|\}$. Hence, we have $[\![y@p_k]\!]_{agcnf'_1} = [\![y@p_k]\!]_{agcnf'_2}$ because of the semantics, $y = x_j$ and $p_k = p_i$.

We can now establish

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i,\ ^{\iota'}\mathsf{stop}}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i,\ ^{\iota'}\mathsf{stop}}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset$$

using Lemma 20.

Pick an arbitrary $x' \in \Lambda_i(\imath') \cup lvars(\mathcal{V}_i, \mathcal{G})$ and assume $[\![\mathcal{V}_i(x')^{p,\imath'\mathsf{stop}}_{Ctx_i}]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. If $x' \in lvars(\mathcal{V}_i, \mathcal{G})$, then we have already shown $[\![x'@p_i]\!]_{agcnf'_1} = [\![x'@p_i]\!]_{agcnf'_2}$. That is, $m'_1(x') = m'_2(x')$. We next assume $x' \in \Lambda_i(\imath')$ and $[\![\mathcal{V}_i(x')^{p,\imath'\mathsf{stop}}_{Ctx_i}]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. We make a case analysis on whether $x' \in \{\vec{x}\}$.

- Suppose $x' \in \{\vec{x}\}$. We have $\Phi(\imath) \triangleright \mathcal{C}(c.j) \preceq \mathcal{V}_i(x_j)[(c.k/(x_k@p_k))_k]$ by the typing of $^\imath\mathsf{recv}(c, \vec{x})$. By reasoning analogous to the case where $x' \in lvars(\mathcal{V}_i, \mathcal{G})$, it can be established that $m'_1(x') = m'_2(x')$.
- Suppose $x' \notin \{\vec{x}\}$. We can obtain $x' \in \Lambda_i(\imath)$, using $x' \in \Lambda_i(\imath')$ and $lv\text{-}now(\Lambda_i, {}^\imath\mathsf{recv}(c, \vec{x}), \imath')$. Hence, we have $m_1(x') = m_2(x')$ by
$$agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$$
Hence, we have $m'_1(x') = m'_2(x')$ because of $m'_1(x') = m_1(x')$ and $m'_2(x') = m_2(x')$.

We have now established $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$.

Using Lemma 25, we can obtain $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$. In addition, there exists $AS' = {}^{\imath'}\mathsf{stop}$ such that $astmt\text{-}of(alcnf'_1) = AS' = astmt\text{-}of(alcnf'_2)$ and ${}^{\imath'}\mathsf{stop} \in atoms(AS')$.

– Case ${}^\imath\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$. In the proof for this case, we use $\underline{\mathsf{if}}$ as a shorthand for ${}^\imath\mathsf{if}\ e\ \mathsf{then}\ AS_1\ \mathsf{else}\ AS_2\ \mathsf{fi}$. Assume $agcnf_1[i] = \langle p_i, AS, m_1\rangle$ and $agcnf_2[i] = \langle p_i, AS, m_2\rangle$ for some m_1 and m_2.
Assume $[\![e]\!]_{m_1} = tt$ (the case where $[\![e]\!]_{m_1} = ff$ is analogous).
We have $\alpha_1 = \tau$ and $alcnf'_1 = \langle p_i, AS_1, m_1\rangle$.
From $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X)\ \underline{\mathsf{if}}\ (l', X') : \imath'$ and using the typing rule for $\underline{\mathsf{if}}$, we obtain $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X \cup fvs(e))\ \underline{\mathsf{if}}\ (l', X') : \imath'$. From
$$\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l', X', \imath')$$
we have $\forall x \in fvs(e) : [\![(\mathcal{V}_i)^{p_i, \underline{\mathsf{if}}}_{Ctx_i}]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$. Hence, for each $y \in \Lambda_i(\imath) \cap fvs(e)$, we have $m_1(y) = m_2(y)$, because of $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$. Hence, we can establish $[\![e]\!]_{m_1} = [\![e]\!]_{m_2}$, using $lv\text{-}now(\Lambda_i, \underline{\mathsf{if}}, \imath')$. Hence, we have $[\![e]\!]_{m_2} = tt$. Hence, there exist $\alpha_2 = \tau$, $alcnf'_2 = \langle p_i, AS_1, m_2\rangle$, such that $agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2} alcnf'_2 : \imath'$ can be derived.
Hence, we have $\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf_2}$.
We next show $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$. Pick an arbitrary $x \in Var$.

From $\underline{\mathsf{if}} \in AStmt_{\mathrm{WF}}$, we can obtain $AS_1 \in Stmt$, $AS_2 \in Stmt$, ${}^{\imath'}\mathsf{stop} \notin atoms(AS_1)$, and ${}^{\imath'}\mathsf{stop} \notin atoms(AS_2)$, for any $\imath'$. Hence, $(\mathcal{V}_i)^{p_i, {}^\imath\{e\}AS_1}_{Ctx_i}(x) = ((\mathcal{V}_i)^{p_i, e}_{Ctx_i(\imath)})^{p_i, AS_1}_{Ctx_i}(x) = (\mathcal{V}_i)^{p_i, e}_{Ctx_i(\imath)}(x) = P'$, where $P' = (inf(\mathcal{V}_i(x)))^{p_i, e}$ or $P' = \mathcal{V}_i(x)$. We make a case analysis on P'.

- Suppose $P' = \mathcal{V}_i(x)$. We have $(\mathcal{V}_i)^{p_i, {}^\imath\{e\}AS_1}_{Ctx_i}(x) = (\mathcal{V}_i)^{p_i, \underline{\mathsf{if}}}_{Ctx_i}(x)$. Hence,
$$[\![(\mathcal{V}_i)^{p_i, \underline{\mathsf{if}}}_{Ctx_i}(x)]\!]_{agcnf_1} = [\![(\mathcal{V}_i)^{p_i, {}^\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_1}$$
$$[\![(\mathcal{V}_i)^{p_i, \underline{\mathsf{if}}}_{Ctx_i}(x)]\!]_{agcnf_2} = [\![(\mathcal{V}_i)^{p_i, {}^\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_2}$$

because $agcnf'_1$ and $agcnf'_2$ have the same memory states as those of $agcnf_1$ and $agcnf_2$, respectively.

- Suppose $P' = (inf(\mathcal{V}_i(x)))^{p_i,e}$. we can deduce

$$[\![(\mathcal{V}_i)^{p_i,\underline{\mathsf{if}}}_{Ctx_i}(x)]\!]_{agcnf_1} = [\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_1}$$
$$[\![(\mathcal{V}_i)^{p_i,\underline{\mathsf{if}}}_{Ctx_i}(x)]\!]_{agcnf_2} = [\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_2}$$

using Lemma 18, and the fact that $agcnf'_1$ and $agcnf'_2$ have the same memory states as those of $agcnf_1$ and $agcnf_2$, respectively.

Hence, we have

$$[\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_2}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_2}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset$$

Assume $x \in \Lambda_i(fst(\imath\{e\}AS_1)) \cup lvars(\mathcal{V}_i, \mathcal{G})$, and $[\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. We have that m_1 and m_2 are the memory states of $agcnf'_1[i]$ and $agcnf'_2[i]$, respectively. We show $m_1(x) = m_2(x)$ by case analysis on the set membership of x.

- Suppose $x \in lvars(\mathcal{V}_i, \mathcal{G})$. We have $m_1(x) = m_2(x)$ by $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$.
- Suppose $x \in \Lambda_i(fst(\imath\{e\}AS_1))$. From $lv\text{-}now(\Lambda_i, \underline{\mathsf{if}}, \imath')$, we can obtain $\Lambda_i(fst(\imath\{e\}AS_1)) \setminus \Lambda_i(\imath) = \emptyset$. Hence, we have $x \in \Lambda_i(\imath)$. We have

$$[\![(\mathcal{V}_i)^{p_i,\underline{\mathsf{if}}}_{Ctx_i}(x)]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$$

because of $[\![(\mathcal{V}_i)^{p_i,\imath\{e\}AS_1}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. Hence, we have $m_1(x) = m_2(x)$, because of $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$.

It is now established that $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf'_2$.

We have $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf'_2$ using $agcnf'_1 = agcnf_1[i \mapsto \langle p_i, \imath\{e\}AS_1, m_1\rangle]$, $agcnf'_2 = agcnf_2[i \mapsto \langle p_i, \imath\{e\}AS_1, m_2\rangle]$, $agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf_2$, and Lemma 23.

By $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X)\ \underline{\mathsf{if}}\ (l', X') : \imath'$, we can obtain

$$\mathcal{C}, (\mathcal{V}_i)^{p_i,e}_{l',X',\imath'}, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X \cup fvs(e))\ AS_1\ (l', X') : \imath'$$

Hence, we can obtain $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X)\ \imath\{e\}AS_1\ (l', X') : \imath'$. Hence, there exists $AS' = \imath\{e\}AS_1$ such that

$$astmt\text{-}of(agcnf'_1[i]) = AS' = astmt\text{-}of(agcnf'_2[i])$$

and $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X)\ AS'\ (l', X') : \imath'$.

– Case $\imath$`while` e `do` AS_1 `od`. In the proof of this case, we use $\underline{\mathsf{wh}}$ as a shorthand for $\imath$`while` e `do` AS_1 `od`. Assume $agcnf_1[i] = \langle p_i, AS, m_1\rangle$, and $agcnf_2[i] = \langle p_i, AS, m_2\rangle$ for some m_1 and m_2. From $\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_1, i, l', X', \imath')$, we can obtain $\forall x \in fvs(e) : [\![(\mathcal{V}_i)^{p_i,\underline{\mathsf{wh}}}_{Ctx_i}]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$. Hence, for each $y \in \Lambda_i(\imath) \cap fvs(e)$, we have $m_1(y) = m_2(y)$, because of $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$. Hence, we can establish $[\![e]\!]_{m_1} = [\![e]\!]_{m_2}$ using $lv\text{-}now(\Lambda_i, \underline{\mathsf{wh}}, \imath')$.

The reasoning can be completed by a case analysis on $[\![e]\!]_{m_1}$. The case where $[\![e]\!]_{m_1} = tt$ is analogous to the reasoning for $\underline{\mathsf{if}}$, and the case where $[\![e]\!]_{m_1} = ff$ is straightforward noting $(\mathcal{V}_i)^{p_i,\iota'\,\mathsf{stop}}_{Ctx_i} = (\mathcal{V}_i)^{p_i,\underline{\mathsf{wh}}}_{Ctx_i}$.

– Case $AS_1; AS_2$. Let m_1 and m_2 be the memory states of $alcnf_1[i]$ and $alcnf_2[i]$, respectively. Let $alcnf_1'' \triangleq alcnf_1[i \mapsto \langle p_i, AS_1, m_1\rangle]$. Let $alcnf_2'' \triangleq alcnf_2[i \mapsto \langle p_i, AS_1, m_2\rangle]$. It is straightforward to obtain that for each j, $astmt\text{-}of(alcnf_1''[j]) \in AStmt_{\mathrm{WF}}$, $astmt\text{-}of(alcnf_2''[j]) \in AStmt_{\mathrm{WF}}$, $\alpha_1 \neq \diamond \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1''), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$, $(\exists \alpha_2 \neq \diamond : agcnf_2''[i] \overset{\alpha_2}{\twoheadrightarrow}_{rci_2}) \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2''), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$,

$$
\begin{aligned}
&prin\text{-}of(agcnf_1''[i]) = p_i = prin\text{-}of(agcnf_2''[i])\\
&astmt\text{-}of(agcnf_1''[i]) = AS_1 = astmt\text{-}of(agcnf_2''[i])\\
&(\alpha_1 \neq \diamond \Rightarrow asst\text{-}now(\Phi_i, agcnf_1'', i))\\
&(\exists \alpha_2 \neq \diamond : agcnf_2''[i] \overset{\alpha_2}{\twoheadrightarrow}_{rci_2}) \Rightarrow asst\text{-}now(\Phi_i, agcnf_2, i)
\end{aligned}
$$

$lv\text{-}now(\Lambda_i, AS_1, \iota')$, and $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X)\ AS\ (l'', X'') : afst(AS_2)$ for some l'' and X'', such that $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l'', X'')\ AS_2\ (l', X') : \iota'$,

$$
\begin{aligned}
&\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_1'', i, l'', X'', afst(AS_2))\\
&\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_2'', i, l'', X'', afst(AS_2))
\end{aligned}
$$

$agcnf_1'' \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2''$, $\forall j \neq i : agcnf_1'' \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf_2''$, $agcnf_1''[i] \overset{\alpha_1'}{\twoheadrightarrow}_{rci_1}$ $alcnf_{\mathrm{a}} : afst(AS_2)$, for some α_1', and $alcnf_{\mathrm{a}} = \langle p_i, AS_1', m_1'\rangle$, with some AS_1' and m_1'. Let $agcnf_{\mathrm{a}} = agcnf_1''[i \mapsto alcnf_{\mathrm{a}}]$.
By the induction hypothesis, there exists sovme α_2', $alcnf_{\mathrm{b}}$, and $agcnf_{\mathrm{b}} = agcnf_2''[i \mapsto alcnf_{\mathrm{b}}]$, such that

$$agcnf_2''[i] \overset{\alpha_2'}{\twoheadrightarrow}_{rci_2} alcnf_{\mathrm{b}} : afst(AS_2) \tag{35}$$

$$\lfloor \alpha_1' \rfloor^{\mathcal{G}}_{agcnf_1''} = \lfloor \alpha_2' \rfloor^{\mathcal{G}}_{agcnf_2''} \tag{36}$$

$$\alpha_1' = \diamond \Leftrightarrow \alpha_2' = \diamond \tag{37}$$

$$agcnf_{\mathrm{a}} \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_{\mathrm{b}} \tag{38}$$

$$\forall j \neq i : agcnf_{\mathrm{a}} \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf_{\mathrm{b}} \tag{39}$$

there exists some AS'' such that $astmt\text{-}of(alcnf_{\mathrm{a}}) = AS'' = astmt\text{-}of(alcnf_{\mathrm{b}})$, and either ${}^{afst(AS_2)}\mathsf{stop} \in atoms(AS'')$, or it can be derived that

$$\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l''', X)\ AS''\ (l'', X'') : afst(AS_2)$$

for some l'''.
We make a case analysis on whether $\alpha_1 = \diamond$ or not.

- Suppose $\alpha_1 \neq \diamond$. We have $\alpha_1' = \alpha_1$. We have $\alpha_2' \neq \diamond$ because of (37). We make a further case analysis on whether $\exists \vec{C} : AS_1' = blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop})$ holds or not.

* Suppose $\exists \vec{C} : AS'_1 = blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop})$ does not hold. The reasoning is straightforward using (35), (36), (38), and (39).
* Suppose $AS'_1 = blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop})$ holds for some $\vec{C}$. We have

$$agcnf'_1 = agcnf_1[i \mapsto \langle p_i, AS_2, m'_1\rangle]$$

We have $alcnf_{\mathrm{b}} = \langle p_i, blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop}), m'_2\rangle$, because of

$$astmt\text{-}of(alcnf_{\mathrm{a}}) = astmt\text{-}of(alcnf_{\mathrm{b}})$$

There exist $\alpha_2 = \alpha'_2$, $alcnf'_2 = \langle p_i, AS_2, m'_2\rangle$, and $agcnf'_2 = agcnf_2[i \mapsto alcnf'_2]$, such that $agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2} alcnf'_2 : \imath'$. We have $\lfloor\alpha_1\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor\alpha_2\rfloor^{\mathcal{G}}_{agcnf_2}$ (because of (36)), and $\alpha_1 = \diamond \Leftrightarrow \alpha_2 = \diamond$. We have

$$\begin{aligned}
agcnf'_1 &= agcnf_1[i \mapsto \langle p_i, AS_2, m'_1\rangle]\\
agcnf'_2 &= agcnf_2[i \mapsto \langle p_i, AS_2, m'_2\rangle]\\
agcnf_{\mathrm{a}} &= agcnf_1[i \mapsto \langle p_i, blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop}), m'_1\rangle]\\
agcnf_{\mathrm{b}} &= agcnf_2[i \mapsto \langle p_i, blks(\vec{C}, {}^{afst(AS_2)}\mathsf{stop}), m'_2\rangle]
\end{aligned}$$

We proceed to show $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$. For an arbitrary $y \in lvars(\mathcal{V}_i, \mathcal{G})$, we have $[\![y@p_i]\!]_{agcnf'_1} = [\![y@p_i]\!]_{agcnf'_2}$, because of

$$agcnf_{\mathrm{a}} \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_{\mathrm{b}}$$

For an arbitrary $y \in lvars(\mathcal{V}_j, \mathcal{G})$, where $j \neq i$, we have $[\![y@p_j]\!]_{agcnf'_1} = [\![y@p_j]\!]_{agcnf'_2}$ because of $\forall j \neq i : agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$. Hence, we can obtain

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i, AS_2}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i, AS_2}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset$$

using Lemma 20.
Pick an arbitrary $x' \in \Lambda_i(afst(AS_2)) \cup lvars(\mathcal{V}_i, \mathcal{G})$. If $x' \in lvars(\mathcal{V}_i, \mathcal{G})$, we have already shown that $m'_1(x') = m'_2(x'_2)$. We next assume that $x' \in \Lambda_i(afst(AS_2))$, and $[\![(\mathcal{V}_i)^{p_i, AS_2}_{Ctx_i}(x')]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$, and we show $m'_1(x') = m'_2(x')$. We have $agcnf''_1[i] \xrightarrow{\alpha_1} \langle p_i, blks(\vec{C}, \imath_{\mathrm{f}}\mathsf{stop}), m'_1\rangle$. Hence, we can deduce $AS_1 = blks(\vec{C}, S)$ for some $S \in Stmt$ because of $AS_1 \in AStmt_{\mathrm{WF}}$ and the semantics. Hence, we have

$$\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X)\ blks(\vec{C}, S)\ (l'', X'') : afst(AS_2)$$

because of

$$\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X)\ AS_1\ (l'', X'') : afst(AS_2)$$

Hence, for each $C_s = \imath_s\{e_s\}$, we have $Ctx_i(\imath_s) = (l'', X'', afst(AS_2))$. Hence, we can deduce $(\mathcal{V}_i)^{p_i, AS''}_{Ctx_i}(x') = (\mathcal{V}_i)^{p_i, AS_1}_{Ctx_i}(x') = \mathcal{V}_i(x') = (\mathcal{V}_i)^{p_i, AS_2}_{Ctx_i}(x')$, because of $AS'' = AS'_1$, $x' \in \Lambda(afst(AS_2))$, and $AS_2 \in Stmt$ (because $AS_1; AS_2 \in AStmt_{\mathrm{WF}}$). Hence, we have

$$[\![(\mathcal{V}_i)^{p_i, AS''}_{Ctx_i}(x')]\!]_{agcnf_{\mathrm{a}}} \cap \mathcal{G} \neq \emptyset$$

Hence, we have $m'_1(x') = m'_2(x')$ because of (38), and $x' \in \Lambda_i(afst(AS''))$. It can be deduced that

$$\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$$

using Lemma 25. There exist $AS' = AS_2$ such that $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l'', X'')\ AS'\ (l', X') : \imath'$.

- Suppose $\alpha_1 = \diamond$. It is straightforward to show that there exists $\alpha_2 = \diamond$ satisfying the desired conditions.

– Case $\imath\{e\}AS'$. Let $\mathcal{V}'_i = (\mathcal{V}_i)^{p_i, e}_{l', X', \imath'}$, $agcnf''_1 \triangleq agcnf_1[i \mapsto \langle p_i, AS', m_1\rangle]$, $agcnf''_2 \triangleq agcnf_2[i \mapsto \langle p_i, AS', m_2\rangle]$.
We have $\lfloor rci_1 \rfloor^{\mathcal{G}}_{agcnf''_1} = \lfloor rci_2 \rfloor^{\mathcal{G}}_{agcnf''_2}$, $nip(\mathcal{C}, \vec{\mathcal{V}}')$ where $\vec{\mathcal{V}}' = \mathcal{V}_1 \ldots \mathcal{V}'_i \ldots \mathcal{V}_n$,

$\forall j : astmt\text{-}of(agcnf''_1[j]) \in AStmt_{\mathrm{WF}}$
$\forall j : astmt\text{-}of(agcnf''_2[j]) \in AStmt_{\mathrm{WF}}$
$\forall j : fvs\text{-}cond(agcnf''_1[j]) \subseteq \mathcal{X}_j$
$\forall j : fvs\text{-}cond(agcnf''_2[j]) \subseteq \mathcal{X}_j$
$(\alpha_1 \neq \diamond \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf''_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}))$
$(\exists \alpha_2 \neq \diamond : agcnf''_2[i] \overset{\alpha_2}{\rightarrow}_{rci_2}) \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf''_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$
$(\alpha_1 \neq \diamond \Rightarrow asst\text{-}now(\Phi_i, agcnf''_1, i))$
$(\exists \alpha_2 \neq \diamond : agcnf''_2[i] \overset{\alpha_2}{\rightarrow}_{rci_2}) \Rightarrow asst\text{-}now(\Phi_i, agcnf''_2, i)$
$lv\text{-}now(\Lambda_i, AS', \imath')$

We also have

$$\mathcal{C}, \mathcal{V}'_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l, X \cup fvs(e))\ AS'\ (l', X') : \imath' \tag{40}$$

with $X' \subseteq \mathcal{X}_i$, $\neg high^{\mathcal{V}'_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf''_1, i, l', X', \imath')$, $\neg high^{\mathcal{V}'_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf''_2, i, l', X', \imath')$, $agcnf''_1 \overset{\mathcal{V}'_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf''_2$, $\forall j \neq i : agcnf''_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf''_2$ (using Lemma 23), and $agcnf''_1[i] \overset{\alpha_1}{\rightarrow}_{rci_1} alcnf_{\mathrm{a}} : \imath'$ for some $alcnf_{\mathrm{a}} = \langle p_i, AS_{\mathrm{a}}, m'_1\rangle$, with some AS_{a} and m'_1. Let $agcnf_{\mathrm{a}}$ be $agcnf''_1[i \mapsto alcnf_{\mathrm{a}}]$.
By the induction hypothesis, there exists some α_2, $alcnf_{\mathrm{b}} = \langle p_i, AS_{\mathrm{b}}, m'_2\rangle$ for some AS_{b} and m'_2, and $agcnf_{\mathrm{b}} = agcnf''_2[i \mapsto alcnf_{\mathrm{b}}]$, such that

$$agcnf''_2[i] \overset{\alpha_2}{\rightarrow}_{rci_2} alcnf_{\mathrm{b}} : \imath' \tag{41}$$
$$\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf''_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf''_2} \tag{42}$$
$$\alpha_1 = \diamond \Leftrightarrow \alpha_2 = \diamond \tag{43}$$
$$agcnf_{\mathrm{a}} \overset{\mathcal{V}'_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_{\mathrm{b}} \tag{44}$$
$$\forall j \neq i : agcnf_{\mathrm{a}} \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_{\mathrm{b}} \tag{45}$$

there exist some AS'', l''', such that $astmt\text{-}of(alcnf_{\mathrm{a}}) = AS'' = astmt\text{-}of(alcnf_{\mathrm{b}})$, and either $\imath' \mathsf{stop} \in atoms(AS'')$ or it can be derived that

$$\mathcal{C}, \mathcal{V}'_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (l''', X)\ AS''\ (l', X') : \imath'$$

Hence, $AS_{\mathrm{a}} = AS'' = AS_{\mathrm{b}}$. From (41), we have $agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2} \langle p_i, \imath\{e\}AS_{\mathrm{b}}, m_2'\rangle : \imath'$. Let $alcnf_2' \triangleq \langle p_i, \imath\{e\}AS_{\mathrm{b}}, m_2'\rangle$, and $agcnf_2' \triangleq agcnf_2[i \mapsto alcnf_2']$. We have $\lfloor\alpha_1\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor\alpha_2\rfloor^{\mathcal{G}}_{agcnf_2}$ because of (42). We have

$$\begin{aligned} agcnf_1' &= agcnf_1[i \mapsto \langle p_i, \imath\{e\}AS'', m_1'\rangle] \\ agcnf_2' &= agcnf_2[i \mapsto \langle p_i, \imath\{e\}AS'', m_2'\rangle] \\ agcnf_{\mathrm{a}} &= agcnf_1[i \mapsto \langle p_i, AS'', m_1'\rangle] \\ agcnf_{\mathrm{b}} &= agcnf_2[i \mapsto \langle p_i, AS'', m_2'\rangle] \end{aligned}$$

Hence, we have $agcnf_1' \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2'$ because of (44), and we have $\forall j \neq i : agcnf_1' \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j,\Lambda_j,\mathcal{X}_j,j}{\approx}} agcnf_2'$, because of (45). By (40), and $agcnf_1''[i] \xrightarrow{\alpha_1}_{rci_1} alcnf_{\mathrm{a}} : \imath'$, we have $\mathcal{C}, \mathcal{V}_i', Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X \cup fvs(e))\ AS''\ (l', X') : \imath'$. Hence, there exists $AS' = \imath\{e\}AS''$, for which it holds that $astmt\text{-}of(alcnf_1') = AS' = astmt\text{-}of(alcnf_2')$, and it can be derived that

$$\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i,\Lambda_i}_{p_i} (l, X)\ AS'\ (l', X') : \imath'$$

– Case subtyping. The reasoning for this case is straightforward, using the induction hypothesis on the typing of AS with a higher initial context and a lower final context.

This induction completes the proof of the lemma. □

The main message of the lemma below is: the low-equality of two processes is preserved by a step of the first process, if it is syntactically high.

Lemma 27. *If* $astmt\text{-}of(agcnf_1[i]) \in AStmt_{\mathrm{WF}}$, $asst\text{-}now(\Phi_i, agcnf_1, i)$, $lv\text{-}now(\Lambda_i, astmt\text{-}of(agcnf_1[i]), \iota_{\mathrm{f}})$, $nip(\mathcal{C}, \vec{\mathcal{V}})$, $high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_1, i, l_1', X_1', \iota_{\mathrm{f}})$, $X_1' \subseteq \mathcal{X}_i$, $agcnf_1[i] \xrightarrow{\alpha} alcnf_1'$, $agcnf_1' = agcnf_1[i \mapsto alcnf_1']$, $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$, *then* $agcnf_1' \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$.

Proof. Let $agcnf_1 \triangleq \langle \ldots, \langle p_i, AS_{i1}, m_{i1}\rangle, \ldots\rangle$ and $agcnf_1' \triangleq \langle \ldots, \langle p_i, AS_{i1}', m_{i1}'\rangle, \ldots\rangle$. Using Lemma 22, we obtain

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i, AS_{i1}'}_{Ctx_i}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \ \Leftrightarrow\ [\![(\mathcal{V}_i)^{p_i, AS_{i1}}_{Ctx_i}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \tag{46}$$

$$\forall x \in upd\text{-}next(AS_{i1}) : [\![(\mathcal{V}_i)^{p_i, AS_{i1}'}_{Ctx_i}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \tag{47}$$

We have $\forall x \in Var : [\![(\mathcal{V}_i)^{p_i, AS_{i1}}_{Ctx_i}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \ \Leftrightarrow\ [\![(\mathcal{V}_i)^{p_i, AS_{i2}}_{Ctx_i}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset$ because of $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i,\Lambda_i,i}{=}} agcnf_2$. Hence, we have

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i, AS_{i1}'}_{Ctx_i}(x)]\!]_{agcnf_1'} \cap \mathcal{G} = \emptyset \ \Leftrightarrow\ [\![(\mathcal{V}_i)^{p_i, AS_{i2}}_{Ctx_i}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset$$

by (46).

Pick an arbitrary $x' \in \Lambda_i(afst(AS'_{i1})) \cap \Lambda_i(afst(AS_{i2})) \cup lvars(\mathcal{V}_i, \mathcal{G})$, and assume that $[\![(\mathcal{V}_i)^{p_i, AS'_{i1}}_{Ctx_i}(x')]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. Then, we have $[\![(\mathcal{V}_i)^{p_i, AS_{i1}}_{Ctx_i}(x')]\!]_{agcnf_1} \cap \mathcal{G} \neq \emptyset$ using (46). If $x' \in \Lambda_i(afst(AS_{i1}))$, then $x' \in \Lambda_i(afst(AS_{i1})) \cap \Lambda_i(afst(AS_{i2}))$. Hence, we have $m_{i1}(x') = m_{i2}(x')$ using $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$. We have $m'_{i1}(x') = m_{i1}(x')$ because of (47) and $[\![(\mathcal{V}_i)^{p_i, AS'_{i1}}_{Ctx_i}(x')]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$. Hence, we have $m'_{i1}(x') = m_{i2}(x')$. On the other hand, if $x' \notin \Lambda_i(afst(AS_{i1}))$, then we have $x' \in upd\text{-}next(AS_{i1})$ by $lv\text{-}now(\Lambda_i, astmt\text{-}of(agcnf_1[i]), \iota_{\mathrm{f}})$ and $x' \in \Lambda_i(afst(AS'_{i1}))$. Hence, we have $[\![(\mathcal{V}_i)^{p_i, AS'_{i1}}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset$ using (47). We therefore have a contradiction with the assumption $[\![(\mathcal{V}_i)^{p_i, AS'_{i1}}_{Ctx_i}(x')]\!]_{agcnf'_1} \cap \mathcal{G} \neq \emptyset$, and the case where $x' \notin \Lambda_i(afst(AS_{i1}))$ is impossible. This completes the proof. □

The lemma below gives conditions under which a step of a syntactically high process in an augmented global configuration can be simulated by another syntactically high process in a different augmented global configuration, while preserving the low-equivalence of the two processes.

Lemma 28. *If all of the following statements hold*

1. $nip(\mathcal{C}, \vec{\mathcal{V}})$,
2. $\lfloor rci_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor rci_2 \rfloor^{\mathcal{G}}_{agcnf_2}$,
3. $\alpha_1 \neq \diamond \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$,
4. $(\exists \alpha_2 \neq \diamond : agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2}) \Rightarrow no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$,
5. $\alpha_1 \neq \diamond \Rightarrow asst\text{-}now(\Phi_i, agcnf_1, i)$
6. $(\exists \alpha_2 \neq \diamond : agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2}) \Rightarrow asst\text{-}now(\Phi_i, agcnf_2, i)$,
7. $lv\text{-}now(\Lambda_i, astmt\text{-}of(agcnf_1[i]), \iota_{\mathrm{f}})$ *and* $lv\text{-}now(\Lambda_i, astmt\text{-}of(agcnf_2[i]), \iota_{\mathrm{f}})$,
8. $astmt\text{-}of(agcnf_1[i]) \in AStmt_{\mathrm{WF}}$ *and* $astmt\text{-}of(agcnf_2[i]) \in AStmt_{\mathrm{WF}}$,
9. $high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l'_1, X'_1, \iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i \vee$ $\iota_{\mathrm{f}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_1[i]))$,
10. $high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_2, i, l'_2, X'_2, \iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i \vee$ $\iota_{\mathrm{f}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_2[i]))$,
11. $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$, $\forall j \neq i : agcnf_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$,
12. $agcnf_1[i] \xrightarrow{\alpha_1}_{rci_1} alcnf'_1 : \iota_{\mathrm{f}}$, $agcnf'_1 = agcnf_1[i \mapsto alcnf'_1]$,

then there exist some α_2, $alcnf'_2$, *and* $agcnf'_2 = agcnf_2[i \mapsto alcnf'_2]$, *such that* $agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2} alcnf'_2 : \iota_{\mathrm{f}}$, $\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf_2}$, $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$, $\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$, $high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf'_1, i, l'_1, X'_1, \iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i \vee \iota_{\mathrm{f}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf'_1[i]))$, *and* $high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf'_2, i, l'_2, X'_2, \iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i \vee \iota_{\mathrm{f}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf'_2[i]))$.

Proof. We prove this lemma with a case analysis based on the conditions 9 and 10 in the lemma statement. Only the following selected case is presented

$$high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_1,i,l'_1,X'_1,\iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i$$
$$high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_2,i,l'_2,X'_2,\iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i$$

We further assume $\alpha_1 \neq \diamond$ – the other case, where $\alpha_1 = \diamond$, is less involved. There exists some α_2 such that $agcnf_2[i] \xrightarrow{\alpha_2}_{rci_2} alcnf'_2 : \iota_{\mathrm{f}}$ for some $alcnf'_2$. We further assume $\alpha_2 \neq \diamond$ – the other case, where $\alpha_2 = \diamond$, is less involved.

We have $agcnf_1[i] \xrightarrow{\alpha_1} alcnf'_1$ and $agcnf_2[i] \xrightarrow{\alpha_2} alcnf'_2$. Suppose

$$alcnf_1 = \langle \ldots, \langle p_i, AS_{i1}, m_{i1}\rangle, \ldots\rangle$$
$$alcnf_2 = \langle \ldots, \langle p_i, AS_{i2}, m_{i2}\rangle, \ldots\rangle$$
$$alcnf'_1 = \langle \ldots, \langle p_i, AS'_{i1}, m'_{i1}\rangle, \ldots\rangle$$
$$alcnf'_2 = \langle \ldots, \langle p_i, AS'_{i2}, m'_{i2}\rangle, \ldots\rangle$$

From $high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_1,i,l'_1,X'_1,\iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i$, we obtain that there exists X_1 such that $high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_1,i,X_1,l'_1,X'_1,\iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i$. We have $[\![\Phi_i(afst(AS_{i1}))]\!]_{agcnf_1} = tt$ and $nip(\mathcal{C},\vec{\mathcal{V}})$ using the hypotheses of the lemma and $\alpha_1 \neq \diamond$. Hence, we can obtain

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i,AS'_{i1}}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i,AS_{i1}}_{Ctx_i}(x)]\!]_{agcnf_1} \cap \mathcal{G} = \emptyset \quad (48)$$
$$\forall c : (\exists \vec{v} : \alpha_1 = c!\vec{v} \vee \exists \vec{v} : \alpha_1 = c?\vec{v}) \Rightarrow \mathcal{C}(c) \cap \mathcal{G} = \emptyset \quad (49)$$
$$\forall x \in upd\text{-}next(AS_{i1}) : [\![(\mathcal{V}_i)^{p_i,AS'_{i1}}_{Ctx_i}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \quad (50)$$
$$^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS'_{i1}) \vee \exists X''_1 : X_1 \subseteq X''_1 \wedge high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_1,i,X''_1,l'_1,X'_1,\iota_{\mathrm{f}}) \quad (51)$$

using Lemma 22. From $high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_2,i,l'_2,X'_2,\iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i$, we can obtain that there exists some X_2 such that $high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf_2,i,X_2,l'_2,X'_2,\iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i$. We have $[\![\Phi_i(afst(AS_{i2}))]\!]_{agcnf_2} = tt$ and $nip(\mathcal{C},\vec{\mathcal{V}})$ using the hypotheses of the lemma and $\alpha_2 \neq \diamond$. Hence, we can obtain

$$\forall x \in Var : [\![(\mathcal{V}_i)^{p_i,AS'_{i2}}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset \Leftrightarrow [\![(\mathcal{V}_i)^{p_i,AS_{i2}}_{Ctx_i}(x)]\!]_{agcnf_2} \cap \mathcal{G} = \emptyset \quad (52)$$
$$\forall c : (\exists \vec{v} : \alpha_2 = c!\vec{v} \vee \exists \vec{v} : \alpha_2 = c?\vec{v}) \Rightarrow \mathcal{C}(c) \cap \mathcal{G} = \emptyset \quad (53)$$
$$\forall x \in upd\text{-}next(AS_{i2}) : [\![(\mathcal{V}_i)^{p_i,AS'_{i2}}_{Ctx_i}(x)]\!]_{agcnf'_2} \cap \mathcal{G} = \emptyset \quad (54)$$
$$^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS'_{i2}) \vee \exists X''_2 : X_2 \subseteq X''_2 \wedge high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_2,i,X''_2,l'_2,X'_2,\iota_{\mathrm{f}}) \quad (55)$$

We have $\lfloor\alpha_1\rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor\alpha_2\rfloor^{\mathcal{G}}_{agcnf_2}$ using (49) and (53). We have

$$high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_1,i,l'_1,X'_1,\iota_{\mathrm{f}}) \wedge X'_1 \subseteq \mathcal{X}_i \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(agcnf'_1[i])$$

using (51) and $X'_1 \subseteq \mathcal{X}_i$. We have

$$high^{\mathcal{V}_i,Ctx_i,\mathcal{G}}_{\Phi_i,\Lambda_i,\mathcal{X}_i}(agcnf'_2,i,l'_2,X'_2,\iota_{\mathrm{f}}) \wedge X'_2 \subseteq \mathcal{X}_i \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(agcnf'_2[i])$$

using (55) and $X_2' \subseteq \mathcal{X}_i$.

Using Lemma 27 on $agcnf_1[i] \xrightarrow{\alpha_1} alcnf'_1$, we obtain $agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2$. Using Lemma 27 again on $agcnf_2[i] \xrightarrow{\alpha_2} alcnf'_2$, we obtain

$$agcnf'_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf'_2$$

Pick an arbitrary $y \in lvars(\mathcal{V}_i, \mathcal{G})$. It is straightforward to show $[\![y@p_i]\!]_{agcnf'_1} = [\![y@p_i]\!]_{agcnf'_2}$. From the hypotheses of the lemma, and $\alpha_1 \neq \diamond$, we have

$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_1), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$

From the hypotheses of the lemma, and $\alpha_2 \neq \diamond$, we have

$$no\text{-}upd\text{-}next(acstmt\text{-}of(agcnf_2), i, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$

From the hypotheses of the lemma, we have $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$. Hence, we have $\forall k : agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$. Hence, we have

$$\forall j \neq i : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf'_2$$

using Lemma 25.

The proof of the lemma is complete with the case analysis based on the conditions 9 and 10 in the lemma statement (selectively presented above). □

The lemma below gives conditions under which a step of a system (executing augmented statements) can be simulated by a step of another system, while preserving the low-equivalence of the two systems and the equality of the observed execution histories of both systems.

Lemma 29. *If* $\xi_1 \overset{\mathcal{G}}{=} \xi_2$, $\forall i \in \{1, \ldots, |CS|\} : asst(\Phi_i, CS, i)$, $\forall i \in \{1, \ldots, |CS|\} : live(\Lambda_i, CS, i)$, $\forall i \in \{1, \ldots, |CS|\} : \mathcal{X}_i = fvs\text{-}cond(CS[i])$, $atr_1 \in atraces\text{-}of_{\xi_1}(CS)$, $atr_2 \in atraces\text{-}of_{\xi_2}(CS)$, $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$, $last(atr_1) \overset{\vec{\mathcal{V}}, C\vec{t}x, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\approx}} last(atr_2)$, *and* $atr_1 \rightarrow_{\xi_1} atr'_1$, *then there exists some* atr'_2 *such that* $atr_2 \rightarrow_{\xi_2} atr'_2$, $last(atr'_1) \overset{\vec{\mathcal{V}}, C\vec{t}x, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\approx}} last(atr'_2)$, *and* $\lfloor atr'_1 \rfloor^{\mathcal{G}} = \lfloor atr'_2 \rfloor^{\mathcal{G}}$.

Proof. Let $agcnf_1 \triangleq last(atr_1)$ and $agcnf_2 \triangleq last(atr_2)$. Furthermore, let CS be of the form $S_1 || \ldots || S_n$ for some $S_1, \ldots, S_n$. Suppose

$$\begin{aligned}
&agcnf_1 = \langle \ldots, \langle p_i, AS_{i1}, m_{i1} \rangle, \ldots \rangle \\
&agcnf_2 = \langle \ldots, \langle p_i, AS_{i2}, m_{i2} \rangle, \ldots \rangle \\
&\forall i : strip\text{-}stmt(AS_{i1}) = S_{i1} \\
&\forall i : strip\text{-}stmt(AS_{i2}) = S_{i2}
\end{aligned}$$

for some $AS_{11}, \ldots, AS_{n1}, AS_{12}, \ldots, AS_{n2}, S_{11}, \ldots, S_{n1}, S_{12}, \ldots, S_{n2}, p_1, \ldots, p_n$. From the assumptions of the lemma (Lemma 29), we have

$$nip(\mathcal{C}, \vec{\mathcal{V}}) \tag{56}$$
$$no\text{-}upd(S_{11}||\ldots||S_{n1}, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}) \tag{57}$$
$$no\text{-}upd(S_{12}||\ldots||S_{n2}, \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}}) \tag{58}$$

with $\mathcal{X}_i = fvs\text{-}cond(S_i)$ for each i. We have

$$\lfloor \xi_1(strip\text{-}tr(atr_1)) \rfloor^{\mathcal{G}}_{last(atr_1)} = \lfloor \xi_2(strip\text{-}tr(atr_2)) \rfloor^{\mathcal{G}}_{last(atr_2)} \tag{59}$$

by $\xi_1 \overset{\mathcal{G}}{=} \xi_2$, and $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$. We have for each k,

$$may\text{-}step(CS, fst(S_{11}) \ldots fst(S_{n1}), k) \Rightarrow [\![\Phi_k(fst(S_{k1}))]\!]_{agcnf_1} = tt \tag{60}$$
$$may\text{-}step(CS, fst(S_{12}) \ldots fst(S_{n2}), k) \Rightarrow [\![\Phi_k(fst(S_{k2}))]\!]_{agcnf_2} = tt \tag{61}$$

because of $asst(\Phi_k, CS, k)$ for each k, $atr_1 = atraces\text{-}of_{\xi_1}(CS)$, $atr_2 = atraces\text{-}of_{\xi_2}(CS)$, $last(atr_1) = \langle \ldots, \langle p_i, AS_{i1}, m_{i1} \rangle, \ldots \rangle$, and $last(atr_2) = \langle \ldots, \langle p_i, AS_{i2}, m_{i2} \rangle, \ldots \rangle$. Hence, we have for each k

$$may\text{-}step(CS, fst(S_{11}) \ldots fst(S_{n1}), k) \Rightarrow asst\text{-}now(\Phi_k, agcnf_1, k) \tag{62}$$
$$may\text{-}step(CS, fst(S_{12}) \ldots fst(S_{n2}), k) \Rightarrow asst\text{-}now(\Phi_k, agcnf_2, k) \tag{63}$$

We have for each k

$$lv\text{-}now(\Lambda_k, AS_{k1}, \iota_{\mathrm{f}}) \tag{64}$$
$$lv\text{-}now(\Lambda_k, AS_{k2}, \iota_{\mathrm{f}}) \tag{65}$$

because for each k, $live(\Lambda_k, CS, k)$ holds.

We have

$$\forall k : agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2 \tag{66}$$

from $agcnf_1 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\simeq}} agcnf_2$.

Let $ii_1 \triangleq \xi_1(strip\text{-}tr(agcnf_1))$, and $ii_2 \triangleq \xi_2(strip\text{-}tr(agcnf_2))$. The rest of the proof is by a case analysis on ii_1 and ii_2.

– Suppose $ii_1 = (p_i, ci_1)$ for some communication intent ci_1. We have $ii_2 = (p_i, ci_2)$ for some ci_2 such that $\lfloor ci_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor ci_2 \rfloor^{\mathcal{G}}_{agcnf_2}$ because it holds that $\lfloor ii_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor ii_2 \rfloor^{\mathcal{G}}_{agcnf_2}$.

  We make a further case analysis on how $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ is derived.

- Suppose $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ is derived by

$$agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2 \tag{67}$$

$$AS_{i1} = AS = AS_{i2} \tag{68}$$

$$\mathcal{C}_i, \mathcal{V}_i, Ctx_i \vdash_{p_i}^{\Phi_i, \Lambda_i} (l, X)\ AS\ (l', X') : \iota_{\mathrm{f}} \wedge X' \subseteq \mathcal{X}_i \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS) \tag{69}$$

$$\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l', X', \iota_{\mathrm{f}}) \tag{70}$$

$$\neg high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_2, i, l', X', \iota_{\mathrm{f}}) \tag{71}$$

  Suppose $atr'_1 = atr_1.\alpha_1.agcnf_1$ for some α_1. We have $agcnf_1[i] \overset{\alpha_1}{\rightarrow\!\!\triangleright}_{ci_1} alcnf'_1 : \iota_{\mathrm{f}}$ where $alcnf'_1$ is such that $agcnf'_1 = agcnf_1[i \mapsto alcnf'_1]$.
  From (69), either ${}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS)$ or

$$\mathcal{C}_i, \mathcal{V}_i, Ctx_i \vdash_{p_i}^{\Phi_i, \Lambda_i} (l, X)\ AS\ (l', X') : \iota_{\mathrm{f}} \wedge X' \subseteq \mathcal{X}_i$$

  We can derive $AS \in AStmt_{\mathrm{WF}}$ using $S_i \in AStmt_{\mathrm{WF}}$ and Lemma 15. If ${}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS)$, we then have $AS = blks(\vec{C}, {}^{\iota_{\mathrm{f}}}\mathsf{stop})$ using Lemma 14. On this basis, it is trivial to resolve the case where ${}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(AS)$. On the other hand, the case where $\mathcal{C}_i, \mathcal{V}_i, Ctx_i \vdash_{p_i}^{\Phi_i, \Lambda_i} (l, X)\ AS\ (l', X') : \iota_{\mathrm{f}} \wedge X' \subseteq \mathcal{X}_i$ can be resolved by reasoning using Lemma 26 whose pre-conditions are established using the conditions (56) to (71), and $AS \in AStmt_{\mathrm{WF}}$.

- Suppose $agcnf_1 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\simeq}} agcnf_2$ is derived by

$$agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2 \tag{72}$$

$$high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l'_1, X'_1, \iota_{\mathrm{f}}) \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_1[i])) \tag{73}$$

$$high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_2, i, l'_2, X'_2, \iota_{\mathrm{f}}) \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_2[i])) \tag{74}$$

  This case is resolved using Lemma 28, whose pre-conditions are established using the conditions (56) to (66), and (72) to (74).

– Suppose $ii_1 = (p_i, p_k)$ for some i and k ($i \neq k$). We have $ii_2 = (p_i, p_k)$ because of $\lfloor ii_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor ii_2 \rfloor^{\mathcal{G}}_{agcnf_2}$. We have

$$AS_{i1} \in AStmt_{\mathrm{WF}} \tag{75}$$

$$AS_{i2} \in AStmt_{\mathrm{WF}} \tag{76}$$

$$AS_{k1} \in AStmt_{\mathrm{WF}} \tag{77}$$

$$AS_{k2} \in AStmt_{\mathrm{WF}} \tag{78}$$

  because of $S_i \in AStmt_{\mathrm{WF}}$, $S_k \in AStmt_{\mathrm{WF}}$, $agcnf_1 = last(atr_1)$, $atr_i = last(atr_2)$, and Lemma 15.
  Assume $atr'_1 = atr_1.\alpha_1.agcnf'_1$. We make a case analysis on the rules used to establish $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ and $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$, respectively.

- Suppose both $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ and $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$ are established using the first rule for deriving the relation.
  We have

$$agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2 \tag{79}$$

$$high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_1, i, l'_{i1}, X'_{i1}, \iota_{\mathrm{f}}) \wedge X'_{i1} \subseteq \mathcal{X}_i \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_1[i])) \tag{80}$$

$$high^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf_2, i, l'_{i2}, X'_{i2}, \iota_{\mathrm{f}}) \wedge X'_{i2} \subseteq \mathcal{X}_i \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_2[i])) \tag{81}$$

  and

$$agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, k}{=}} agcnf_2 \tag{82}$$

$$high^{\mathcal{V}_k, Ctx_k, \mathcal{G}}_{\Phi_k, \Lambda_k, \mathcal{X}_k}(agcnf_1, k, l'_{k1}, X'_{k1}, \iota_{\mathrm{f}}) \wedge X'_{k1} \subseteq \mathcal{X}_k \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_1[k])) \tag{83}$$

$$high^{\mathcal{V}_k, Ctx_k, \mathcal{G}}_{\Phi_k, \Lambda_k, \mathcal{X}_k}(agcnf_2, k, l'_{k2}, X'_{k2}, \iota_{\mathrm{f}}) \wedge X'_{k2} \subseteq \mathcal{X}_k \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in atoms(astmt\text{-}of(agcnf_2[k])) \tag{84}$$

  We make a case analysis on whether $\alpha_1 = \diamond$.
  - Suppose $\alpha_1 \neq \diamond$. Assume (w.l.o.g.)

$$agcnf_1[i] \xrightarrow{c!\vec{v}} alcnf'_{i1}$$
$$agcnf_1[k] \xrightarrow{c?\vec{v}} alcnf'_{k1}$$

    for some $alcnf'_{i1}$, $alcnf'_{k1}$, c, and $\vec{v}$, such that

$$agcnf'_1 = agcnf_1[i \mapsto alcnf'_{i1}][k \mapsto alcnf'_{k1}]$$

    We have $\alpha_1 = \tau$.
    Hence, we can obtain $\mathcal{C}(c) \cap \mathcal{G} = \emptyset$ using Lemma 22.
    Let $rci_1 \triangleq c?\cdot$. We have

$$agcnf_1[i] \overset{c!\vec{v}}{\rightarrowtriangle}_{rci_1} alcnf'_{i1} : \iota_{\mathrm{f}} \tag{85}$$

    We make a further case analysis on whether $\alpha_2 = \diamond$.
    - Suppose $\alpha_2 \neq \diamond$. Assume (w.l.o.g.)

$$agcnf_2[i] \xrightarrow{c'!\vec{v}'} alcnf'_{i2}$$
$$agcnf_2[k] \xrightarrow{c'?\vec{v}'} alcnf'_{k2}$$

      We have $\alpha_2 = \tau$. This case can be resolved by reasoning using Lemma 28 twice, the first time with $c?\cdot$ and $c'?\cdot$ as the relaxed communication intents, and the second time with $c!\vec{v}$ and $c'!\vec{v}$ as the relaxed communication intents.

· Suppose $\alpha_2 = \diamond$. Let $agcnf''_1 = agcnf_1[i \mapsto agcnf'_{i1}]$. Using (56), (80), (75), and (60), all the pre-conditions of Lemma 22 can be established. Using this lemma, we can obtain $\mathcal{C}(c) \cap \mathcal{G} = \emptyset$, and

$$\forall x \in \mathit{upd\text{-}next}(AS_{i1}) : [\![(\mathcal{V}_i)^{p_i, AS'_{i1}}_{Ctx_i}(x)]\!]_{agcnf''_1} \cap \mathcal{G} = \emptyset \tag{86}$$

$$\begin{aligned} &\mathit{high}^{\mathcal{V}_i, Ctx_i, \mathcal{G}}_{\Phi_i, \Lambda_i, \mathcal{X}_i}(agcnf''_1, i, l'_{i1}, X'_{i1}, \iota_{\mathrm{f}}) \wedge X'_{i1} \subseteq \mathcal{X}_i \\ &\quad \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in \mathit{atoms}(\mathit{astmt\text{-}of}(agcnf''_1[i])) \end{aligned} \tag{87}$$

where $AS'_{i1} = \mathit{astmt\text{-}of}(\mathit{alcnf}'_{i1})$. Using Lemma 27, we can obtain

$$agcnf''_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_2 \tag{88}$$

Using (86), we can obtain $\forall y \in \mathit{lvars}(\mathcal{V}_i, \mathcal{G}) : [\![y@p_i]\!]_{agcnf''_1} = [\![y@p_i]\!]_{agcnf_2}$. Hence, we have

$$\forall j \neq i : agcnf''_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2 \tag{89}$$

using Lemma 25. Using (88), (87), and (81), we can obtain

$$agcnf''_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$$

Hence, we have

$$\forall j : agcnf''_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2 \tag{90}$$

Using (84), and (57), we can obtain

$$\begin{aligned} &\mathit{high}^{\mathcal{V}_k, Ctx_k, \mathcal{G}}_{\Phi_k, \Lambda_k, \mathcal{X}_k}(agcnf''_1, k, l'_{k1}, X'_{k1}, \iota_{\mathrm{f}}) \wedge X'_{k1} \subseteq \mathcal{X}_k \\ &\quad \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in \mathit{atoms}(\mathit{astmt\text{-}of}(agcnf''_1[k])) \end{aligned} \tag{91}$$

Using (56), (91), (60), and $agcnf_1[k] \xrightarrow{c?\vec{v}} \mathit{alcnf}'_{k1}$, all the pre-conditions of Lemma 22 can be established. Using Lemma 22 we obtain

$$\forall x \in \mathit{upd\text{-}next}(AS_{k1}) : [\![(\mathcal{V}_k)^{p_k, AS'_{k1}}_{Ctx_k}(x)]\!]_{agcnf'_1} \cap \mathcal{G} = \emptyset \tag{92}$$

$$\begin{aligned} &\mathit{high}^{\mathcal{V}_k, Ctx_k, \mathcal{G}}_{\Phi_k, \Lambda_k, \mathcal{X}_k}(agcnf'_1, k, l'_{k1}, X'_{k1}, \iota_{\mathrm{f}}) \wedge X'_{k1} \subseteq \mathcal{X}_k \\ &\quad \vee {}^{\iota_{\mathrm{f}}}\mathsf{stop} \in \mathit{atoms}(\mathit{astmt\text{-}of}(agcnf'_1[k])) \end{aligned} \tag{93}$$

where $AS'_{k1} = \mathit{astmt\text{-}of}(\mathit{alcnf}'_{k1})$. Using Lemma 27, we can obtain

$$agcnf'_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, k}{=}} agcnf_2 \tag{94}$$

Using (92) and (89), we can obtain

$$\forall y \in \mathit{lvars}(\mathcal{V}_k, \mathcal{G}) : [\![y@p_k]\!]_{agcnf'_1} = [\![y@p_k]\!]_{agcnf_2}$$

Hence, we have $\forall j \neq k : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$ using Lemma 25 with (90). Hence, we have $\forall j : agcnf'_1 \overset{\mathcal{V}_j, Ctx_j, \mathcal{G}}{\underset{\Phi_j, \Lambda_j, \mathcal{X}_j, j}{\approx}} agcnf_2$ using (93), (84), and (94). We can obtain

$$no\text{-}upd(cstmt\text{-}of(strip\text{-}gcnf(agcnf'_1)), \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$
$$no\text{-}upd(cstmt\text{-}of(strip\text{-}gcnf(agcnf_2)), \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$$

from (57) and (58). Hence, we have

$$agcnf'_1 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\approx}} agcnf_2$$

We have $\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf_2}$ because $\alpha_1 = \tau$ and $\alpha_2 = \diamond$. Let $atr'_2 = atr_2.\diamond.agcnf_2$. We have $\lfloor atr'_1 \rfloor^{\mathcal{G}} = \lfloor atr'_2 \rfloor^{\mathcal{G}}$ because of $\lfloor \alpha_1 \rfloor^{\mathcal{G}}_{agcnf_1} = \lfloor \alpha_2 \rfloor^{\mathcal{G}}_{agcnf_2}$ and $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$.

  - Suppose $\alpha_1 = \diamond$. The sub-case where $\alpha_2 \neq \diamond$ is analogous to the case where $\alpha_1 \neq \diamond$ and $\alpha_2 = \diamond$. The sub-case where $\alpha_2 = \diamond$ is trivial.

- Suppose $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ is established using the first rule, while $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$ is established using the second rule. For the sub-cases where $agcnf_1$ can perform a non-$\diamond$ action under ii_1, or $agcnf_2$ can perform a non-$\diamond$ action under ii_2, a contradiction can be derived using Lemma 22 and Lemma 19. Hence, these sub-cases are impossible. The other sub-case where both $agcnf_1$ and $agcnf_2$ can only perform $\diamond$ under ii_1 and ii_2, respectively, is trivial.
- Suppose $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ is established using the second rule, while $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$ is established using the first rule. The reasoning is analogous to the previous sub-case.
- Suppose both $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ and $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$ are established using the second rule for deriving the relation. The reasoning needed is analogous to the sub-case where both $agcnf_1 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\approx}} agcnf_2$ and $agcnf_1 \overset{\mathcal{V}_k, Ctx_k, \mathcal{G}}{\underset{\Phi_k, \Lambda_k, \mathcal{X}_k, k}{\approx}} agcnf_2$ are established using the first rule for deriving the relation, with the main difference being the use of Lemma 26 instead of Lemma 28 for establishing the simulation for the i-th and the k-th processes.

The proof of Lemma 29 is complete by the case analysis above. □

Based on the preceding lemmas, the proof of Theorem 1 is given below.

Proof (of Theorem 1). Let $CS = p_1 : S_1 || \ldots || p_n : S_n$ be a concurrent statement such that $\mathcal{C}, \vec{\mathcal{V}} \vdash^{\vec{\Phi}}_{\vec{\Lambda}} CS$ can be derived. Assume $\vec{\Phi}$ satisfies $\forall i : asst(\Phi_i, CS, i)$, and $\vec{\Lambda}$ satisfies $\forall i : live(\Lambda_i, CS, i)$. Let ξ_1 and ξ_2 be two strategies, and $\mathcal{G} \in \mathcal{P}(Pr)$, with $\xi_1 \overset{\mathcal{G}}{=} \xi_2$.

We first inductively show that for all $atr_1 \in atraces\text{-}of_{\xi_1}(CS)$, there exists $atr_2 \in atraces\text{-}of_{\xi_2}(CS)$ such that $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$, and $last(atr_1) \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} last(atr_2)$. Note that CS is itself an augmented concurrent statement. The induction is on the number of global configurations in atr_1.

- Case atr_1 contains one global configuration.
  We have $atr_1 = \langle \langle p_1, S_1, m_\star \rangle, \ldots, \langle p_n, S_n, m_\star \rangle \rangle$ where $m_\star$ is the initial memory state.
  Take $atr_2 = \langle \langle p_1, S_1, m_\star \rangle, \ldots, \langle p_n, S_n, m_\star \rangle \rangle$.
  We have $\lfloor atr_1 \rfloor^{\mathcal{G}} = \epsilon$ and $\lfloor atr_2 \rfloor^{\mathcal{G}} = \epsilon$. Hence, $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$ holds.
  Let $agcnf_0 = \langle \langle p_1, S_1, m_\star \rangle, \ldots, \langle p_n, S_n, m_\star \rangle \rangle$. We have $agcnf_0 = last(atr_1)$ and $agcnf_0 = last(atr_2)$.
  We move on to show that $agcnf_0 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} agcnf_0$ holds. From $\mathcal{C}, \vec{\mathcal{V}} \vdash^{\vec{\Phi}}_{\vec{\Lambda}} CS$ we have $nip(\mathcal{C}, \vec{\mathcal{V}})$, and $no\text{-}upd(acstmt\text{-}of(agcnf_0), \vec{\mathcal{V}}, \vec{\Lambda}, \vec{\mathcal{X}})$. Pick an arbitrary $i \in \{1, \ldots, n\}$. From $\mathcal{C}, \vec{\mathcal{V}} \vdash^{\vec{\Phi}}_{\vec{\Lambda}} CS$ we have $\mathcal{C}, \mathcal{V}_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (\star, \emptyset)\ S_i\ (l_i, X_i) : \iota_{\mathrm{f}}$ for some l_i and X_i. Hence, there exists some Ctx_i such that $\mathcal{C}, \mathcal{V}_i, Ctx_i \vdash^{\Phi_i, \Lambda_i}_{p_i} (\star, \emptyset)\ S_i\ (l_i, X_i) : \iota_{\mathrm{f}}$ can be derived. In addition, it can be established that $agcnf_0 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, i}{=}} agcnf_0$. It is not difficult to show that $agcnf_0 \overset{\mathcal{V}_i, Ctx_i, \mathcal{G}}{\underset{\Phi_i, \Lambda_i, \mathcal{X}_i, i}{\cong}} agcnf_0$ holds. Hence, $agcnf_0 \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} agcnf_0$ can be established.
- Case atr_1 contains $k > 1$ augmented global configurations. We have $atr_1 = atr_1'.\alpha_1.gcnf_1$ for some atr_1', α_1, and $agcnf_1$. Here, atr_1' contains $k-1$ augmented global configurations, and it holds that $atr_1' \in atraces\text{-}of_{\xi_1}(CS)$. By the induction hypothesis, there exists some atr_2' such that $atr_2' \in atraces\text{-}of_{\xi_2}(CS)$, $\lfloor atr_1' \rfloor^{\mathcal{G}} = \lfloor atr_2' \rfloor^{\mathcal{G}}$, and $last(atr_1') \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} last(atr_2')$.
  Using Lemma 29, we obtain that there exists some atr_2 such that $atr_2' \rightarrow_{\xi_2} atr_2$, $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$, and $last(atr_1) \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} last(atr_2)$. It can also be derived that $atr_2 \in atraces\text{-}of_{\xi_2}(CS)$ because $atr_2' \in atraces\text{-}of_{\xi_2}(CS)$, and $atr_2' \rightarrow_{\xi_2} atr_2$.

It has now been established that under arbitrarily given ξ_1, ξ_2 and $\mathcal{G}$ such that $\xi_1 \overset{\mathcal{G}}{=} \xi_2$, for all $atr_1 \in atraces\text{-}of_{\xi_1}(CS)$, there exists $atr_2 \in atraces\text{-}of_{\xi_2}(CS)$ such that $\lfloor atr_1 \rfloor^{\mathcal{G}} = \lfloor atr_2 \rfloor^{\mathcal{G}}$, and $last(atr_1) \overset{\vec{\mathcal{V}}, \vec{Ctx}, \mathcal{G}}{\underset{\vec{\Phi}, \vec{\Lambda}, \vec{\mathcal{X}}}{\cong}} last(atr_2)$. The conclusion of Theorem 1 can now be deduced using Lemma 13. □