Typing access control and secure information flow in sessions

Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini

Typing Access Control and Secure Information Flow in Sessions

Sara Capecchi[a], Ilaria Castellani[b], Mariangiola Dezani-Ciancaglini[a]

[a] *Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy*
[b] *INRIA, 2004 route des Lucioles, 06902 Sophia Antipolis, France*

Abstract

We consider a calculus for multiparty sessions with delegation, enriched with security levels for session participants and data. We propose a type system that guarantees both session safety and a form of access control. Moreover, this type system ensures secure information flow, including controlled forms of declassification. In particular, it prevents information leaks due to the specific control constructs of the calculus, such as session opening, selection, branching and delegation. We illustrate the use of our type system with a number of examples, which reveal an interesting interplay between the constraints of security type systems and those used in session types to ensure properties like communication safety and session fidelity.

Keywords: Communication-centred computing, session types, access control, secure information flow.

1. Introduction

With the advent of web technologies and the proliferation of programmable and interconnectable devices, we are faced today with a powerful and heterogeneous computing environment. This environment is inherently parallel and distributed and, unlike previous computing environments, it heavily relies on communication. It therefore calls for a new programming paradigm which is sometimes called *communication-centred*. Moreover, since computations take place concurrently in all kinds of different devices, controlled by parties which possibly do not trust each other, security properties such as the confidentiality and integrity of data become of crucial importance. The issue is then to develop models, as well as programming abstractions and methodologies, to be able to exploit the rich potential of this new computing environment, while making sure that we can contain its complexity and get around its security vulnerabilities. To this end, calculi and languages for communication-centred programming have to be *security-minded* from their very conception, and make use of specifications not only for data structures, but also for communication interfaces and for security properties.

The aim of this paper is to investigate type systems for safe and secure sessions. A *session* is an abstraction for various forms of "structured communication" that may occur in a parallel and distributed computing environment. Examples of sessions are a client-service negotiation, a financial transaction, or a multiparty interaction among different services within a web application.

Language-based support for sessions has now become the subject of active research. Primitives for enabling programmers to code sessions in a flexible way, as well as type systems ensuring the compliance of programs to session specifications (session types), have been studied in a variety of calculi and languages in the last decade. *Session types* were originally introduced in a variant of the pi-calculus [29]. We refer to [12] for a survey on the session type literature. The key properties ensured by session types are *communication safety*, namely the consistency of the communication patterns exhibited by the partners (implying the absence of communication errors), and *session fidelity*, ensuring that channels which carry messages of different types do it in a specific order.

Enforcement of security properties via session types has been studied in [3, 24]. These papers propose a compiler which, given a multiparty session description, implements cryptographic protocols that guarantee *session execution integrity*. A type system assuring that services always comply with the policy regulating data exchange is presented in [20]. The question of ensuring *access control* in binary sessions has been recently addressed in [19] for the Calculus of Services with Pipelines and Sessions of [4], where delegation is absent. On the other hand, the property of *secure information flow* has not been investigated within session calculi so far. This property, first studied in the early eighties [14],

[☆] Work partially funded by the EU Project 257414 ASCENS, by the ICT COST Action IC1201 BETTY, by the MIUR PRIN Project CINA Prot. 2010LHT4KM and by the Torino University and Compagnia San Paolo Project SALT.

has regained interest in the last decade, due to the evolution of the computing environment. It has now been thoroughly studied for both programming languages (cf [25] for a review) and process calculi [13, 15, 18].

In this paper, we address the question of incorporating mandatory access control and secure information flow within session types. We consider a calculus for multiparty sessions with delegation, enriched with security levels for both session participants and data, and providing a form of declassification for data [27], as required by most practical applications. We propose a type system that ensures access control, namely that each participant receives data of security level less than or equal to its own. For instance, in a well-typed session involving a Customer, a Seller and a Bank, the secret credit card number of the Customer will be communicated to the Bank, but not to the Seller. Moreover, our type system prevents insecure flows that could occur via the specific constructs of the language, such as session opening, selection, branching and delegation. Finally, we show that it allows controlled forms of declassification during communication, namely those explicitly permitted by the sender and respecting the access control policy in the receiver. Our work reveals an interesting interplay between the constraints of security type systems and those used in session types to ensure properties like communication safety and session fidelity.

The rest of the paper is organised as follows. Section 2 motivates our access control and declassification policies with an example. Sections 3 and 4 introduce the syntax and semantics of our calculus. Section 5 defines the secure information flow property. Section 6 illustrates this property by means of examples. Section 7 presents our type system and Section 8 establishes its soundness. Section 9 discusses future work. Appendix Appendix A discusses a more expressive notion of security (called *robust security*) and shows that our type system remains sound for this new notion. Appendix Appendix B summarises various features of the running example introduced in Section 2.

This paper is a deeply revised version of [9], with improved definitions, new examples and results and complete proofs. In a companion paper [8] we give a monitored semantics, which only assures secure information flow, for a simplification of the present calculus without declassification and delegation.

2. An Example on Access Control and Declassification

In this section we illustrate by an example the basic features of our typed calculus, as well as our access control policy and our specific form of declassification. The question of secure information flow will only be briefly touched upon here, while it will be discussed in depth in Sections 5 and 6, once the semantics of the calculus has been formally defined. Suffice it to say, for the time being, that a process has secure information flow if none of its actions depends on some *input action* of higher or incomparable level.

For the sake of simplicity, in this introductory example we shall only consider binary sessions, and two security levels for data and participants, namely high (secret) and low (public)[1]. In our calculus, the level of data represents their degree of confidentiality, while the level of participants represents their reading rights: thus a high participant can read both high and low data, whereas a low participant can only read low data.

Consider the following scenario, involving a bookseller, a client and a bank. A client `C` sends the title of a book to a bookseller `S`. The seller `S` *delegates* to a bank `B` both the reception of `C`'s credit card number and its validity check. This frequently happens in online shopping, when clients are redirected to a new page for a secure payment. Delegation is crucial for assuring the secrecy of the credit card number, which should be read by `B` but not by `S`. To express this access control constraint, the typing will assign high security level to the credit card and to `B`, and low security level to `S`. Then `B` notifies `S` about the result of the validity check: since this result is low and the card is high, for this information flow to be admissible a *declassification* is needed. Finally, if the credit card is valid, `C` receives a delivery date from `S`, otherwise the deal falls through. More precisely, the protocol is as follows:

1. `C` opens a connection with `S` and sends a title to `S`;
2. `S` opens a connection with `B` and *delegates* to `B` part of his conversation with `C`;
3. `C` sends his secret credit card number *apparently* to the low party `S` but *really* - thanks to delegation - to the high party `B`; moreover `C` authorises the high recipient of his card number to declassify it (although not knowing `B`, `C` knows that a delegation to a high party must take place since `S` is low);
4. `B` reads the credit card number and then delegates back to `S` the conversation with `C`;
5. `B` selects the answer `ok` or `ko` for the low party `S` depending on the validity or not of the credit card, thus performing a declassification;
6. `S` sends to `C` either `ok` and a date, or just `ko`, depending on the label `ok` or `ko` chosen by `B`.

[1] However, our formal treatment will cover the general case of a finite lattice of security levels, and a number of examples involving multiparty sessions will be discussed in Section 6.

$$
\begin{array}{lcl}
\mathtt{I} & = & \bar{a}[2] \mid \bar{b}[2] \\
\mathtt{C} & = & a[1](\alpha_1).\alpha_1!\langle 2,\mathsf{Title}^{\bot}\rangle.\alpha_1!^{\bot}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\ \alpha_1\&(2,\{\mathsf{ok}:\alpha_1?(2,date^{\bot}).\mathbf{0},\mathsf{ko}:\mathbf{0}\}) \\
\mathtt{S} & = & a[2](\alpha_2).\alpha_2?(1,x^{\bot}).b[2](\beta_2).\beta_2!\langle\langle 1,\alpha_2\rangle\rangle.\beta_2?((1,\eta)). \\
 & & \qquad\qquad \beta_2\&(1,\{\mathsf{ok}:\eta\oplus\langle 1,\mathsf{ok}\rangle.\eta!\langle 1,\mathsf{Date}^{\bot}\rangle.\mathbf{0},\mathsf{ko}:\eta\oplus\langle 1,\mathsf{ko}\rangle.\mathbf{0}\}) \\
\mathtt{B} & = & b[1](\beta_1).\beta_1?((2,\zeta)).\zeta?^{\top}(2,cc^{\bot}).\beta_1!\langle\langle 2,\zeta\rangle\rangle.\ \mathsf{if}\ \mathit{valid}(cc^{\bot})\ \mathsf{then}\ \beta_1\oplus\langle 2,\mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ \beta_1\oplus\langle 2,\mathsf{ko}\rangle.\mathbf{0}
\end{array}
$$

Table 1: Processes of the C, S, B example.

1.	$\mathtt{C}\to\mathtt{S}:\left\langle\mathsf{String}^{\bot}\right\rangle$	
2.	$\mathtt{S}\,\text{↱}\delta$	$\mathtt{S}\to\mathtt{B}:\langle T\rangle$
3.	$\mathtt{C}\to\mathtt{S}:\left\langle\mathsf{Number}^{\top\downarrow\bot}\right\rangle$	
4.		$\mathtt{B}\to\mathtt{S}:\langle T'\rangle$
5.		$\mathtt{B}\to\mathtt{S}:\{\mathsf{ok}:\mathsf{end},\mathsf{ko}:\mathsf{end}\}$
6.	$\mathtt{S}\to\mathtt{C}:\{\mathsf{ok}:\mathtt{S}\to\mathtt{C}:\left\langle\mathsf{String}^{\bot}\right\rangle;\mathsf{end},\mathsf{ko}:\mathsf{end}\}$	

Table 2: Global types of the C, S, B example.

Notice that this scenario could not be represented by a unique protocol involving C, S, and B, i.e., by a ternary session, since this would imply that the client knows the bank used by the bookseller for payments.
In our calculus, which is an enrichment with security levels of the calculus in [2], this scenario may be described as the parallel composition of the processes shown in Table 1, where security levels appear as superscripts on both data and operators. Here $\top$ means "secret" and $\bot$ means "public", and a level appears on an I/O operator whenever a declassification is involved. For the sake of simplicity, we omit the security levels on the other operators. They will be filled in later in the paper, and this running example, completed with all security levels, will be recalled in Table 6.

A session is a particular activation of a service, involving a number of parties with predefined roles. Here C and S communicate by opening a session on service a, while S and B communicate by opening a session on service b. The initiators $\bar{a}[2]$ and $\bar{b}[2]$ specify the number of participants of each service. We associate integers with participants in services: here C=1, S=2 in service a and B=1, S=2 in service b.

In process C, the prefix $a[1](\alpha_1)$ means that C wants to act as participant 1 in service a using channel α_1, matching channel α_2 of participant 2, who is S. When the session is established, C sends to S a title of level $\bot$ and a credit card number of level $\top$, indicating (by the superscript $\bot$ on the output operator) that the credit card number may be declassified to $\bot$. Then he waits for either ok, followed by a date, or ko.

Process S receives a value from C in service a and then enters service b as participant 2. Here the output $\beta_2!\langle\langle 1,\alpha_2\rangle\rangle$ sends channel α_2 to the participant 1 of b, who is B, thus delegating to B the use of α_2. This means that B will now act as participant 2 in service a, thus directly interacting with C. Then S waits back for the channel α_2 from B. Henceforth, S communicates using both channels β_2 and α_2: on channel β_2 he waits for one of the labels ok or ko, which he then forwards to C on α_2, sending also a date if the label is ok.

Forgetting about session opening and abstracting from values to types, we may represent the whole communication protocol by the *global types* of Table 2 (where we use B, C, S instead of 1, 2), where the left-hand side and right-hand side describe services a and b, respectively. Line 1 says that C sends a String of level $\bot$ to S. In line 2, $\mathtt{S}\,\text{↱}\delta$ means that the channel from S to C is delegated: this delegation is realised by the transmission of the channel (with type T) from S to B, as shown on the right-hand side. Line 3 says that C sends a Number of level $\top$ to S, allowing him to declassify it to $\bot$. Notice that due to the previous delegation the Number is received by B and not by S. Line 4 describes a delegation which is the inverse of that in Line 2: here the (behavioural) type of the channel has changed to T', since the channel has already been used to receive the Number. Line 5 says that B sends to S one of the labels ok or ko. Finally, line 6 says that S sends to C either the label ok followed by a String of level $\bot$, or the label ko. Since B's choice of the label ok or ko depends on a test on the Number, it is crucial that Number be previously declassified to $\bot$, otherwise the reception of a String of level $\bot$ by C would depend on a value of level $\top$ (this is where secure information flow comes into play).

Type T represents the conversation between C and S after the first communication, seen from the viewpoint of S. Convening that $?(-),!\langle-\rangle$ represent input and output in types, that ";" stands for sequencing and that $\oplus\langle-\{-\}\rangle$ represents the choice of sending one among different labels, the *session type* T is:

P	::=	$\bar{u}[n]$	n-ary session initiator
	$\mid$	$u[\mathtt{p}](\alpha).P$	$\mathtt{p}$-th session participant
	$\mid$	$c!^{\ell}\langle \Pi, e\rangle.P$	Value sending
	$\mid$	$c?^{\ell}(\mathtt{p}, x^{\ell'}).P$	Value receiving
	$\mid$	$c!^{\ell}\langle\langle \mathtt{q}, c'\rangle\rangle.P$	Delegation sending
	$\mid$	$c?^{\ell}((\mathtt{p}, \alpha)).P$	Delegation reception
	$\mid$	$c \oplus^{\ell} \langle \Pi, \lambda\rangle.P$	Selection
	$\mid$	$c\&^{\ell}(\mathtt{p}, \{\lambda_i : P_i\}_{i\in I})$	Branching
	$\mid$	$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	Conditional
	$\mid$	$P \mid Q$	Parallel
	$\mid$	$\mathbf{0}$	Inaction
	$\mid$	$(\nu a^{\ell})P$	Name hiding
	$\mid$	$\mathsf{def}\ D\ \mathsf{in}\ P$	Recursion
	$\mid$	$X\langle e, c\rangle$	Process call
r	::=	$a^{\ell} \mid s$	Service/Session Name
c	::=	$\alpha \mid s[\mathtt{p}]$	Channel
u	::=	$x^{\ell} \mid a^{\ell}$	Identifier
v	::=	$a \mid \mathsf{true} \mid \mathsf{false} \mid \ldots$	Value
e	::=	$x^{\ell} \mid v^{\ell} \mid e\ \mathsf{and}\ e' \mid \mathsf{not}\ e \mid \ldots$	Expression
D	::=	$X(x^{\ell}, \alpha) = P$	Declaration
Π	::=	$\{\mathtt{p}\} \mid \Pi \cup \{\mathtt{p}\}$	Set of participants
ϑ	::=	$v^{\ell\downarrow\ell'} \mid s[\mathtt{p}]^{\ell} \mid \lambda^{\ell}$	Message content
m	::=	$(\mathtt{p}, \Pi, \vartheta)$	Message in transit
h	::=	$m \cdot h \mid \varepsilon$	Queue
H	::=	$H \cup \{s : h\} \mid \emptyset$	**Q**-set

Table 3: Syntax of processes, expressions and queues.

$$?\left(\mathtt{C}, \mathsf{Number}^{\top\downarrow\bot}\right); \oplus\left\langle \mathtt{C}, \{\mathsf{ok} :! \left\langle \mathtt{C}, \mathsf{String}^{\bot}\right\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\right\rangle$$

where the communication partner of $\mathtt{S}$ (namely $\mathtt{C}$) is explicitly mentioned. The session type T' is the rest of type T after the first communication has been done:

$$\oplus\left\langle \mathtt{C}, \{\mathsf{ok} :! \left\langle \mathtt{C}, \mathsf{String}^{\bot}\right\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\right\rangle$$

As hinted previously, to formalise access control we give security levels to service participants, and require that a participant of a given level does not receive data of higher or incomparable level. Since the only secret data in our example is CreditCard, it is natural to associate $\bot$ with participant $\mathtt{S}$ in both services a and b, and $\top$ with participant $\mathtt{B}$ in service b. Notice that participant $\mathtt{C}$ may indifferently have level $\top$ or $\bot$ in service a, since he only sends, but does not receive, the high data CreditCard.

3. Syntax

Our calculus for multiparty asynchronous sessions is essentially the same as that considered in [2], with the addition of runtime configurations and security levels.

Let $(\mathscr{S},\leq)$ be a finite lattice of *security levels*, ranged over by ℓ,ℓ'. We denote by $\sqcup$ and $\sqcap$ the join and meet operations on the lattice, and by $\bot$ and $\top$ its minimal and maximal elements.

We assume the following sets:

- *service names*, ranged over by $a,b,\ldots$, each of which has an *arity* $n \geq 2$ (its number of participants) and a security level ℓ,
- *value variables*, ranged over by $x,y,\ldots$, all decorated with security levels,
- *identifiers*, i.e., service names and value variables, ranged over by $u,w,\ldots$, all decorated with security levels,
- *channel variables*, ranged over by $\alpha,\beta,\ldots$,
- *labels*, ranged over by $\lambda,\lambda',\ldots$ (acting like labels in labelled records).

Values v are either service names or basic values (boolean values, integers, etc.). When treated as an expression, a value is decorated with a security level ℓ; when used in a message, it is decorated with a declassified level of the form $\ell \downarrow \ell'$, where $\ell' \leq \ell$. When $\ell' = \ell$, we will write simply ℓ instead of $\ell \downarrow \ell$. In this case we will speak of *trivial declassification*. Let us point out that declassification of services is not allowed in our calculus, for lack of evidence of its usefulness (see Example 6.5). Hence, if v is a service name, its level will always be of the simple form ℓ.

Sessions, the central abstraction of our calculus, are denoted with $s,s'\ldots$. A session represents a particular instance or activation of a service. Hence sessions only appear at runtime. We use $\mathtt{p}$, $\mathtt{q}$,... to denote the *participants* of a session. In an n-ary session (a session corresponding to an n-ary service) $\mathtt{p}$, $\mathtt{q}$ are assumed to range over the natural numbers $1,\ldots,n$. We denote by Π a non empty set of participants. Each session s has an associated set of *channels with role* $s[\mathtt{p}]$, one for each participant. Channel $s[\mathtt{p}]$ is the private channel through which participant $\mathtt{p}$ communicates with the other participants in the session s. A new session s on an n-ary service a^ℓ is opened when the *initiator* $\bar{a}^\ell[n]$ of the service synchronises with n processes of the form $a^\ell[1](\alpha_1).P_1,\ldots,a^\ell[n](\alpha_n).P_n$. We use c to range over channel variables and channels with roles. Finally, we assume a set of *process variables* $X,Y,\ldots$, in order to define recursive behaviours.

As in [16], in order to model TCP-like asynchronous communications (with non-blocking send but message order preservation between a given pair of participants), we use *queues of messages*, denoted by h; an element of h may be a value message $(\mathtt{p},\Pi,v^{\ell\downarrow\ell'})$, indicating that the value v^ℓ is sent by participant $\mathtt{p}$ to all participants in Π, with the right of declassifying it from level ℓ to level ℓ'; a channel message $(\mathtt{p},\mathtt{q},s[\mathtt{p}']^\ell)$, indicating that $\mathtt{p}$ delegates to $\mathtt{q}$ the role of $\mathtt{p}'$ with level ℓ in the session s; and a label message $(\mathtt{p},\Pi,\lambda^\ell)$, indicating that $\mathtt{p}$ selects the process with label λ among those offered by the set of participants Π. The empty queue is denoted by ε, and the concatenation of a new message m to a queue h by $h\cdot m$. Conversely, $m\cdot h$ means that m is the head of the queue. Since there may be nested and parallel sessions, we distinguish their queues by naming them. We denote by $s:h$ the *named queue* h associated with session s. We use H,K to range over sets of named queues, also called **Q**-sets.

The set of *expressions*, ranged over by $e,e',\ldots$, and the set of *processes*, ranged over by $P,Q\ldots$, are given in Table 3, together with the *runtime syntax* of the calculus (sessions, channels with role, messages, queues), which appears shaded for ease of reading. We say that a process is a *user process* if it can be written without using runtime syntax.

Let us informally comment on the primitives of the language, whose operational semantics will be given in the next section. The primitive for *session initiation* $\bar{u}[n]$ opens a new n-ary session for the shared service u by synchronising with n processes of the form $u[\mathtt{p}](\alpha_\mathtt{p}).P_\mathtt{p}$, for $\mathtt{p}=1,\ldots,n$. The effect of this synchronisation is to create a new session s with an associated queue $s:\varepsilon$, and to replace the channel $\alpha_\mathtt{p}$ in each process $P_\mathtt{p}$ by the channel with role $s[\mathtt{p}]$. This is the only synchronous interaction of the calculus. All the other communications, which take place within an established session, are performed asynchronously in two steps, via push and pop operations on the queue associated with the session, using the next three pairs of primitives: the send and receive of a value; the send and receive of a delegation (where one participant transmits to another one the capability of participating in another session with a given role); and the selection and branching operators (where one participant chooses one of the branches offered by another participant). The send/receive and choice primitives are decorated with security levels, whose use will be justified later.

When there is no risk of confusion we will omit the set delimiters $\{,\}$, particularly around singletons.

4. Operational Semantics

In this section we define the operational semantics of our calculus, which consists of a reduction relation on configurations, coupled with a structural equivalence.

A *configuration* is a pair $C = < P\ ,\ H >$ of a process P and a **Q**-set H, possibly restricted with respect to service and session names, or a parallel composition $C \| C$ of configurations. In a configuration of the form $(\nu s) < P\ ,\ H >$, all occurrences of $s[\mathtt{p}]$ in P and H and of s in H are bound. By abuse of notation we will often write P instead of $< P\ ,\ \emptyset >$.

$$P \mid \mathbf{0} \equiv P \quad P \mid Q \equiv Q \mid P \quad (P \mid Q) \mid R \equiv P \mid (Q \mid R) \quad (\nu r)P \mid Q \equiv (\nu r)(P \mid Q)$$

$$(\nu r r')P \equiv (\nu r' r)P \quad (\nu r)\mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ \mathbf{0} \equiv \mathbf{0} \quad \mathsf{def}\ D\ \mathsf{in}\ (\nu r)P \equiv (\nu r)\mathsf{def}\ D\ \mathsf{in}\ P$$

$$(\mathsf{def}\ D\ \mathsf{in}\ P) \mid Q \equiv \mathsf{def}\ D\ \mathsf{in}\ (P \mid Q) \quad \mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ P) \equiv \mathsf{def}\ D'\ \mathsf{in}\ (\mathsf{def}\ D\ \mathsf{in}\ P)$$

$$h \cdot (\mathtt{p}, \Pi, \vartheta) \cdot (\mathtt{p}', \Pi', \vartheta') \cdot h' \equiv h \cdot (\mathtt{p}', \Pi', \vartheta') \cdot (\mathtt{p}, \Pi, \vartheta) \cdot h' \qquad \text{if } \Pi \cap \Pi' = \emptyset \text{ or } \mathtt{p} \neq \mathtt{p}'$$

$$h \cdot (\mathtt{p}, \Pi, \vartheta) \cdot h' \equiv h \cdot (\mathtt{p}, \Pi', \vartheta) \cdot (\mathtt{p}, \Pi'', \vartheta) \cdot h' \qquad \text{if } \Pi = \Pi' \cup \Pi'' \text{ and } \Pi' \cap \Pi'' = \emptyset$$

$$h \equiv h' \quad \Rightarrow \quad H \cup \{s : h\} \equiv H \cup \{s : h'\}$$

$$P \equiv Q \text{ and } H \equiv K \quad \Rightarrow \quad <P\ ,\ H> \equiv <Q\ ,\ K> \qquad C \equiv C' \quad \Rightarrow \quad (\nu r)C \equiv (\nu r)C'$$

$$(\nu \tilde{r}) <P\ ,\ H> \parallel (\nu \tilde{r'}) <Q\ ,\ K> \equiv (\nu \tilde{r r'}) <P \mid Q\ ,\ H \cup K> \quad \text{if } \mathrm{qn}(H) \cap \mathrm{qn}(K) = \emptyset$$

$$(\nu r r')C \equiv (\nu r' r)C \quad (\nu r)(<P\ ,\ H>) \equiv <(\nu r)P\ ,\ H> \quad (\nu r)(<P\ ,\ H>) \equiv <P\ ,\ (\nu r)H>$$

Table 4: Structural equivalence.

Let $\mathrm{qn}(H)$ stand for the queue names of H. Then the *structural equivalence* $\equiv$ for processes, queues and configurations is given in Table 4, where assuming Barendregt convention no bound name can occur free or in two different bindings. The rules for processes are standard [22]. Among the rules for queues, we have one for commuting independent messages and another one for splitting a message for multiple recipients. Modulo $\equiv$ each configuration has the form $(\nu \tilde{r}) <P\ ,\ H>$, where $(\nu \tilde{r})C$ stands for $(\nu r_1) \cdots (\nu r_k)C$ if $\tilde{r} = r_1 \cdots r_k$. In a configuration $(\nu a^\ell)C$, we assume that α-conversion on the name a^ℓ preserves the level ℓ.

The transitions for configurations have the form $C \longrightarrow C'$. They are derived using the reduction rules in Table 5. Let us briefly comment on these rules.

Rule [Link] describes the initiation of a new session among n processes, corresponding to an activation of the service a^ℓ of arity n. After the connection, the participants share a private session name s and the corresponding queue, initialised to $s : \varepsilon$. In each participant $P_\mathtt{p}$, the channel variable $\alpha_\mathtt{p}$ is replaced by the channel with role $s[\mathtt{p}]$.

The output rules [Send], [DelSend] and [Label] push values, channels and labels, respectively, into the queue $s : h$. In rule [Send], $e \downarrow v^\ell$ denotes the evaluation of the expression e to the value v^ℓ, where if v is a basic value, ℓ is the join of the security levels of the variables and values occurring in e, while if v is a service name, ℓ is simply the level associated with the service. The superscript ℓ' on the output sign indicates that v^ℓ can be declassified to level ℓ' when received by an input process $s[\mathtt{q}]?^\ell(\mathtt{p}, x^{\ell'}).P$. This is why the value needs to be recorded with both levels in the queue. Recall that in case v is a service name, then $\ell' = \ell$ (services cannot be declassified).

The rules [Rec], [DelRec] and [Branch] perform the corresponding complementary operations. Rule [ScopC] is a standard contextual rule. In this rule, Barendregt convention ensures that the names in $\tilde{s}$ are disjoint from those in $\tilde{r}$ and do not appear in C''. We will often omit the level on the I/O operators $s[\mathtt{p}]!^{\ell'}\langle \Pi, e \rangle.P$ and $s[\mathtt{q}]?^\ell(\mathtt{p}, x^{\ell'}).P$ when $\ell = \ell'$, using the simpler notation $s[\mathtt{p}]!\langle \Pi, e \rangle.P$ and $s[\mathtt{q}]?(\mathtt{p}, x^\ell).P$.

As usual, we use $\longrightarrow^*$ for the reflexive and transitive closure of $\longrightarrow$.

4.1. Semantics of the Client, Seller, Bank example

We illustrate the use of our reduction rules on the introductory example of Section 2. We start by rewriting in Table 6 the various components equipped with all the required security levels. Then the global process has the reductions shown in Table 7, where the continuation processes $P_\mathtt{C}, P_\mathtt{S}, P_\mathtt{B}$ are given by:

$P_\mathtt{C} = s_a[1]\&^\perp(2, \{\mathsf{ok} : s_a[1]?^\perp(2, date^\perp).\mathbf{0}, \mathsf{ko} : \mathbf{0}\})$
$P_\mathtt{S} = s_b[2]\&^\perp(1, \{\mathsf{ok} : \eta \oplus^\perp \langle 1, \mathsf{ok}\rangle.\eta!^\perp\langle 1, \mathsf{Date}^\perp\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^\perp \langle 1, \mathsf{ko}\rangle.\mathbf{0}\})$
$P_\mathtt{B} = s_b[1]!^\perp\langle\langle 2, s_a[2]\rangle\rangle.\mathsf{if}\ valid(cc^\perp)\ \mathsf{then}\ s_b[1] \oplus^\perp \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ s_b[1] \oplus^\perp \langle 2, \mathsf{ko}\rangle.\mathbf{0}$

$$a^\ell[1](\alpha_1).P_1 \mid \ldots \mid a^\ell[n](\alpha_n).P_n \mid \bar{a}^\ell[n] \longrightarrow (\nu s) < P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\} \;,\; s : \varepsilon >$$
[Link]

$$< s[\mathtt{p}]!^{\ell'}\langle \Pi, e\rangle.P \;,\; s : h > \longrightarrow < P \;,\; s : h \cdot (\mathtt{p}, \Pi, v^{\ell\downarrow\ell'}) > \qquad (e\downarrow v^\ell)$$
[Send]

$$< s[\mathtt{q}]?^\ell(\mathtt{p}, x^{\ell'}).P \;,\; s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h > \longrightarrow < P\{v^{\ell'}/x^{\ell'}\} \;,\; s : h >$$
[Rec]

$$< s[\mathtt{p}]!^\ell\langle\langle \mathtt{q}, s'[\mathtt{p}']\rangle\rangle.P \;,\; s : h > \longrightarrow < P \;,\; s : h \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^\ell) >$$
[DelSend]

$$< s[\mathtt{q}]?^\ell((\mathtt{p}, \alpha)).P \;,\; s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^\ell) \cdot h > \longrightarrow < P\{s'[\mathtt{p}']/\alpha\} \;,\; s : h >$$
[DelRec]

$$< s[\mathtt{p}] \oplus^\ell \langle \Pi, \lambda\rangle.P \;,\; s : h > \longrightarrow < P \;,\; s : h \cdot (\mathtt{p}, \Pi, \lambda^\ell) >$$
[Label]

$$< s[\mathtt{q}]\&^\ell(\mathtt{p}, \{\lambda_i : P_i\}_{i\in I}) \;,\; s : (\mathtt{p}, \mathtt{q}, \lambda^\ell_{i_0}) \cdot h > \longrightarrow < P_{i_0} \;,\; s : h > \quad (i_0 \in I)$$
[Branch]

$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow P \quad (e \downarrow \mathsf{true}^\ell) \qquad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \longrightarrow Q \quad (e \downarrow \mathsf{false}^\ell)$$
[If-T, If-F]

$$\mathsf{def}\ X(x^\ell, \alpha) = P\ \mathsf{in}\ (X\langle e, s[\mathtt{p}]\rangle \mid Q) \longrightarrow \mathsf{def}\ X(x^\ell, \alpha) = P\ \mathsf{in}\ (P\{v^\ell/x^\ell\}\{s[\mathtt{p}]/\alpha\} \mid Q) \quad (e \downarrow v^\ell)$$
[Def]

$$< P \;,\; H > \longrightarrow (\nu\tilde{s}) < P' \;,\; H' > \quad \Rightarrow \quad < \mathsf{def}\ D\ \mathsf{in}\ P \;,\; H > \longrightarrow (\nu\tilde{s}) < \mathsf{def}\ D\ \mathsf{in}\ P' \;,\; H' >$$
[Defin]

$$C \longrightarrow (\nu\tilde{s})C' \quad \Rightarrow \quad (\nu\tilde{r})(C \,\|\, C'') \longrightarrow (\nu\tilde{r})(\nu\tilde{s})(C' \,\|\, C'')$$
[ScopC]

Table 5: Reduction rules.

5. Information Flow Security in Sessions

Let us turn now to the main concern of this work, namely that of ensuring *secure information flow* [11] within sessions. We shall be interested in the classical property of *noninterference* (NI) [14], combined with a limited form of *declassification* [1, 27], which is assumed to only take place during the transmission of a basic value. The property of NI requires that there is no dependency or *information flow* from objects of a given level to objects of lower or incomparable level [30, 28, 25]. Such an information flow is considered insecure since by observing the latter objects one could potentially reconstruct information about the former.
To set the stage for our information flow analysis, the first questions to ask are:

1. Which objects of the calculus should carry security levels?
2. Which information leaks can occur and how can they be detected?

As concerns objects, we shall see that besides basic values, also labels, delegated channels and services need security levels. Since this question requires some discussion, which is best understood through examples, we defer it to the next section, just assuming here as a fact that queue messages have the form $(\mathtt{p}, \Pi, \vartheta)$, where ϑ may be of the form $v^{\ell\downarrow\ell'}$, λ^ℓ or $s[\mathtt{p}]^\ell$. We shall focus here on the observation model, which will be based on a notion of bisimulation as is now standard for concurrent processes [28, 26].

We are looking for a security property which is *persistent* in the sense of [5], namely which holds in any reachable state of a process, assuming the process may be restarted with fresh **Q**-sets at each step. This means that we view processes as evolving in a dynamic and potentially hostile environment, where at each step an attacker may change the high context by adding or subtracting messages, or changing their content.

Before formally introducing the security property, let us illustrate by a couple of examples the kinds of insecure flow that may arise in our calculus. Essentially, we have an insecure information flow – or *information leak* – when a communication action of level ℓ depends on an input action of level $\ell' \not\leq \ell$.

$$
\begin{array}{lcl}
\mathtt{I} & = & \bar{a}^{\perp}[2] \mid \bar{b}^{\perp}[2] \\
\mathtt{C} & = & a^{\perp}[1](\alpha_1).\alpha_1!^{\perp}\langle 2,\mathsf{Title}^{\perp}\rangle.\alpha_1!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\ \alpha_1\&^{\perp}(2,\{\mathsf{ok}:\alpha_1?^{\perp}(2,date^{\perp}).\mathbf{0},\mathsf{ko}:\mathbf{0}\}) \\
\mathtt{S} & = & a^{\perp}[2](\alpha_2).\alpha_2?^{\perp}(1,x^{\perp}).b^{\perp}[2](\beta_2).\beta_2!^{\perp}\langle\langle 1,\alpha_2\rangle\rangle.\beta_2?^{\perp}((1,\eta)). \\
 & & \qquad\qquad \beta_2\&^{\perp}(1,\{\mathsf{ok}:\eta\oplus^{\perp}\langle 1,\mathsf{ok}\rangle.\eta!^{\perp}\langle 1,\mathsf{Date}^{\perp}\rangle.\mathbf{0},\mathsf{ko}:\eta\oplus^{\perp}\langle 1,\mathsf{ko}\rangle.\mathbf{0}\}) \\
\mathtt{B} & = & b^{\perp}[1](\beta_1).\beta_1?^{\perp}((2,\zeta)).\zeta?^{\top}(2,cc^{\perp}).\beta_1!^{\perp}\langle\langle 2,\zeta\rangle\rangle.\mathsf{if}\ \mathit{valid}(cc^{\perp})\ \mathsf{then}\ \beta_1\oplus^{\perp}\langle 2,\mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ \beta_1\oplus^{\perp}\langle 2,\mathsf{ko}\rangle.\mathbf{0}
\end{array}
$$

Table 6: Processes of the C,S,B example with all security levels.

$$
\begin{array}{l}
\mathtt{I}\mid\mathtt{C}\mid\mathtt{S}\mid\mathtt{B} \quad\longrightarrow \\
(\nu s_a)(<\bar{b}^{\perp}[2]\mid s_a[1]!^{\perp}\langle 2,\mathsf{Title}^{\perp}\rangle.\cdots\mid s_a[2]?^{\perp}(1,x^{\perp}).\cdots\mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a:\varepsilon\}>) \\
\quad\longrightarrow \\
(\nu s_a)(<\bar{b}^{\perp}[2]\mid s_a[1]!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\cdots\mid s_a[2]?^{\perp}(1,x^{\perp}).\cdots\mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a:(1,2,\mathsf{Title}^{\perp\downarrow\perp})\}>) \\
\quad\longrightarrow \\
(\nu s_a)(<\bar{b}^{\perp}[2]\mid s_a[1]!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\cdots\mid b^{\perp}[2](\beta_2).\cdots\mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a:\varepsilon\}>) \\
\quad\longrightarrow \\
(\nu s_a)(\nu s_b)(<s_a[1]!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\cdots\mid s_b[2]!^{\perp}\langle\langle 1,s_a[2]\rangle\rangle.\cdots\mid s_b[1]?^{\perp}((2,\zeta)).\cdots,\ \{s_a:\varepsilon,s_b:\varepsilon\}>) \\
\quad\longrightarrow \\
(\nu s_a)(\nu s_b)(<s_a[1]!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.\cdots\mid s_b[2]?^{\perp}((1,\eta)).\cdots\mid s_b[1]?^{\perp}((2,\zeta)).\cdots,\ \{s_a:\varepsilon,s_b:(2,1,s_a[2]^{\perp})\}>) \\
\quad\longrightarrow \\
(\nu s_a)(\nu s_b)(<s_a[1]!^{\perp}\langle 2,\mathsf{CreditCard}^{\top}\rangle.P_{\mathtt{C}}\mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}}\mid s_a[2]?^{\top}(2,cc^{\perp}).P_{\mathtt{B}},\ \{s_a:\varepsilon,s_b:\varepsilon\}>) \\
\quad\longrightarrow \\
(\nu s_a)(\nu s_b)(<P_{\mathtt{C}}\mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}}\mid s_a[2]?^{\top}(1,cc^{\perp}).P_{\mathtt{B}},\ \{s_a:(1,2,\mathsf{CreditCard}^{\top\downarrow\perp}),s_b:\varepsilon\}>)
\end{array}
$$

Table 7: Some reductions of the C,S,B example.

A typical information leak occurs when a low action is guarded by a high input, as in the following process P, where s is some previously opened session with three participants:

$$(*) \qquad P = s[1]?(2,x^{\top}).s[1]!\langle 3,e\rangle.\mathbf{0} \qquad \text{where } e{\downarrow}v^{\perp}$$

In process P, participant 1 waits for a high message from participant 2, and then sends a low message to participant 3.

This process is insecure because the low output is guarded by a high input whose occurrence depends on whether or not a matching message is offered by the high environment, which is something we do not control. Since input is a blocking operator, the low output will be able to occur in the first case but not in the latter, and the observer will conclude that the low output depends on the high input. Note that in process P the value $v^{\perp}$ of the expression e cannot depend on the value received for $x^{\top}$, since in this case e should contain $x^{\top}$ and therefore its value would have level $\top$[2]. Hence the leak in P is only due to the *presence* or *absence* of a high message in the environment, and not on its value.

Another typical leak occurs when a high input guards a conditional which tests the received value in order to choose between two different low behaviours. Consider the following process:

$$(**) \qquad Q = s[2]!\langle 1,\mathsf{true}^{\top}\rangle.\mathbf{0}\mid s[1]?(2,x^{\top}).\mathsf{if}\ x^{\top}\ \mathsf{then}\ s[1]!\langle 3,\mathsf{true}^{\perp}\rangle.\mathbf{0}\ \mathsf{else}\ s[1]!\langle 3,\mathsf{false}^{\perp}\rangle.\mathbf{0}$$

Again, session s has three participants. Here participant 1 receives a message from participant 2, and depending on its value sends two different low values to participant 3. This process is insecure because the value of the low message sent by participant 1 to participant 3 depends on the value of the high message received by participant 1 from participant 2. Here the leak is caused by the *value* of the high message rather than by its presence or absence.

In fact, Example $(**)$ is not completely correct, since in our persistent notion of security we assume that the attacker can add or withdraw high messages at each step. Hence, after participant 2 has sent the message to participant 1, this message could disappear again from the queue, in which case participant 1 would be blocked just like participant 1 in Example $(*)$. However, Example $(**)$ can be easily adapted by rendering both components of process Q recursive, so that the high message is continuously offered by participant 2 and hence the high input of participant 1 is indeed guaranteed to occur (Example 6.4 is the recursive version of this example).

[2]Recall that the level of an expression value is the join of the levels of the variables and values occurring in the expression.

Let us turn now to the definition of security. Security is a behavioural property, based on a notion of observation of processes. In our calculus, what is observable of processes is the way they transform the **Q**-sets at each step. Moreover, for defining security one uses a set of observers, one for each downward-closed set of security levels $\mathscr{L}$ (the intuition being that an observer who can see messages of level ℓ should also be able to see all messages of level ℓ' lower than ℓ). For every $\mathscr{L}$, the $\mathscr{L}$-observer is only able to see **Q**-set messages whose level belongs to $\mathscr{L}$ (in case of a declassified value $v^{\ell\downarrow\ell'}$, the observed level is the lowest one). Let us call these messages $\mathscr{L}$-messages, and the complementary ones $\overline{\mathscr{L}}$-messages. Hence, two **Q**-sets containing the same $\mathscr{L}$-messages will be indistinguishable for the $\mathscr{L}$-observer. This will be formalised by a notion of $\mathscr{L}$-equality $=_{\mathscr{L}}$ on **Q**-sets. Based on $=_{\mathscr{L}}$, a notion of $\mathscr{L}$-bisimulation $\simeq_{\mathscr{L}}$ on processes will then formalise indistinguishability of processes by the $\mathscr{L}$-observer: two processes will be $\mathscr{L}$-bisimilar if they transform $\mathscr{L}$-equal **Q**-sets into $\mathscr{L}$-equal **Q**-sets. Then, a process will be secure for the $\mathscr{L}$-observer if it is $\mathscr{L}$-bisimilar to itself when run with $\mathscr{L}$-equal **Q**-sets, namely if it preserves at each step the $\mathscr{L}$-equality of **Q**-sets. Intuitively, this means that the process cannot exhibit different $\mathscr{L}$-behaviours when fed with equal $\mathscr{L}$-messages but different $\overline{\mathscr{L}}$-messages (in fact, as we shall see the situation is slightly more complex since **Q**-sets are not simply multisets of messages, but they have some structure). Finally, a process is secure if it is secure for every $\mathscr{L}$-observer.

In the following, we shall always use $\mathscr{L}$ to denote a downward-closed set of levels of $\mathscr{S}$.

Formally, a queue $s : h$ is $\mathscr{L}$-observable if it contains some $\mathscr{L}$-message, that is, some message with level in $\mathscr{L}$. Then two **Q**-sets are *$\mathscr{L}$-equal* if their $\mathscr{L}$-observable queues have the same names and contain the same $\mathscr{L}$-messages. This equality is based on the following $\mathscr{L}$-projection operation on **Q**-sets, which discards all messages whose level is not in $\mathscr{L}$.

Definition 5.1. *Let the functions $lev_\uparrow$ and $lev_\downarrow$ be defined by:*

$$lev_\uparrow(v^{\ell\downarrow\ell'}) = \ell \qquad lev_\downarrow(v^{\ell\downarrow\ell'}) = \ell' \qquad lev_\uparrow(s[\mathtt{p}]^\ell) = lev_\uparrow(\lambda^\ell) = \ell = lev_\downarrow(s[\mathtt{p}]^\ell) = lev_\downarrow(\lambda^\ell).$$

Definition 5.2. *The projection operation $\Downarrow \mathscr{L}$ is defined inductively on messages, queues and **Q**-sets as follows:*

$$(\mathtt{p},\Pi,\vartheta) \Downarrow \mathscr{L} = \begin{cases} (\mathtt{p},\Pi,\vartheta) & \text{if } lev_\downarrow(\vartheta) \in \mathscr{L}, \\ \varepsilon & \text{otherwise.} \end{cases}$$

$$\varepsilon \Downarrow \mathscr{L} = \varepsilon \qquad (m \cdot h) \Downarrow \mathscr{L} = m \Downarrow \mathscr{L} \cdot h \Downarrow \mathscr{L}$$

$$\emptyset \Downarrow \mathscr{L} = \emptyset \qquad (H \cup \{s : h\}) \Downarrow \mathscr{L} = \begin{cases} H \Downarrow \mathscr{L} \cup \{s : h \Downarrow \mathscr{L}\} & \text{if } h \Downarrow \mathscr{L} \neq \varepsilon, \\ H \Downarrow \mathscr{L} & \text{otherwise.} \end{cases}$$

Notice that the projection operation $\Downarrow \mathscr{L}$ discards a message $(\mathtt{p},\Pi,v^{\ell\downarrow\ell'})$ only if $\ell' \notin \mathscr{L}$. This implies that the message containing the CreditCard (declassified to $\bot$) in the `C`, `S`, `B` example is present in all projections.

Definition 5.3 ($\mathscr{L}$-Equality of Q-sets).

*Two **Q**-sets H and K are $\mathscr{L}$-*equal*, written $H =_{\mathscr{L}} K$, if $H \Downarrow \mathscr{L} = K \Downarrow \mathscr{L}$.*

When $\mathscr{L} = \{\bot\}$, we shall write $=_\bot$ instead of $=_{\{\bot\}}$ (and similarly for $\bot$-observer, $\bot$-equality, $\bot$-security).

We proceed now to introduce our notion of $\mathscr{L}$-bisimulation. The idea is to test processes by running them in conjunction with $\mathscr{L}$-equal **Q**-sets. However, when combining processes with **Q**-sets we cannot allow arbitrary queues, otherwise we could rule out processes which are intuitively secure. Indeed, what could be safer than a process which just wants to write a bottom value in a queue or read a bottom value from a queue? However, even these simple processes turn out to be insecure without conditions on **Q**-sets.

For example, the process $P = s[1]!\langle 2,\mathsf{true}^\bot\rangle.\mathbf{0}$ does not react in the same way when combined with the two **Q**-sets $\{s : \varepsilon\} =_\bot \emptyset$. Indeed, while P reduces when combined with $\{s : \varepsilon\}$, it is stuck when combined with $\emptyset$. Formally:

$$< P \,,\, \{s : \varepsilon\} > \longrightarrow < \mathbf{0} \,,\, \{s : (1,2,\mathsf{true}^\bot)\} > \qquad\qquad < P \,,\, \emptyset > \not\longrightarrow$$

Here the $\bot$-observer would detect an insecurity by looking at the resulting **Q**-sets $\{s : (1,2,\mathsf{true}^\bot)\} \neq_\bot \emptyset$.

To get around this problem, we require that the **Q**-sets contain enough queues to enable all outputs of the process to reduce. For this it suffices that in a configuration $< P \,,\, H >$, every session name that occurs free in P has a corresponding queue in H. Let $\mathrm{fsn}(P)$ denote the set of free session names in P.

Definition 5.4. *A configuration $\langle P,H\rangle$ is* saturated *if all free session names in P have corresponding queues in H, i.e. $s \in \mathrm{fsn}(P)$ implies $s \in \mathrm{qn}(H)$.*

It is easy to check that saturation is preserved by reduction, since only rule [Link] creates new sessions, always together with the corresponding queues. In particular, starting from a closed user process and the empty **Q**-set, we always obtain saturated configurations. This is formally established by the following definitions and lemma.

Definition 5.5 (Initial configurations).

A configuration $\langle P, H\rangle$ *is* initial *if* P *is a closed user process and* $H = \emptyset$.

Definition 5.6 (Reachable configurations).

A configuration C *is* reachable *if there is an initial configuration* C_0 *such that* $C_0 \longrightarrow^* C$.

Lemma 1. *A reachable configuration is saturated.*

A dual phenomenon occurs with inputs. Consider the process $Q = s[2]?(1,x^{\perp}).\mathbf{0}$ with the **Q**-sets $H_1 =_{\perp} H_2$:

$$H_1 = \{s : (1,2,\mathsf{true}^{\perp})\}, \qquad H_2 = \{s : (1,2,\mathsf{true}^{\top})\cdot(1,2,\mathsf{true}^{\perp})\}$$

Here we have $< Q\ ,\ H_1 > \longrightarrow < \mathbf{0}\ ,\ \{s : \varepsilon\} >$, while $< Q\ ,\ H_2 > \not\longrightarrow$. Again, the $\perp$-observer would detect a leak since $\{s : \varepsilon\} \neq_{\perp} H_2$. What happens here is that the top level message in H_2 is transparent for the $\perp$-equality, but it prevents the process from reading the subsequent bottom level message. Note that the problem comes from the fact that both messages have the same receiver and the same sender, since if instead of H_2 we took $H_3 = \{s : (3,2,\mathsf{true}^{\top})\cdot(1,2,\mathsf{true}^{\perp})\}$, then we would have a matching transition $< Q\ ,\ H_3 > \longrightarrow < \mathbf{0}\ ,\ \{s : (3,2,\mathsf{true}^{\top})\} >$, thanks to the structural equivalence on queues given in Table 4.

Our way out of this problem is to ask that queues are monotone, namely that the security levels of messages with the same sender and common receivers can only stay equal or increase along a queue. Intuitively, this means that an attacker can only experiment on a process by providing or not high messages, as well as changing their content, but cannot affect the availability of low messages. In this way we allow the **Q**-sets H_1 and H_3 in the above example, but forbid H_2. More formally:

Definition 5.7. *A queue* h *is* monotone *if* $lev_{\downarrow}(\vartheta_1) \leq lev_{\downarrow}(\vartheta_2)$ *whenever the message* $(\mathtt{p}, \Pi_1, \vartheta_1)$ *precedes the message* $(\mathtt{p}, \Pi_2, \vartheta_2)$ *in* h *and* $\Pi_1 \cap \Pi_2 \neq \emptyset$. *A* **Q***-set is* monotone *if it contains only monotone queues.*

Unlike saturation, monotonicity is not preserved by reduction, since for example

$$< s[1]!\langle 2,\mathsf{true}^{\top}\rangle.s[1]!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}\ ,\ \{s : \varepsilon\} > \longrightarrow^* < \mathbf{0}\ ,\ \{s : (1,2,\mathsf{true}^{\top})\cdot(1,2,\mathsf{true}^{\perp})\} >$$

This could not be avoided by starting from initial configurations only, since the above process occurs as a subprocess in a configuration that is reachable from the initial configuration:

$$< \bar{a}^{\perp}[2] \mid a^{\perp}[1](\alpha).\alpha!\langle 2,\mathsf{true}^{\top}\rangle.\alpha!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0} \mid a^{\perp}[2](\beta).\beta?(1,x^{\top}).\beta?(1,y^{\perp}).\mathbf{0}\ ,\ \emptyset >$$

Notice that all the presented processes are typable in standard session type systems, for example in that of [2]. All these processes will also be typable in the type system we will define in Section 7, except for $a^{\perp}[2](\beta).\beta?(1,x^{\top}).\beta?(1,y^{\perp}).\mathbf{0}$, since a high input will not be allowed to guard a low action. In particular, the process $s[1]!\langle 2,\mathsf{true}^{\top}\rangle.s[1]!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}$ will be typable, in spite of the fact that it generates a non monotone queue. In Theorem 14 we will show that our type system assures monotonicity of **Q**-sets that are generated by well-typed initial configurations.

We are now ready for defining our bisimulation. A relation $\mathscr{R}$ on processes is a $\mathscr{L}$*-bisimulation* if, whenever two related processes are coupled with $\mathscr{L}$-equal monotone **Q**-sets yielding saturated configurations, then the reduction relation preserves both the relation $\mathscr{R}$ on processes and the $\mathscr{L}$-equality of **Q**-sets:

Definition 5.8 ($\mathscr{L}$-Bisimulation on processes).

A symmetric relation $\mathscr{R} \subseteq (\mathscr{P}r \times \mathscr{P}r)$ *is a* $\mathscr{L}$-bisimulation *if* $P_1 \mathscr{R} P_2$ *implies, for any pair of monotone* **Q***-sets* H_1 *and* H_2 *such that* $H_1 =_{\mathscr{L}} H_2$ *and each* $< P_i\ ,\ H_i >$ *is saturated:*

If $< P_1\ ,\ H_1 > \longrightarrow (\nu\tilde{r}) < P_1'\ ,\ H_1' >$, *then either* $H_1' =_{\mathscr{L}} H_2$ *and* $P_1' \mathscr{R} P_2$, *or there exist* P_2', H_2' *such that* $< P_2\ ,\ H_2 > \longrightarrow^* (\nu\tilde{r}) < P_2'\ ,\ H_2' >$, *where* $H_1' =_{\mathscr{L}} H_2'$ *and* $P_1' \mathscr{R} P_2'$.

Processes P_1, P_2 *are* $\mathscr{L}$-bisimilar, $P_1 \simeq_{\mathscr{L}} P_2$, *if* $P_1 \mathscr{R} P_2$ *for some* $\mathscr{L}$*-bisimulation* $\mathscr{R}$.

Note that $\tilde{r}$ may either be the empty string or a single name, since it appears after a one-step transition. If it is a name, it may either be a service name a^{ℓ} (extrusion of a private service) or a fresh session name s (opening of a new session). In the latter case, s cannot occur in P_2 and H_2 by Barendregt convention.

Intuitively, a transition that adds or removes a message with level in $\mathscr{L}$ must be simulated in one or more steps, producing the same effect on the **Q**-set, whereas a transition that only affects messages with level not in $\mathscr{L}$ may be simulated by inaction.

The notions of $\mathscr{L}$-security and security are now defined in the standard way:

Definition 5.9 ($\mathscr{L}$-Security). *A process P is* $\mathscr{L}$-secure *if $P \simeq_{\mathscr{L}} P$.*

Notice that each process is $\mathscr{S}$-secure, since for $\mathscr{L} = \mathscr{S}$ the $\mathscr{L}$-equality of saturated **Q**-sets reduces to syntactic identity, and hence $\simeq_{\mathscr{S}}$ reduces to ordinary bisimilarity, which is a reflexive relation.

The soundness of our definition of $\mathscr{L}$-security comes from the observation that for an arbitrary process P the *monotone* **Q**-set $H_P = \{s : \varepsilon \mid s \in \mathrm{fsn}(P)\}$ gives a *saturated* configuration $< P \, , \, H_P >$.

Definition 5.10 (Security). *A process P is* secure *if it is $\mathscr{L}$-secure for every $\mathscr{L}$.*

The necessity of considering all down-closed sets of levels included in $\mathscr{S}$ is justified by the following example. Let $\mathscr{S} = \{\bot, \ell, \top\}$ where $\bot \leq \ell \leq \top$ and

$$P = s[2]?(1, x^{\top}).s[2]!^{\ell}\langle 1, \mathsf{true}^{\ell}\rangle.\mathbf{0}$$

The process P is $\bot$-secure (and $\mathscr{S}$-secure), but it is not $\{\bot, \ell\}$-secure, as can be seen by taking the two $\{\bot, \ell\}$-equal **Q**-sets $H_1 = \{s : (1, 2, \mathsf{true}^{\top})\}$ and $H_2 = \{s : \varepsilon\}$.

6. Examples of Information Flow Security in Sessions

In this section we illustrate the various kinds of information flow that can occur in our calculus, by means of simple examples. Since we aim at justifying the introduction of security levels in the syntax, other than on basic values and participants, we *initially omit levels on all other objects.* Then, as we go along examining examples, we will find out that other objects need to carry security levels. These levels will be used in the type system to reject processes that are innocuous in isolation but may be dangerous in some contexts. For the sake of simplicity, we use here just two security levels $\bot$ and $\top$ (also called low and high).

6.1. Insecurity of high input followed by low action

Consider the process P (which coincides with that of Example (*) at page 8, assuming $e = \mathsf{true}^{\bot}$), together with the **Q**-sets H_1 and H_2, where $H_1 =_{\bot} H_2$:

$$P = s[1]?(2, x^{\top}).s[1]!\langle 3, \mathsf{true}^{\bot}\rangle.\mathbf{0}, \quad H_1 = \{s : (2, 1, \mathsf{true}^{\top})\} \qquad H_2 = \{s : \varepsilon\}$$

Here we have $< P \, , \, H_1 > \longrightarrow < s[1]!\langle 3, \mathsf{true}^{\bot}\rangle.\mathbf{0} \, , \, \{s : \varepsilon\} > = < P_1 \, , \, H_1' >$, while $< P \, , \, H_2 > \not\longrightarrow$. Since $H_1' = \{s : \varepsilon\} = H_2$, we can proceed with $P_1 = s[1]!\langle 3, \mathsf{true}^{\bot}\rangle.\mathbf{0}$ and $P_2 = P$. Take now $K_1 = K_2 = \{s : \varepsilon\}$. Then $< P_1 \, , \, K_1 > \longrightarrow < \mathbf{0} \, , \, \{s : (1, 3, \mathsf{true}^{\bot})\} >$, while $< P_2 \, , \, K_2 > \not\longrightarrow$. Since $\{s : (1, 3, \mathsf{true}^{\bot})\} \neq_{\bot} \{s : \varepsilon\} = K_2$, P is not $\bot$-secure.

With a similar argument we could show that the process $s[1]?(2, x^{\top}).s[1]?(3, y^{\bot}).\mathbf{0}$ is not $\bot$-secure.

6.2. Security of declassified high-to-low input followed by low action

Consider the following process Q, which differs from process P in Example 6.1 in that the input action declassifies the received value from high to low:

$$Q = s[1]?^{\top}(2, x^{\bot}).s[1]!\langle 3, \mathsf{true}^{\bot}\rangle.\mathbf{0},$$

Let $H_1 = \{s : (2, 1, v^{\top\downarrow\bot})\}$. Note that $H_1 \neq_{\bot} \{s : \varepsilon\}$. Indeed, for H_2 to be such that $H_1 =_{\bot} H_2$, it must be the case that $H_2 \equiv \{s : (2, 1, v^{\top\downarrow\bot}) \cdot h\} \cup H$, where $\{s : h\} \cup H =_{\bot} \{s : \varepsilon\}$. Then, it is easy to see that $< Q \, , \, H_1 > \longrightarrow < Q' \, , \, H_1' >$ implies $< Q \, , \, H_2 > \longrightarrow < Q' \, , \, H_2' >$, with $H_1' =_{\bot} H_2'$. This is enough to conclude that Q is secure.

Interestingly, an insecure component may be "sanitised" by its context, so that the insecurity is not detectable in the overall process. Clearly, in case the insecure component is wrapped into a deadlocked context such as $(\nu b)b[1](\beta_1).[\,]$, the insecurity disappears simply because the component is not executed. However, the curing context could also be a partner of the component, as shown by the next example.

6.3. Insecurity sanitised by recursion and parallel context

Let us consider the unfolded recursive version R of the process P of Example 6.1, in parallel with a process $\overline{R}$ that continuously emits a matching message.

$$\begin{array}{rcl} R & = & \mathsf{def}\ X(x^{\perp},\alpha) = \alpha?(2,y^{\top}).\alpha!\langle 3,x^{\perp}\rangle.X(x^{\perp},\alpha)\ \mathsf{in}\ s[1]?(2,y^{\top}).s[1]!\langle 3,\mathsf{true}^{\perp}\rangle.X\langle \mathsf{true}^{\perp},s[1]\rangle \\ \overline{R} & = & \mathsf{def}\ Y(z^{\top},\beta) = \beta!\langle 1,z^{\top}\rangle.Y(z^{\top},\beta)\ \mathsf{in}\ s[2]!\langle 1,\mathsf{true}^{\top}\rangle.Y\langle \mathsf{true}^{\top},s[2]\rangle \end{array}$$

Take the two $\perp$-equal $\mathbf{Q}$-sets $H_1 = \{s : (2,1,\mathsf{true}^{\top})\} =_{\perp} \{s : \varepsilon\} = H_2$. Then the move

$$< R \mid \overline{R}\ ,\ H_1 > \longrightarrow < R' \mid \overline{R}\ ,\ \{s : \varepsilon\} >$$

where $R' = \mathsf{def}\ X(x^{\perp},\alpha) = \alpha?(2,y^{\top}).\alpha!\langle 3,x^{\perp}\rangle.X(x^{\perp},\alpha)\ \mathsf{in}\ s[1]!\langle 3,\mathsf{true}^{\perp}\rangle.X\langle \mathsf{true}^{\perp},s[1]\rangle$ can be simulated by

$$< R \mid \overline{R}\ ,\ H_2 > \longrightarrow < R \mid \overline{R}'\ ,\ \{s : (2,1,\mathsf{true}^{\top})\} > \longrightarrow^{*} < R' \mid \overline{R}\ ,\ \{s : \varepsilon\} >$$

where $\overline{R}'$ is the folded version of $\overline{R}$. Note that it would make no difference if we replaced $H_1 = \{s : (2,1,\mathsf{true}^{\top})\}$ with $H_3 = \{s : (2,1,\mathsf{false}^{\top})\}$.

6.4. Insecurity of high conditional with different low branches

Consider the following process $\widehat{Q}$, which is the recursive unfolded version of process Q in Example (**) of page 8:

$$\begin{array}{rcl} \widehat{Q} & = & Q_1 \mid Q_2 \\ Q_1 & = & \mathsf{def}\ X(x^{\perp},\alpha) = Q'\ \mathsf{in}\ Q'\{\mathsf{true}^{\perp}/x^{\perp}\}\{s[1]/\alpha\} \\ Q_2 & = & \mathsf{def}\ Y(y^{\top},\beta) = \beta!\langle 1,y^{\top}\rangle.Y(y^{\top},\beta)\ \mathsf{in}\ s[2]!\langle 1,\mathsf{true}^{\top}\rangle.Y\langle \mathsf{true}^{\top},s[2]\rangle \end{array}$$

where $Q' = \alpha?(2,z^{\top}).\mathsf{if}\ z^{\top}\ \mathsf{then}\ \alpha!\langle 3,x^{\perp}\rangle.X\langle x^{\perp},\alpha\rangle\ \mathsf{else}\ \alpha!\langle 3,\mathsf{not}\,x^{\perp}\rangle.X\langle x^{\perp},\alpha\rangle$. Take the $\mathbf{Q}$-sets $H_1 = \{s : (2,1,\mathsf{false}^{\top})\}$ and $H_2 = \{s : \varepsilon\}$. We have $H_1 =_{\perp} H_2$. Now, if $< \widehat{Q}\ ,\ H_1 >$ starts by reading the message $(2,1,\mathsf{false}^{\top})$ from H_1, then $< \widehat{Q}\ ,\ H_2 >$ could try to reply by letting first Q_2 output the message $(2,1,\mathsf{true}^{\top})$ and then Q_1 read this message. In both cases we would be left with the $\mathbf{Q}$-set $H_1' = H_2' = \{s : \varepsilon\}$. However this would result in the replacement of different values for $z^{\top}$ in the conditional, leading in the next steps to the output of two different low values from participant 1 to participant 3, yielding the non $\perp$-equal $\mathbf{Q}$-sets $H_1'' = \{s : (1,3,\mathsf{false}^{\perp})\}$ and $H_2'' = \{s : (1,3,\mathsf{true}^{\perp})\}$.

The next series of examples justifies the introduction of security levels on services, labels and delegated channels, by showing that they are necessary in the type system to rule out some indirect information flows.

6.5. Need for levels on services

Consider the following process:

$$s[2]?(1,x^{\top}).\bar{b}[2] \mid\ b[1](\beta_1).\beta_1!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0} \mid\ b[2](\beta_2).\beta_2?(1,y^{\perp}).\mathbf{0}$$

This process is insecure since if there is no $\top$-message in the $\mathbf{Q}$-set the process is blocked, hence it has a null $\perp$-behaviour, while if there is a $\top$-message in the $\mathbf{Q}$-set the process subsequently exhibits a $\perp$-action. To be able to rule out this kind of leak in our type system, we annotate service names with security levels which in a typable process will act as a lower bound for all the actions executed by the service. Then service b will have to be of level $\top$ since it is guarded by an input of a $\top$-value, and hence it will not be allowed to perform the exchange of the value $\mathsf{true}^{\perp}$.

The communication of a service name only makes sense if the name is restricted, and it is used for allowing a set of participants to start a session on that private name. The above example shows that the service name must have the same security level in the initiator and in all participants, otherwise the session could not start. This is why we do not allow service declassification.

6.6. Need for levels on selection and branching

Consider the process:

$$s[2]?(1,x^{\top}).s[2] \oplus \langle 3,\lambda\rangle.\mathbf{0} \mid\ s[3]\&(2,\{\lambda : s[3]!\langle 4,\mathsf{true}^{\perp}\rangle.\mathbf{0}\}) \mid\ s[4]?(3,y^{\perp}).\mathbf{0}$$

As in the previous example, this process is insecure because if there is no $\top$-message in the $\mathbf{Q}$-set the process is blocked, while if there is a $\top$-message in the $\mathbf{Q}$-set the process subsequently exhibits a $\perp$-action. To prevent this kind of leak, the selection and branching operators will be annotated with a security level which acts as a lower bound for all actions executed in the various branches.

6.7. Need for levels on delegated channels

Consider the following process:

$$s[2]?(1,x^{\top}).s[2]!\langle\langle 3,s'[1]\rangle\rangle.\mathbf{0} \mid\ s[3]?((2,\eta)).\eta!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0} \mid\ s'[2]?(1,y^{\bot}).\mathbf{0}$$

This process is insecure since if there is no $\top$-message in the $\mathbf{Q}$-set the process is blocked, while if there is a $\top$-message in the $\mathbf{Q}$-set the process subsequently exhibits a $\bot$-action. This shows that delegation send and receive should also carry a level, which will act as a lower bound for all actions executed in the receiving participant after the delegation.

6.8. Need for levels in queue messages

So far, we have identified which objects of the calculus need security levels, namely: basic values, service names, and the operators of selection, branching and delegation. Let us now discuss how levels are recorded into queue messages.

Values are recorded in the queues with both their original level and their declassified level. The reason for recording also the original level is access control: the semantics does not allow a low input process to fetch a high value declassified to low. More formally, a value $v^{\top\downarrow\bot}$ in the queue can only be read by a process $s[\mathtt{q}]?^{\top}(\mathtt{p},x^{\bot}).P$. Concerning service names a^{ℓ}, the level ℓ guarantees that the session initiator and all the participants get started in a context of level $\ell' \leq \ell$ (see Example 6.5). Once the session is established, the name a^{ℓ} disappears and it is its global type (cf next section) that will ensure that all participants perform actions of levels greater than or equal to ℓ. As for the operators of branching/selection and delegation, they disappear after the reduction and their level is recorded respectively into labels and delegated channels within queue messages. This is essential to ensure that the sender and receiver have matching security levels, since in this case the communication is asynchronous and occurs in two steps. In conclusion, queue messages need to be of the form $(\mathtt{p},\Pi,\vartheta)$, where ϑ is of one of $v^{\ell\downarrow\ell'}$, $s[\mathtt{p}]^{\ell}$, or λ^{ℓ}.

To conclude this section, note that while $P_1 = s[1]?(2,x^{\top}).s[1]?(2,y^{\bot}).\mathbf{0}$ and $P_2 = s[1]?(2,x^{\top}).s[1]!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0}$ are insecure, their dual processes $Q_1 = s[2]!\langle 1,\mathsf{true}^{\top}\rangle.s[2]!\langle 1,\mathsf{true}^{\bot}\rangle.\mathbf{0}$ and $Q_2 = s[2]!\langle 1,\mathsf{true}^{\top}\rangle.s[2]?(1,\mathsf{true}^{\bot}).\mathbf{0}$ are secure. Intuitively, this is because output is not blocking.

Finally, we let the reader verify that the process of the `C`, `S`, `B` example in Section 4.1 is secure.

7. Type system

In this section we present our type system for secure sessions and state its properties. Just like process syntax, types will contain security levels.

7.1. Safety Global Types

A *safety global type* is a pair $\langle \mathsf{L},G\rangle^{\ell}$, decorated with a security level ℓ, describing a service where:

- $\mathsf{L} : \{1,\ldots,n\} \to \mathscr{S}$ is a *safety mapping* from participants to security levels;
- G is a *global type*, describing the whole conversation scenario of an n-ary service;
- ℓ is the meet of all levels appearing in G, denoted by $\mathsf{M}(G)$.

The grammar of global types is:

$$\begin{array}{llcl} \text{Global} & G & ::= & \mathtt{p} \to \Pi : \langle U\rangle.G \\ & & \mid & \mathtt{p} \to \Pi : \{\lambda_i : G_i\}_{i\in I}^{\ell} \\ & & \mid & \mathtt{p}\,\Pi\delta.G \\ & & \mid & \mu\mathbf{t}.G \mid \mathbf{t} \mid \mathsf{end} \end{array} \qquad \begin{array}{llcl} \text{Exchange} & U & ::= & S^{\ell\downarrow\ell'} \mid T \mid \langle \mathsf{L},G\rangle^{\ell} \\ \text{Sorts} & S & ::= & \mathsf{bool} \mid \ldots \end{array}$$

The type $\mathtt{p} \to \Pi : \langle U\rangle.G$ says that participant $\mathtt{p}$ multicasts a message of type U to all participants in Π and then the interactions described in G take place.

Exchange types U may be *sort* types $S^{\ell\downarrow\ell'}$ for values (base types decorated with a declassification $\ell \downarrow \ell'$), *session types* T for channels (defined below), or safety global types for services. If $U = T$, then Π is a singleton. We use S^{ℓ} as short for $S^{\ell\downarrow\ell}$, called a *trivial declassification*.

Type $\mathtt{p} \to \Pi : \{\lambda_i : G_i\}_{i\in I}^{\ell}$, where $\ell \leq \bigsqcap_{i\in I} \mathsf{M}(G_i)$, says that participant $\mathtt{p}$ multicasts one of the labels λ_i to the participants in Π. If λ_j is sent, interactions described in G_j take place.

Type $\mathtt{p}\,\Rsh\delta.G$ says that the role of $\mathtt{p}$ is delegated to another participant; this construct does not appear in the original global types of [16]. It is needed here to "mark" the delegated part of the type, which is discharged when calculating its join (see below).

Type $\mu\mathbf{t}.G$ is a recursive type, where the type variable $\mathbf{t}$ is guarded in the standard way. In the grammar of exchange types, we suppose that G does not contain free type variables. Type end represents the termination of a session.

The meet of global types is needed for information flow control, and it takes into account the lower level of declassified exchanged values, and the level of all other constructs, including delegation. More precisely the meet of global types is defined as follows, using the meet of session types given in next section.

$$\begin{array}{ll}
\mathsf{M}(\mathtt{p}\to\Pi:\langle S^{\ell\downarrow\ell'}\rangle.G)=\ell'\sqcap\mathsf{M}(G) & \mathsf{M}(\mathtt{p}\to\Pi:\langle\langle\mathsf{L},G\rangle^{\ell}\rangle.G)=\ell\sqcap\mathsf{M}(G)\\
\mathsf{M}(\mathtt{p}\to\mathtt{q}:\langle T\rangle.G)=\mathsf{M}(T)\sqcap\mathsf{M}(G) & \mathsf{M}(\mathtt{p}\to\Pi:\{\lambda_i:G_i\}^{\ell}_{i\in I})=\ell\\
\mathsf{M}(\mathtt{p}\,\Rsh\delta.G)=\mathsf{M}(G) & \mathsf{M}(\mathbf{t})=\top\\
\mathsf{M}(\mu\mathbf{t}.G)=\mathsf{M}(G) & \mathsf{M}(\mathsf{end})=\top
\end{array}$$

The condition $\ell\le\bigsqcap_{i\in I}\mathsf{M}(G_i)$ (see above) justifies the absence of $\mathsf{M}(G_i)$ in the definition of $\mathsf{M}(G)$ for the case of label exchanges.

7.2. Session Types

While global types represent the whole session protocol, *session types* correspond to the communication actions, representing each participant's contribution to the session.

Session	T	::=					
			$!\langle\Pi,S^{\ell\downarrow\ell'}\rangle;T$	*send*	$?(\mathtt{p},S^{\ell\downarrow\ell'});T$	*receive*
		$\mid$	$!\langle\Pi,\langle\mathsf{L},G\rangle^{\ell}\rangle;T$	*servsend*	$?(\Pi,\langle\mathsf{L},G\rangle^{\ell});T$	*servreceive*
		$\mid$	$!^{\ell}\langle\mathtt{q},T\rangle;T'$	*delsend*	$?^{\ell}(\mathtt{p},T);T'$	*delreceive*
		$\mid$	$\oplus^{\ell}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle$	*selection*	$\&^{\ell}(\mathtt{p},\{\lambda_i:T_i\}_{i\in I})$	*branching*
		$\mid$	$\mu\mathbf{t}.T$	*recursive*	$\mathbf{t}$	*variable*
		$\mid$	$\Rsh\delta;T$	*delegation*	end	*end*

The *send* and *servsend* types ($!\langle\Pi,S^{\ell\downarrow\ell'}\rangle;T$ and $!\langle\Pi,\langle\mathsf{L},G\rangle^{\ell}\rangle;T$) express the sending to all participants in Π of a value of type S of level ℓ, declassified to ℓ' and of a service of type $\langle\mathsf{L},G\rangle^{\ell}$ of level ℓ respectively, followed by the communications described in T. The two different types for sending are necessary since services cannot be declassified. The *delsend* type $!^{\ell}\langle\mathtt{q},T\rangle;T'$says that a channel of type T is sent to participant $\mathtt{q}$, and then the protocol specified by T' takes place. The *selection* type $\oplus^{\ell}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle$, represents the transmission to all participants in Π of a label λ_j in $\{\lambda_i\mid i\in I\}$, followed by the communications described in T_j. The *delegation* type $\Rsh\delta;T$, says that the communications described in T will be delegated to another agent. The *receive*, *servreceive*, *delreceive* and *branching* types are dual to the *send*, *servsend delsend*, and *selection* ones. In all cases, the need for the security level ℓ is motivated by one of the examples in Section 6. Recursive types are considered modulo fold/unfold.

We say that $!\langle\Pi,S^{\ell\downarrow\ell'}\rangle$, $!\langle\Pi,\langle\mathsf{L},G\rangle^{\ell}\rangle$, $!^{\ell}\langle\mathtt{q},T\rangle$, $?(\mathtt{p},S^{\ell\downarrow\ell'})$, $?(\mathtt{p},\langle\mathsf{L},G\rangle^{\ell})$, $?^{\ell}(\mathtt{p},T)$ are *type prefixes* and we use τ to range over them.

A type prefix τ *occurs* in a session type T if one of the following conditions holds:

- $T=\tau;T'$;
- $T=\tau';T'$ or $T=\mu\mathbf{t}.T'$ or $T=\Rsh\delta;T'$ and τ occurs in T';
- $T=\oplus^{\ell}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle$ or $T=\&^{\ell}(\mathtt{p},\{\lambda_i:T_i\}_{i\in I})$ and τ occurs in T_i for some $i\in I$.

A type prefix τ *precedes* a type prefix τ' in a session type T if one of the following conditions holds:

- $T=\tau;T'$ and τ' occurs in T';
- $T=\tau'';T'$ or $T=\mu\mathbf{t}.T'$ or $T=\Rsh\delta;T'$ and τ precedes τ' in T';
- $T=\oplus^{\ell}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle$ or $T=\&^{\ell}(\mathtt{p},\{\lambda_i:T_i\}_{i\in I})$ and τ precedes τ' in T_i for some $i\in I$.

As for $\mathsf{M}(G)$, we denote by $\mathsf{M}(T)$ the meet of all security levels appearing in T.

$$\begin{array}{ll}
\mathsf{M}(\,!\langle\Pi,S^{\ell\downarrow\ell'}\rangle;T)=\ell'\sqcap\mathsf{M}(T) & \mathsf{M}(?(\mathtt{p},S^{\ell\downarrow\ell'});T)=\ell'\\
\mathsf{M}(\,!\langle\Pi,\langle\mathsf{L},G\rangle^{\ell}\rangle;T)=\ell\sqcap\mathsf{M}(T) & \mathsf{M}(?(\mathtt{p},\langle\mathsf{L},G\rangle^{\ell});T)=\ell\\
\mathsf{M}(\,!^{\ell}\langle\mathtt{q},T\rangle;T')=\ell\sqcap\mathsf{M}(T') & \mathsf{M}(?^{\ell}(\mathtt{p},T);T')=\ell\\
\mathsf{M}(\oplus^{\ell}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle)=\ell & \mathsf{M}(\&^{\ell}(\mathtt{p},\{\lambda_i:T_i\}_{i\in I}))=\ell\\
\mathsf{M}(\mathbf{t})=\top & \mathsf{M}(\mu\mathbf{t}.T)=\mathsf{M}(T)\\
\mathsf{M}(\Rsh\delta;T)=\mathsf{M}(T) & \mathsf{M}(\mathsf{end})=\top
\end{array}$$

The type system will assure some relations between levels appearing in types and meets of continuations that justify the definition of $\mathsf{M}(T)$ in the cases of inputs, output of a channel, selection and branching, see Proposition 2.

7.3. Projections

The relation between global types and session types is formalised by the notion of projection [16]. The *projection of G onto* $\mathtt{q}$, denoted $(G \upharpoonright \mathtt{q})$, gives participant $\mathtt{q}$'s view of the protocol described by G.

Definition 7.1. *The* projection of G onto $\mathtt{q}$ *(written $G \upharpoonright \mathtt{q}$) is defined by induction on G:*

$$(\mathtt{p} \to \Pi : \langle S^{\ell\downarrow\ell'}\rangle.G') \upharpoonright \mathtt{q} = \begin{cases} !\langle \Pi, S^{\ell\downarrow\ell'}\rangle; (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ ?(\mathtt{p}, S^{\ell\downarrow\ell'}); (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \in \Pi, \\ G' \upharpoonright \mathtt{q} & \text{otherwise}. \end{cases}$$

$$(\mathtt{p} \to \Pi : \langle \langle \mathsf{L}, G\rangle^{\ell}\rangle.G') \upharpoonright \mathtt{q} = \begin{cases} !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle; (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ ?(\mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell}); (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \in \Pi, \\ G' \upharpoonright \mathtt{q} & \text{otherwise}. \end{cases}$$

$$(\mathtt{p} \to \mathtt{p}' : \langle T\rangle.G') \upharpoonright \mathtt{q} = \begin{cases} !^{\ell}\langle \mathtt{p}', T\rangle; (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ ?^{\ell}(\mathtt{p}, T); (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}', \\ G' \upharpoonright \mathtt{q} & \text{otherwise} \end{cases}$$
$$\text{where } \ell = \mathsf{M}(T).$$

$$(\mathtt{p} \to \Pi : \{\lambda_i : G_i\}^{\ell}_{i\in I}) \upharpoonright \mathtt{q} = \begin{cases} \oplus^{\ell}(\Pi, \{\lambda_i : G_i \upharpoonright \mathtt{q}\}_{i\in I}) & \text{if } \mathtt{q} = \mathtt{p} \\ \&^{\ell}(\mathtt{p}, \{\lambda_i : G_i \upharpoonright \mathtt{q}\}_{i\in I}) & \text{if } \mathtt{q} \in \Pi \\ G_1 \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \neq \mathtt{p}, \mathtt{q} \notin \Pi \text{ and} \\ & G_i \upharpoonright \mathtt{q} = G_j \upharpoonright \mathtt{q} \text{ for all } i, j \in I. \end{cases}$$

$$\mathtt{p}\,\Rsh\delta.G' \upharpoonright \mathtt{q} = \begin{cases} \Rsh\delta; (G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} = \mathtt{p}, \\ G' \upharpoonright \mathtt{q} & \text{otherwise}. \end{cases}$$

$$(\mu \boldsymbol{t}.G') \upharpoonright \mathtt{q} = \begin{cases} \mu \boldsymbol{t}.(G' \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \text{ occurs in } G', \\ \mathsf{end} & \text{otherwise}. \end{cases} \qquad \boldsymbol{t} \upharpoonright \mathtt{q} = \boldsymbol{t} \qquad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

The mapping of the global type for message multicasting assures that the message has sort S and security level ℓ declassified to ℓ' in both the sender and the receivers. The mapping of the global type for delegation assures that the security levels of the projections be the same and equal to the meet of the session type of the delegated channel. From Example 6.7 it is clear that we could choose for the projections also a level less than this meet, but this would render projection a relation instead of a function. The condition $G_i \upharpoonright \mathtt{q} = G_j \upharpoonright \mathtt{q}$ for all $i, j \in I$ assures that the projections of all the participants not involved in the branching are identical session types.

7.4. Well-formedness of safety global types

To formulate the well-formedness condition for safety global types, we define the join $\mathsf{J}(T)$ of a session type T. Intuitively, while $\mathsf{M}(T)$ is needed for secure information flow, $\mathsf{J}(T)$ will be used for access control. Recall from Section 2 our access control policy, requiring that participants in a session only read data of level less than or equal to their own level. This motivates our (slightly non standard) definition of join: in short, $\mathsf{J}(T)$ is the join of all the security levels decorating the input constructs in T (*receive, servreceive, delreceive, branching*) and selection:

$$\begin{array}{ll} \mathsf{J}(\,!\langle \Pi, S^{\ell\downarrow\ell'}\rangle; T) = \mathsf{J}(T) & \mathsf{J}(?(\mathtt{p}, S^{\ell\downarrow\ell'}); T) = \ell \sqcup \mathsf{J}(T) \\ \mathsf{J}(\,!\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle; T) = \mathsf{J}(T) & \mathsf{J}(?(\mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell}); T) = \ell \sqcup \mathsf{J}(T) \\ \mathsf{J}(\,!^{\ell}\langle \mathtt{q}, T\rangle; T') = \mathsf{J}(T') & \mathsf{J}(?^{\ell}(\mathtt{p}, T); T') = \ell \sqcup \mathsf{J}(T') \\ \mathsf{J}(\oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I}\rangle) = \ell \sqcup \bigsqcup_{i\in I} \mathsf{J}(T_i) & \mathsf{J}(\&^{\ell}(\mathtt{p}, \{\lambda_i : T_i\}_{i\in I})) = \ell \sqcup \bigsqcup_{i\in I} \mathsf{J}(T_i) \\ \mathsf{J}(\boldsymbol{t}) = \bot & \mathsf{J}(\mu \boldsymbol{t}.T) = \mathsf{J}(T) \\ \mathsf{J}(\Rsh\delta; T) = \bot & \mathsf{J}(\mathsf{end}) = \bot \end{array}$$

Notice that in the definition of $\mathsf{J}(T)$ we cannot omit ℓ in the case of selection, because we could have $\mathsf{M}(T) \not\leq \mathsf{J}(T)$. For example $\mathsf{M}(\mathsf{end}) = \mathsf{M}(\mu \mathbf{t}.\mathbf{t}) = \top$ and $\mathsf{J}(\mathsf{end}) = \mathsf{J}(\mu \mathbf{t}.\mathbf{t}) = \bot$. Then, if all T_i are such that $\mathsf{M}(T_i) = \top$ and $\mathsf{J}(T_i) = \bot$, we have $\bigsqcup_{i\in I} \mathsf{J}(T_i) = \bot$ and $\ell = \bigsqcap_{i\in I} \mathsf{M}(T_i) = \top$ and thus $\bigsqcup_{i\in I} \mathsf{J}(T_i) = \bot \neq \top = \ell \sqcup \bigsqcup_{i\in I} \mathsf{J}(T_i)$.

The condition of well-formedness for safety global types requires that the security level of the participant $\mathtt{p}$ is greater than or equal to the join of the $\mathtt{p}$-th projection of the global type. More formally, the condition of well-formedness is the following, where $dom(\mathsf{L})$ denotes the domain of L:

$$\textit{A safety global type } \langle \mathsf{L}, G\rangle^{\ell} \textit{ is well formed if for all } \mathtt{p} \in dom(\mathsf{L}):\ \ \mathsf{L}(\mathtt{p}) \geq \mathsf{J}(G \upharpoonright \mathtt{p}).$$

Henceforth we shall only consider well-formed safety global types.

7.5. Typing rules

Typing expressions. The typing judgments for expressions are of the form:

$$\Gamma \vdash e : S^{\ell}$$

where Γ is the *standard environment* which maps variables with security levels to sort types with trivial declassification or to safety global types, service names with security levels to safety global types, and process variables to pairs of sort types with trivial declassification and session types. Formally, we define:

$$\Gamma ::= \emptyset \mid \Gamma, x^{\ell} : S^{\ell'} \mid \Gamma, x^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'} \mid \Gamma, a^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'} \mid \Gamma, X : S^{\ell}\ T$$

assuming that we can write $\Gamma, x^{\ell} : S^{\ell'}$ or $\Gamma, x^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'}$ (respectively $\Gamma, a^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'}$) only if there does not exist ℓ'' such that $x^{\ell''}$ (respectively $a^{\ell''}$) belongs to the domain of Γ, that is, Γ should not contain the same variable or service name with two different security levels. Similarly, when writing $\Gamma, X : S^{\ell}\ T$, we assume that X does not belong to the domain of Γ.

An environment Γ is well formed if $x^{\ell} : S^{\ell'} \in \Gamma$, $x^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'} \in \Gamma$, or $a^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'} \in \Gamma$ implies that $\ell' = \ell$. In the following we will only consider well-formed environments.

We type values by decorating their type with their security level, and names according to Γ:

$$\Gamma \vdash \mathsf{true}^{\ell}, \mathsf{false}^{\ell} : \mathsf{bool}^{\ell} \qquad\qquad \Gamma, u : S^{\ell} \vdash u : S^{\ell}$$

We type expressions by decorating their type with the join of the security levels of the variables and values they are built from. For example a typing rule for and is:

$$\frac{\Gamma \vdash e_1 : \mathsf{bool}^{\ell_1} \qquad \Gamma \vdash e_2 : \mathsf{bool}^{\ell_2}}{\Gamma \vdash e_1\ \mathsf{and}\ e_2 : \mathsf{bool}^{\ell_1 \sqcup \ell_2}}$$

Typing processes. The typing judgments for processes are of the form:

$$\Gamma \vdash_{\ell} P \triangleright \Delta$$

where Δ is the *process environment* which associates session types with channels:

$$\Delta ::= \emptyset \mid \Delta, c : T$$

We decorate the derivation symbol $\vdash$ with the security level ℓ inferred for the process: this level is a lower bound for the actions and communications performed in the process. The set of typing rules for processes is given in Table 8.

$\lfloor$SUBS$\rfloor$ allows the security level inferred for a process to be decreased.

$\lfloor$INACT$\rfloor$ This rule gives security level $\top$ to $\mathbf{0}$ since a terminated process cannot leak any information. The hypothesis assures that there is no further behaviour after inaction.

$\lfloor$CONC$\rfloor$ This rule allows the parallel composition of two processes P, Q to be typed if both processes are typable and their process environments have disjoint domains. The level of the resulting process is the level of both P and Q. If P, Q should have different levels ℓ_1, ℓ_2 we can give them the level $\ell_1 \sqcap \ell_2$ using rule $\lfloor$SUBS$\rfloor$.

$\lfloor$IF$\rfloor$ This rule requires that the two branches P, Q of a conditional be typed with the same process environment Δ, and with the same security level ℓ. Note that the classical requirement that the security level ℓ' of the tested expression be less than or equal to the security level ℓ of the branches is not needed here. This is because the process $\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$ can only be executed if the expression e can be evaluated, and this is the case only if all the variables occurring in e have been instantiated by previous inputs. Since these inputs guard the conditional, the join of their levels will be less than or equal to ℓ by Rule $\lfloor$RCV$\rfloor$. This means that the process $\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$ is typable whenever P and Q are typable with the same process environment Δ, since the levels of the branches can always be made equal using Rule $\lfloor$SUBS$\rfloor$.

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta \quad \ell' \leq \ell}{\Gamma \vdash_{\ell'} P \triangleright \Delta} \lfloor \textsc{SubS} \rfloor \qquad \frac{\Delta \text{ end only}}{\Gamma \vdash_\top \mathbf{0} \triangleright \Delta} \lfloor \textsc{Inact} \rfloor$$

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta \quad \Gamma \vdash_\ell Q \triangleright \Delta'}{\Gamma \vdash_\ell P \mid Q \triangleright \Delta, \Delta'} \lfloor \textsc{Conc} \rfloor \qquad \frac{\Gamma \vdash e : \mathsf{bool}^{\ell'} \quad \Gamma \vdash_\ell P \triangleright \Delta \quad \Gamma \vdash_\ell Q \triangleright \Delta}{\Gamma \vdash_\ell \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \triangleright \Delta} \lfloor \textsc{If} \rfloor$$

$$\frac{\Gamma, a^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_{\ell'} P \triangleright \Delta}{\Gamma \vdash_{\ell'} (\nu a^\ell) P \triangleright \Delta} \lfloor \textsc{NRes} \rfloor \qquad \frac{\Gamma \vdash e : S^\ell \quad \ell' = \mathsf{M}(T) \quad \Delta \text{ end only}}{\Gamma, X : S^\ell\ T \vdash_{\ell \sqcap \ell'} X\langle e, c \rangle \triangleright \Delta, c : T} \lfloor \textsc{Var} \rfloor$$

$$\frac{\Gamma, X : S^\ell\ T, x^\ell : S^\ell \vdash_{\ell \sqcap \ell'} P \triangleright \{\alpha : T\} \quad \ell' = \mathsf{M}(T) \qquad \Gamma, X : S^\ell\ T \vdash_{\ell''} Q \triangleright \Delta}{\Gamma \vdash_{\ell''} \mathsf{def}\ X(x^\ell, \alpha) = P\ \mathsf{in}\ Q \triangleright \Delta} \lfloor \textsc{Def} \rfloor$$

$$\frac{dom(\mathsf{L}) = \{1, \ldots, n\}}{\Gamma, u : \langle \mathsf{L}, G \rangle^\ell \vdash_\ell \bar{u}[n] \triangleright \emptyset} \lfloor \textsc{MInit} \rfloor \qquad \frac{\Gamma, u : \langle \mathsf{L}, G \rangle^\ell \vdash_\ell P \triangleright \Delta, \alpha : G \upharpoonright \mathtt{p}}{\Gamma, u : \langle \mathsf{L}, G \rangle^\ell \vdash_\ell u[\mathtt{p}](\alpha).P \triangleright \Delta} \lfloor \textsc{MAcc} \rfloor$$

$$\frac{\Gamma \vdash e : S^\ell \quad \Gamma \vdash_{\ell''} P \triangleright \Delta, c : T \quad \ell'' \leq \ell' \leq \ell}{\Gamma \vdash_{\ell''} c!^{\ell'}\langle \Pi, e \rangle.P \triangleright \Delta, c : !\langle \Pi, S^{\ell \downarrow \ell'} \rangle; T} \lfloor \textsc{Send} \rfloor \qquad \frac{\Gamma, x^{\ell'} : S^{\ell'} \vdash_{\ell'} P \triangleright \Delta, c : T \qquad \ell' \leq \ell}{\Gamma \vdash_{\ell'} c?^{\ell}(\mathtt{p}, x^{\ell'}).P \triangleright \Delta, c : ?(\mathtt{p}, S^{\ell \downarrow \ell'}); T} \lfloor \textsc{Rcv} \rfloor$$

$$\frac{\Gamma \vdash e : \langle \mathsf{L}, G \rangle^\ell \quad \Gamma \vdash_{\ell'} P \triangleright \Delta, c : T \quad \ell' \leq \ell}{\Gamma \vdash_{\ell'} c!^{\ell}\langle \Pi, e \rangle.P \triangleright \Delta, c : !\langle \Pi, \langle \mathsf{L}, G \rangle^\ell \rangle; T} \lfloor \textsc{ServSend} \rfloor \qquad \frac{\Gamma, x^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_\ell P \triangleright \Delta, c : T}{\Gamma \vdash_\ell c?^{\ell}(\mathtt{p}, x^\ell).P \triangleright \Delta, c : ?(\mathtt{p}, \langle \mathsf{L}, G \rangle^\ell); T} \lfloor \textsc{ServRcv} \rfloor$$

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta, c : T \quad \ell \leq \ell' = \mathsf{M}(T')}{\Gamma \vdash_\ell c!^{\ell'}\langle\langle \mathtt{q}, c' \rangle\rangle.P \triangleright \Delta, c : !^{\ell'}\langle \mathtt{q}, T' \rangle; T, c' : \vec{r}\delta; T'} \lfloor \textsc{Deleg} \rfloor \qquad \frac{\Gamma \vdash_\ell P \triangleright \Delta, c : T, \alpha : T' \quad \ell = \mathsf{M}(T')}{\Gamma \vdash_\ell c?^{\ell}((\mathtt{p}, \alpha)).P \triangleright \Delta, c : ?^{\ell}(\mathtt{p}, T'); T} \lfloor \textsc{Srec} \rfloor$$

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta, c : T_j \quad j \in I \quad \ell \leq \ell' \leq \prod_{i \in I} \mathsf{M}(T_i)}{\Gamma \vdash_\ell c \oplus^{\ell'} \langle \Pi, \lambda_j \rangle.P \triangleright \Delta, c : \oplus^{\ell'} \langle \Pi, \{\lambda_i : T_i\}_{i \in I} \rangle} \lfloor \textsc{Sel} \rfloor$$

$$\frac{\Gamma \vdash_\ell P_i \triangleright \Delta, c : T_i \quad \ell \leq \prod_{i \in I} \mathsf{M}(T_i)}{\Gamma \vdash_\ell c \&^{\ell}(\mathtt{p}, \{\lambda_i : P_i\}_{i \in I}) \triangleright \Delta, c : \&^{\ell}(\mathtt{p}, \{\lambda_i : T_i\}_{i \in I})} \lfloor \textsc{Branch} \rfloor$$

Table 8: Typing rules for processes.

$\lfloor$NRES$\rfloor$ This rule types the restriction of a process: the use of the restricted name a^ℓ in the process P is constrained by the premise.

$\lfloor$VAR$\rfloor$ This rule types $X\langle e, c \rangle$ with a level which is the meet of the security level of expression e and the security levels of the communications performed on channel c. In this way we take into account the levels of all communications which can be performed by a process bound to X, see also rule $\lfloor$DEF$\rfloor$.

$\lfloor$DEF$\rfloor$ The process P is typed with the meet of the security levels associated with x and α, in agreement with rule $\lfloor$VAR$\rfloor$. The levels of P and Q can be unrelated, since X could not occur in Q. Clearly if X occurs in Q we get $\ell'' \leq \ell \sqcap \ell'$.

$\lfloor$MINIT$\rfloor$ In this rule the standard environment must associate with the identifier u a safety global type. The premise matches the number of participants in the domain of L with the number declared by the initiator. The emptiness of the process environment in the conclusion specifies that there is no further communication behaviour after the initiator.

$\lfloor$MACC$\rfloor$ In this rule the standard environment must also associate with u a safety global type. The premise guarantees that the type of the continuation P in the $\mathtt{p}$-th participant is the $\mathtt{p}$-th projection of the global type G of u. Concerning security levels, in rule $\lfloor$MACC$\rfloor$ we check that the continuation process P conforms to the security level ℓ associated with the service name u. Note that this condition does not follow from well-formedness of environments, since the process P may participate in other sessions, but it is necessary to avoid information leaks. For example, without this

condition we could type

$$\begin{array}{ll} \bar{a}^{\top}[2] & \mid a^{\top}[1](\alpha_1).\alpha_1!\langle 2,\mathsf{true}^{\top}\rangle.\mathbf{0} \mid a^{\top}[2](\alpha_2).\alpha_2?(1,x^{\top}).\bar{b}^{\top}[2] \\ & \mid b^{\top}[1](\beta_1).c^{\bot}[1](\gamma_1).\gamma_1!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0} \mid b^{\top}[2](\beta_2).\mathbf{0} \\ \bar{c}^{\bot}[2] & \mid c[2](\gamma_2).\gamma_2?(1,y^{\bot}).\mathbf{0} \end{array}$$

$\lfloor$SEND$\rfloor$ This rule types the sending of the basic value followed by the conversation P. The first hypothesis binds expression e with type S^{ℓ}, where ℓ is the join of all variables and values in e. The second hypothesis imposes typability of the continuation of the output with security level ℓ''. The third hypothesis relates levels ℓ, ℓ'' and ℓ' (the level to which e will be declassified), preserving the invariant that ℓ'' is a lower bound for all security levels of the actions in the process.

Note that the hypothesis $\ell'' \leq \ell' \leq \ell$ is not really constraining, since P can always be downgraded to ℓ'' using rule $\lfloor$SUBS$\rfloor$ and $\ell' \leq \ell$ follows from well-formedness of $S^{\ell\downarrow\ell'}$.

$\lfloor$RCV$\rfloor$ This rule is the dual of rule $\lfloor$SEND$\rfloor$, but it is more restrictive in that it requires the continuation P to be typable with *exactly* the level ℓ'.

Notice for instance that we cannot type the reception of a $\top$-value followed by a $\bot$-action. In particular we cannot type a process which receives a $\top$-value and then sends it declassified to $\bot$. On the other hand we can type the reception of a $\top\downarrow\bot$-value followed by a $\bot$-action. For instance, in our introductory example of Section 2, rule [Rcv] allows the delegation send in process B to be decorated by $\bot$: this is essential for the typability of process B with the first projection of the global type of service b.

$\lfloor$SERVSEND$\rfloor$ This rule types the sending of a service name followed by the conversation P. The first hypothesis binds expression e with type $\langle \mathsf{L}, G\rangle^{\ell}$, where $\ell = \mathsf{M}(G)$. The second hypothesis imposes typability of the continuation of the output with security level ℓ'. The third hypothesis relates levels ℓ and ℓ' preserving the invariant that ℓ' is a lower bound for all security levels of the actions in the process. As for rule $\lfloor$SEND$\rfloor$ the hypothesis $\ell' \leq \ell$ is not really constraining, since P can always be downgraded to ℓ' using rule $\lfloor$SUBS$\rfloor$.

$\lfloor$SERVRCV$\rfloor$ This rule is the dual of rule $\lfloor$SERVSEND$\rfloor$, but it is more restrictive in that it requires the continuation P to be typable with *exactly* the level ℓ (like $\lfloor$RECV$\rfloor$).

$\lfloor$DELEG$\rfloor$ This rule types a delegating process with the meet of the type of the delegated channel, provided the continuation process P is typable with a lower or equal level.

$\lfloor$SREC$\rfloor$ This rule is the dual of the $\lfloor$DELEG$\rfloor$ rule.

$\lfloor$SEL$\rfloor$ This rule types a selecting process with the level associated with the selection operator, which is less than or equal to the meets of the types in the different possible continuations.

$\lfloor$BRANCH$\rfloor$ This rule is the dual of the $\lfloor$SEL$\rfloor$ rule.

We say that a process P is typable in Γ if $\Gamma \vdash_{\ell} P \triangleright \Delta$ holds for some ℓ, Δ.

The following relations between derived types can be easily shown by induction on type derivations.

Proposition 2. *Let $\Gamma \vdash_{\ell''} P \triangleright \Delta$ and $c : T \in \Delta$.*

- *$T = ?(\mathsf{p}, S^{\ell\downarrow\ell'}); T'$ implies $\ell' \leq \mathsf{M}(T')$,*
- *$T = ?(\mathsf{p}, \langle \mathsf{L}, G\rangle^{\ell}); T'$ implies $\ell \leq \mathsf{M}(T')$,*
- *$T = !^{\ell}\langle \mathsf{q}, T'\rangle; T''$ or $T = ?^{\ell}(\mathsf{p}, T'); T''$ imply $\ell = \mathsf{M}(T')$,*
- *$T = ?^{\ell}(\mathsf{p}, T'); T''$ implies $\ell \leq \mathsf{M}(T'')$,*
- *$T = \oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I}\rangle$ or $T = \&^{\ell}(\mathsf{p}, \{\lambda_i : T_i\}_{i\in I})$ imply $\ell \leq \bigsqcap_{i\in I} \mathsf{M}(T_i)$.*

Proposition 3. *Let $\Gamma \vdash_{\ell''} P \triangleright \Delta$ and $c : T \in \Delta$. If τ_1 precedes τ_2 in T, where $\tau_1 \in \{?(\mathsf{p}_1, S_1^{\ell_1\downarrow\ell}), ?(\mathsf{p}_1, \langle \mathsf{L}_1, G_1\rangle^{\ell}), ?^{\ell}(\mathsf{p}_1, T_1)\}$ and $\tau_2 \in \{?(\mathsf{p}_2, S_2^{\ell_2\downarrow\ell'}), ?(\mathsf{p}_2, \langle \mathsf{L}_2, G_2\rangle^{\ell'}), ?^{\ell'}(\mathsf{p}_2, T_2)\}$, then $\ell \leq \ell'$.*

By way of example, let us consider the typing of a fragment of the component S in our C,S,B example of Section 4.1. Letting:

$$P = \beta_2 \&^{\bot}(1, \{\mathsf{ok} : \eta \oplus^{\bot} \langle 1, \mathsf{ok}\rangle.\eta!^{\bot}\langle \mathsf{Date}^{\bot}, 1\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^{\bot} \langle 1, \mathsf{ko}\rangle.\mathbf{0}\})$$

we can derive:

$$\vdash_\perp P \triangleright \{\beta_2 : \&^\perp(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\}), \eta : T'\}$$

and hence:

$$\vdash_\perp \beta_2!^\perp\langle\langle 1, \alpha_2\rangle\rangle.\beta_2?^\perp((1,\eta)).P \triangleright \{\alpha_2 : \Rsh\delta; T, \beta_2 : !^\perp\langle 1, T\rangle; ?^\perp(1, T'); \&^\perp(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\})\}$$

where

$$T = ?(1, \mathsf{Number}^{\top\downarrow\perp}); T' \qquad T' = \oplus^\perp\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle$$

7.6. Typing queues and **Q***-sets*

We type queues describing the messages they contain: *message types* represent the messages contained in the queues.

Message	$\mathbf{T}$	::=	$!\langle \Pi, S^{\ell\downarrow\ell'}\rangle$	*message value send*	|	$!\langle \Pi, \langle \mathsf{L}, G\rangle^\ell\rangle$	*message service send*
		|	$!^\ell\langle \mathsf{q}, T\rangle$	*message delegation*	|	$\oplus^\ell\langle \Pi, \lambda\rangle$	*message selection*
		|	$\mathbf{T}; \mathbf{T}'$	*message sequence*			

The *message value send type* $!\langle \Pi, S^{\ell\downarrow\ell'}\rangle$, the *message service send type* $!\langle \Pi, \langle \mathsf{L}, G\rangle^\ell\rangle$ and the *message delegation type* $!^\ell\langle \mathsf{q}, T\rangle$ express the communication to all $\mathsf{p} \in \Pi$ or to q of a value of type S^ℓ declassified to level ℓ', of a service of type $\langle \mathsf{L}, G\rangle^\ell$, or of a channel of type T with $\ell = \mathsf{M}(T)$. The *message selection type* $\oplus^\ell\langle \Pi, \lambda\rangle$ represents the communication to all $\mathsf{p} \in \Pi$ of the label λ with level ℓ and $\mathbf{T}; \mathbf{T}'$ represents sequencing of message types (we assume associativity for ;). For example $\oplus^\perp\langle \{1,3\}, \mathsf{ok}\rangle$ is the message type for the message $(2, \{1,3\}, \mathsf{ok}^\perp)$.

In order to take into account the structural congruence on queues given in Table 4, we consider message types modulo the equivalence relation $\approx$ induced by the rules shown in Table 9 (with $\natural \in \{!, !^\ell, \oplus^\ell\}$ and $Z \in \{S^{\ell\downarrow\ell'}, \langle \mathsf{L}, G\rangle^\ell, T, \lambda\}$).

$$\begin{array}{c} \mathbf{T};\, \natural\langle \Pi, Z\rangle;\, \natural'\langle \Pi', Z\rangle; \mathbf{T}' \approx \mathbf{T};\, \natural'\langle \Pi', Z\rangle;\, \natural\langle \Pi, Z\rangle; \mathbf{T}' \quad \text{if } \Pi \cap \Pi' = \emptyset \\ \mathbf{T};\, \natural\langle \Pi, Z\rangle; \mathbf{T}' \approx \mathbf{T};\, \natural\langle \Pi', Z\rangle;\, \natural\langle \Pi'', Z\rangle; \mathbf{T}' \quad \text{if } \Pi = \Pi' \cup \Pi'', \Pi' \cap \Pi'' = \emptyset \end{array}$$

Table 9: Equivalence relation on message types.

$$\frac{}{\Gamma \vdash s : \varepsilon \triangleright \emptyset}\ \lfloor \textsc{QInit} \rfloor \qquad \frac{\Gamma \vdash s : h \triangleright \Theta \qquad \Gamma \vdash v^\ell : S^\ell}{\Gamma \vdash s : h \cdot (\mathsf{p}, \Pi, v^{\ell\downarrow\ell'}) \triangleright \Theta; \{s[\mathsf{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}}\ \lfloor \textsc{QSend} \rfloor$$

$$\frac{\Gamma \vdash s : h \triangleright \Theta \qquad \Gamma \vdash v^\ell : \langle \mathsf{L}, G\rangle^\ell}{\Gamma \vdash s : h \cdot (\mathsf{p}, \Pi, v^{\ell\downarrow\ell}) \triangleright \Theta; \{s[\mathsf{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^\ell\rangle\}}\ \lfloor \textsc{QServSend} \rfloor$$

$$\frac{\Gamma \vdash s : h \triangleright \Theta}{\Gamma \vdash s : h \cdot (\mathsf{p}, \mathsf{q}, s'[\mathsf{p}']^\ell) \triangleright \Theta, s'[\mathsf{p}'] : T; \{s[\mathsf{p}] : !^\ell\langle \mathsf{q}, T\rangle\}}\ \lfloor \textsc{QDeleg} \rfloor$$

$$\frac{\Gamma \vdash s : h \triangleright \Theta}{\Gamma \vdash s : h \cdot (\mathsf{p}, \Pi, \lambda^\ell) \triangleright \Theta; \{s[\mathsf{p}] : \oplus^\ell\langle \Pi, \lambda\rangle\}}\ \lfloor \textsc{QSel} \rfloor \qquad \frac{\Gamma \vdash s : h \triangleright \Theta \qquad \Theta \approx \Theta'}{\Gamma \vdash s : h \triangleright \Theta'}\ \lfloor \textsc{QCong} \rfloor$$

Table 10: Typing rules for single queues.

$$\frac{}{\Gamma \vdash_\emptyset \emptyset \triangleright \emptyset}\ \lfloor \textsc{QSInit} \rfloor \qquad \frac{\Gamma \vdash_\Sigma H \triangleright \Theta \qquad \Gamma \vdash s : h \triangleright \Theta'}{\Gamma \vdash_{\Sigma, s} H \cup \{s : h\} \triangleright \Theta, \Theta'}\ \lfloor \textsc{QSUnion} \rfloor$$

Table 11: Typing rules for **Q**-sets.

Typing judgments for queues have the shape

$$\Gamma \vdash s : h \triangleright \Theta$$

where Θ is a *queue environment* associating session types or message types with channels:

$$\Theta ::= \emptyset \mid \Theta, s[\mathsf{p}] : T \mid \Theta, s[\mathsf{p}] : \mathbf{T}$$

The equivalence $\approx$ on message types can be trivially extended to queue environments:

$$\{s[\mathtt{p}_i] : \mathbf{T}_i \mid i \in I\} \approx \{s[\mathtt{p}_i] : \mathbf{T}'_i \mid i \in I\} \text{ if } \mathbf{T}_i \approx \mathbf{T}'_i \text{ for all } i \in I$$

Typing rules for queues are given in Table 10. The empty queue has an empty queue environment (rule $\lfloor$QINIT$\rfloor$). In rules $\lfloor$QSEND$\rfloor$, $\lfloor$QSERVSEND$\rfloor$, $\lfloor$QDELEG$\rfloor$ and $\lfloor$QSEL$\rfloor$, each message adds an output type to the current type of the sending channel. Rule $\lfloor$QDELEG$\rfloor$ also adds a session type for the delegated channel. The composition ";" of queue environments is defined only if no channel has a session type in both environments. This condition is essential for channel linearity, since it assures that the same channel cannot be exchanged in more than one message at time. More precisely:

$$\begin{array}{rcl} \Theta;\Theta' & = & \{s[\mathtt{p}] : \mathbf{T};\mathbf{T}' \mid s[\mathtt{p}] : \mathbf{T} \in \Theta \text{ and } s[\mathtt{p}] : \mathbf{T}' \in \Theta'\} \cup \{s[\mathtt{p}] : \mathbf{T} \mid s[\mathtt{p}] : \mathbf{T} \in \Theta \text{ and } s[\mathtt{p}] \notin dom(\Theta')\} \cup \\ & & \{s[\mathtt{p}] : \mathbf{T}' \mid s[\mathtt{p}] \notin dom(\Theta) \text{ and } s[\mathtt{p}] : \mathbf{T}' \in \Theta'\} \cup \{s[\mathtt{p}] : T \mid s[\mathtt{p}] : T \in \Theta \cup \Theta'\} \end{array}$$

if $s[\mathtt{p}] : T \in \Theta$ implies $s[\mathtt{p}] \notin dom(\Theta')$ and $s[\mathtt{p}] : T \in \Theta'$ implies $s[\mathtt{p}] \notin dom(\Theta)$.

Rule $\lfloor$QCONG$\rfloor$ allows a queue to be typed with equivalent queue environments.

Example: we can derive $\vdash s : (2, \{1,3\}, \mathsf{ok}^\top) \triangleright \{s[2] : \oplus^\top \langle \{1,3\}, \mathsf{ok} \rangle\}$.

Typing judgments for **Q**-sets have the shape:

$$\Gamma \vdash_\Sigma H \triangleright \Theta$$

where Σ is the set of session names which occur free in H.

Typing rules for **Q**-sets are given in Table 11. As for process environments, Σ, Σ' and Θ, Θ' are defined only if $\Sigma \cap \Sigma' = \emptyset$ and $dom(\Theta) \cap dom(\Theta') = \emptyset$, respectively.

The empty **Q**-set has an empty queue environment (rule $\lfloor$QSINIT$\rfloor$). Rule $\lfloor$QSUNION$\rfloor$ types the addition of a queue to the set H by checking that a queue associated with that session name is not already present.

7.7. Typing configurations

Typing judgments for runtime configurations C have the form:

$$\Gamma \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$$

They associate with a configuration the environments Δ and Θ mapping channels to session and message types, respectively. We call $< \Delta \diamond \Theta >$ a *configuration environment*. Channel linearity is assured also by forbidding a channel to occur both in the process and in a message as delegated channel. This condition is mirrored by the following definition of well-formedness of configuration environments.

Definition 7.2. *A configuration environment $< \Delta \diamond \Theta >$ is well formed if no channel with a session type in Θ occurs in Δ, i.e. if $s[\mathtt{p}] : T \in \Theta$ implies $s[\mathtt{p}] \notin dom(\Delta)$.*

A *configuration type* is a session type, or a message type, or a message type followed by a session type:

$$\begin{array}{lcll} \text{Configuration} & \mathscr{T} & ::= & T \qquad \textit{session} \\ & & \mid & \mathbf{T} \qquad \textit{message} \\ & & \mid & \mathbf{T};T \qquad \textit{continuation} \end{array}$$

An example of configuration type is:

$$\oplus^\perp \langle \{1,3\}, \mathsf{ok} \rangle ; !\langle \{3\}, \mathsf{String}^\top \rangle ; ?(3, \mathsf{Number}^\perp); \mathsf{end}$$

It is easy to check that for typable initial configurations the set of session names and the process and queue environments are all empty.

Since channels with roles occur both in processes and in queues, a configuration environment associates configuration types with channels.

Definition 7.3. *The configuration type of a channel $s[\mathtt{p}]$ in a configuration environment $< \Delta \diamond \Theta >$ (notation $< \Delta \diamond \Theta > (s[\mathtt{p}])$) is defined by:*

$$< \Delta \diamond \Theta > (s[\mathtt{p}]) = \begin{cases} \mathbf{T};T & \textit{if } s[\mathtt{p}] : \mathbf{T} \in \Theta \textit{ and } s[\mathtt{p}] : T \in \Delta, \\ \mathbf{T} & \textit{if } s[\mathtt{p}] : \mathbf{T} \in \Theta \textit{ and } s[\mathtt{p}] \notin dom(\Delta), \\ T & \textit{if } s[\mathtt{p}] : T \in \Theta \textit{ and } s[\mathtt{p}] \notin dom(\Delta), \\ T & \textit{if } s[\mathtt{p}] \notin dom(\Theta) \textit{ and } s[\mathtt{p}] : T \in \Delta, \\ \mathsf{end} & \textit{if } s[\mathtt{p}] \notin dom(\Theta) \cup dom(\Delta). \end{cases}$$

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta \quad \Gamma \vdash_\Sigma H \triangleright \Theta \quad <\Delta \diamond \Theta> \text{ is coherent}}{\Gamma \vdash_\Sigma <P, H> \triangleright <\Delta \diamond \Theta>} \lfloor \text{CINIT} \rfloor$$

$$\frac{\Gamma \vdash_{\Sigma_1} C_1 \triangleright <\Delta_1 \diamond \Theta_1> \quad \Gamma \vdash_{\Sigma_2} C_2 \triangleright <\Delta_2 \diamond \Theta_2> \quad <\Delta_1, \Delta_2 \diamond \Theta_1, \Theta_2> \text{ is coherent}}{\Gamma \vdash_{\Sigma_1, \Sigma_2} C_1 \| C_2 \triangleright <\Delta_1, \Delta_2 \diamond \Theta_1, \Theta_2>} \lfloor \text{CPAR} \rfloor$$

$$\frac{\Gamma \vdash_\Sigma C \triangleright <\Delta \diamond \Theta>}{\Gamma \vdash_{\Sigma \setminus s} (\nu s) C \triangleright <\Delta \setminus s \diamond \Theta \setminus s>} \lfloor \text{GSRES} \rfloor \qquad \frac{\Gamma, a^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_\Sigma C \triangleright <\Delta \diamond \Theta>}{\Gamma \vdash_\Sigma (\nu a^\ell) C \triangleright <\Delta \diamond \Theta>} \lfloor \text{GNRES} \rfloor$$

Table 12: Typing rules for configurations.

Clearly the configuration types of all channels are defined only for well-formed configuration environments.

Configuration types can be projected on participants.

Definition 7.4. *The* projection of the configuration type $\mathscr{T}$ onto $\mathtt{q}$*, denoted by* $\mathscr{T} \upharpoonright \mathtt{q}$*, is defined by:*

$$(\,!\langle \Pi, S^{\ell \downarrow \ell'} \rangle; \mathscr{T}) \upharpoonright \mathtt{q} = \begin{cases} !S^{\ell \downarrow \ell'}; \mathscr{T} \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ \mathscr{T} \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (\,!\langle \Pi, \langle \mathsf{L}, G \rangle^\ell \rangle; \mathscr{T}) \upharpoonright \mathtt{q} = \begin{cases} !\langle \mathsf{L}, G \rangle^\ell; \mathscr{T} \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ \mathscr{T} \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$(\,!^\ell \langle \mathtt{p}, T \rangle; \mathscr{T}) \upharpoonright \mathtt{q} = \begin{cases} !^\ell T; \mathscr{T} \upharpoonright \mathtt{q} & \text{if } \mathtt{q} = \mathtt{p}, \\ \mathscr{T} \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (\oplus^\ell \langle \Pi, \lambda \rangle; \mathscr{T}) \upharpoonright \mathtt{q} = \begin{cases} \oplus^\ell \lambda; \mathscr{T} \upharpoonright \mathtt{q} & \text{if } \mathtt{q} \in \Pi, \\ \mathscr{T} \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$(?(\mathtt{p}, S^{\ell \downarrow \ell'}); T) \upharpoonright \mathtt{q} = \begin{cases} ?S^{\ell \downarrow \ell'}; T \upharpoonright \mathtt{q} & \text{if } \mathtt{q} = \mathtt{p} \\ T \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (?(\mathtt{p}, \langle \mathsf{L}, G \rangle^\ell); T) \upharpoonright \mathtt{q} = \begin{cases} ?\langle \mathsf{L}, G \rangle^\ell; T \upharpoonright \mathtt{q} & \text{if } \mathtt{q} = \mathtt{p} \\ T \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$(?^\ell(\mathtt{p}, T); T') \upharpoonright \mathtt{q} = \begin{cases} ?^\ell T; T' \upharpoonright \mathtt{q} & \text{if } \mathtt{q} = \mathtt{p} \\ T' \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (\mu \boldsymbol{t}.T) \upharpoonright \mathtt{q} = \begin{cases} \mu \boldsymbol{t}.(T \upharpoonright \mathtt{q}) & \text{if } \mathtt{q} \text{ occurs in } T, \\ \mathsf{end} & \text{otherwise.} \end{cases}$$

$$(\oplus^\ell \langle \Pi, \{\lambda_i : T_i\}_{i \in I} \rangle) \upharpoonright \mathtt{q} = \begin{cases} \oplus^\ell \{\lambda_i : T_i \upharpoonright \mathtt{q}\}_{i \in I} & \text{if } \mathtt{q} \in \Pi, \\ T_1 \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases} \qquad (\&^\ell(\mathtt{p}, \{\lambda_i : T_i\}_{i \in I})) \upharpoonright \mathtt{q} = \begin{cases} \&^\ell \{\lambda_i : T_i \upharpoonright \mathtt{q}\}_{i \in I} & \text{if } \mathtt{q} = \mathtt{p}, \\ T_1 \upharpoonright \mathtt{q} & \text{otherwise.} \end{cases}$$

$$\boldsymbol{t} \upharpoonright \mathtt{q} = \boldsymbol{t} \quad (\Upsilon \delta; T) \upharpoonright \mathtt{q} = T \upharpoonright \mathtt{q} \quad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

We also define a duality relation $\bowtie$ between projections of configuration types, which holds when opposite communications are offered (input/output, selection/branching).

Definition 7.5. *The* duality relation *between projections of configuration types is the minimal symmetric relation which satisfies:*

$$\mathsf{end} \bowtie \mathsf{end} \qquad \boldsymbol{t} \bowtie \boldsymbol{t} \qquad T \bowtie T' \implies \mu \boldsymbol{t}.T \bowtie \mu \boldsymbol{t}.T'$$

$$\forall i \in I \; T_i \bowtie T_i' \implies \oplus^\ell \{\lambda_i : T_i\}_{i \in I} \bowtie \&^\ell \{\lambda_i : T_i'\}_{i \in I}$$

$$\mathscr{T} \bowtie T \implies !S^{\ell \downarrow \ell'}; \mathscr{T} \bowtie ?S^\ell; T \qquad \mathscr{T} \bowtie T \implies !\langle \mathsf{L}, G \rangle^\ell; \mathscr{T} \bowtie ?\langle \mathsf{L}, G \rangle^\ell; T \qquad \mathscr{T} \bowtie T \implies !^\ell T'; \mathscr{T} \bowtie ?^\ell T'; T$$

$$\exists i \in I \; \lambda = \lambda_i \;\&\; \mathscr{T} \bowtie T_i \implies \oplus^\ell \lambda; \mathscr{T} \bowtie \&^\ell \{\lambda_i : T_i\}_{i \in I}$$

The above definitions are needed to define coherence of well-formed configuration environments. Informally, this holds when the inputs and the branchings offered by the process agree both with the outputs and the selections offered by the process and with the messages in the queues. More formally:

Definition 7.6. *A well-formed configuration environment* $<\Delta \diamond \Theta>$ *is coherent if* $s[\mathtt{p}] \in dom(\Delta) \cup dom(\Theta)$ *and* $s[\mathtt{q}] \in dom(\Delta) \cup dom(\Theta)$ *imply*

$$<\Delta \diamond \Theta> (s[\mathtt{p}]) \upharpoonright \mathtt{q} \bowtie <\Delta \diamond \Theta> (s[\mathtt{q}]) \upharpoonright \mathtt{p}.$$

The typing rules for configurations are given in Table 12. Rule $\lfloor$CPAR$\rfloor$ types the parallel composition of two configurations. Rules $\lfloor$GSRES$\rfloor$ and $\lfloor$GNRES$\rfloor$ type restriction on session and service names respectively, where $\Sigma \setminus s$, $\Delta \setminus s$ and $\Theta \setminus s$ are defined as expected:

- $\Sigma \setminus s = \{s' |\, s' \in \Sigma \;\&\; s' \neq s\}$

- $\Delta \setminus s = \{s'[\mathtt{p}] : T \mid s'[\mathtt{p}] : T \in \Delta \ \&\ s' \neq s\}$
- $\Theta \setminus s = \{s'[\mathtt{p}] : \mathbf{T} \mid s'[\mathtt{p}] : \mathbf{T} \in \Theta \ \&\ s' \neq s\}$.

Typing rules assure that configurations are always typed with coherent environments.

Proposition 4. *If $\Gamma \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$, then $< \Delta \diamond \Theta >$ is coherent.*

Proof. All rules assure that the configuration environments in the conclusions are coherent. Rules $\lfloor$CINIT$\rfloor$ and $\lfloor$CPAR$\rfloor$ explicitly check this condition. For rule $\lfloor$GSRES$\rfloor$, note that the coherence of $< \Delta \setminus s \diamond \Theta \setminus s >$ follows easily from the coherence of $< \Delta \diamond \Theta >$.

Let us look back again at the C, S, B example of Section 4.1. Recall the following definitions:

$P_{\mathtt{C}} = s_a[1]\&^{\perp}(2, \{\mathsf{ok} : s_a[1]?^{\perp}(2, date^{\perp}).\mathbf{0}, \mathsf{ko} : \mathbf{0}\})$

$P_{\mathtt{S}} = s_b[2]\&^{\perp}(1, \{\mathsf{ok} : \eta \oplus^{\perp} \langle 1, \mathsf{ok}\rangle.\eta!^{\perp}\langle 1, \mathsf{Date}^{\perp}\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^{\perp} \langle 1, \mathsf{ko}\rangle.\mathbf{0}\})$

$P_{\mathtt{B}} = s_b[1]!^{\perp}\langle\langle 2, s_a[2]\rangle\rangle.\mathsf{if}\ valid(cc^{\perp})\ \mathsf{then}\ s_b[1] \oplus^{\perp} \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ s_b[1] \oplus^{\perp} \langle 2, \mathsf{ko}\rangle.\mathbf{0}$

Then we can type the configuration

$$< P_{\mathtt{C}} \mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}} \mid s_a[2]?^{\top}(1, cc^{\perp}).P_{\mathtt{B}}\ ,\ \{s_a : (1,2,\mathsf{CreditCard}^{\top\downarrow\perp}), s_b : \varepsilon\} >$$

by the configuration environment:

$$< \{s_a[1] : T_{a,1}, s_a[2] : T_{a,2}, s_b[1] : T_{b,1}, s_b[2] : T_{b,2}\} \diamond \{\, s_a[1] :!\langle 2, \mathsf{Number}^{\top\downarrow\perp}\rangle\} >$$

where:

$T_{a,1} = \&^{\perp}(2, \{\mathsf{ok} :?(2, \mathsf{String}^{\perp\downarrow\perp}); \mathsf{end}, \mathsf{ko} : \mathsf{end}\})$

$T_{a,2} = ?(1, \mathsf{Number}^{\top\downarrow\perp}); \oplus^{\perp}\langle 1, \{\mathsf{ok} :!\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle$

$T_{b,1} = !^{\perp}\langle 2, \oplus^{\perp}\langle 1, \{\mathsf{ok} :!\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle\rangle; \oplus^{\perp}(2, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\})$

$T_{b,2} = ?^{\perp}(1, \oplus^{\perp}\langle 1, \{\mathsf{ok} :!\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle); \&^{\perp}(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\})$

7.8. Reduction of configuration environments

Since process and queue environments represent future communications, by reducing processes we get different configuration environments. This is formalised by the notion of reduction of configuration environments, denoted by $< \Delta \diamond \Theta > \ \Rightarrow\ < \Delta' \diamond \Theta' >$.

Definition 7.7 (Reduction of configuration environments). *Let $\Rightarrow$ be the reflexive and transitive relation on configuration environments generated by:*

1. $< \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle;\ T\} \diamond \Theta > \ \Rightarrow\ < \{s[\mathtt{p}] : T\} \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\} >$
2. $< \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle;\ T\} \diamond \Theta > \ \Rightarrow\ < \{s[\mathtt{p}] : T\} \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\} >$
3. $< \{s[\mathtt{p}] : !^{\ell}\langle \mathtt{q}, T\rangle;\ T', s'[\mathtt{p}'] : T\} \diamond \Theta > \ \Rightarrow\ < \{s[\mathtt{p}] : T'\} \diamond \Theta; \{s[\mathtt{p}] : !^{\ell}\langle \mathtt{q}, T\rangle, s'[\mathtt{p}'] : T\} >$
4. $< \{s[\mathtt{p}] : \oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I}\rangle\} \diamond \Theta > \ \Rightarrow\ < \{s[\mathtt{p}] : T_j\} \diamond \Theta; \{s[\mathtt{p}] : \oplus^{\ell}(\Pi, \lambda_j)\} > \quad (j \in I)$
5. $< \{s[\mathtt{q}] :?(\mathtt{p}, S^{\ell\downarrow\ell'});\ T\} \diamond \{\, s[\mathtt{p}] : !\langle \mathtt{q}, S^{\ell\downarrow\ell'}\rangle\}; \Theta > \ \Rightarrow\ < \{s[\mathtt{q}] : T\} \diamond \Theta >$
6. $< \{s[\mathtt{q}] :?(\mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell});\ T\} \diamond \{\, s[\mathtt{p}] : !\langle \mathtt{q}, \langle \mathsf{L}, G\rangle^{\ell}\rangle\}; \Theta > \ \Rightarrow\ < \{s[\mathtt{q}] : T\} \diamond \Theta >$
7. $< \{s[\mathtt{q}] :?^{\ell}(\mathtt{p}, T);\ T'\} \diamond \{\, s[\mathtt{p}] : !^{\ell}\langle \mathtt{q}, T\rangle, s'[\mathtt{p}'] : T\}; \Theta > \ \Rightarrow\ < \{s[\mathtt{q}] : T', s'[\mathtt{p}'] : T\} \diamond \Theta >$
8. $< \{s[\mathtt{q}] : \&^{\ell}(\mathtt{p}, \{\lambda_i : T_i\}_{i\in I})\} \diamond \{\, s[\mathtt{p}] : \oplus^{\ell}(\mathtt{q}, \lambda_j)\}; \Theta > \ \Rightarrow\ < \{s[\mathtt{q}] : T_j\} \diamond \Theta > \quad (j \in I)$
9. $< \Delta, \Delta'' \diamond \Theta > \ \Rightarrow\ < \Delta', \Delta'' \diamond \Theta' >$ *if* $< \Delta \diamond \Theta > \ \Rightarrow\ < \Delta' \diamond \Theta' >$

where message types are considered modulo the equivalence relation of Table 9.

The first four rules correspond to participant p putting a value, a session name, a channel or a label in the queue. The following four rules correspond to participant p reading a value, a session name, a channel or a label from the queue. Notice that in the rules for delegation the type of the delegated channel is only moved from Δ to Θ and vice versa. All these rules are computational.[3] The last rule is contextual: notice that we add only statements to the process environments, since only one statement is considered in the first eight reduction rules, while we do not need to add statements to the queue environments, which are always arbitrary.

We say that a queue is *generated by service a*, or *a-generated*, if it is created by applying rule [Link] to the parallel composition of a's participants and initiator.

[3] We refer to [23], Section 3.3, for the difference between computational and contextual rules.

8. Properties

We are now able to state our main results, namely type preservation under reduction and the soundness of our type system for both access control and noninterference.

Our type system also preserves the fundamental properties ensured by classical session types, i.e. linearity of communications inside sessions and absence of communication mismatches. This is easy to see, once we observe that our type system projects down to the classical session type system of [2] when we ignore security levels and associated type constraints. We shall therefore concentrate here on establishing the security properties.

8.1. Subject Reduction

We first state some auxiliary results that are used in the subject reduction proof. The first lemma says that reduction of configuration environments preserves coherence.

Lemma 5. *If* $<\Delta\diamond\Theta>$ *is coherent and* $<\Delta\diamond\Theta> \Rightarrow <\Delta'\diamond\Theta'>$*, then* $<\Delta'\diamond\Theta'>$ *is coherent.*

Proof. Easy by induction on the definition of $\Rightarrow$ (Definition 7.7).

The next three lemmas are inversion lemmas for processes, **Q**-sets and configurations, which immediately follow from our typing rules.

Lemma 6 (Inversion Lemma for Processes).

1. *If* $\Gamma\vdash_\ell \mathbf{0}\triangleright\Delta$*, then* Δ end *only.*
2. *If* $\Gamma\vdash_\ell P\mid Q\triangleright\Delta$*, then* $\Delta=\Delta_1,\Delta_2$ *and* $\Gamma\vdash_\ell P\triangleright\Delta_1$ *and* $\Gamma\vdash_\ell Q\triangleright\Delta_2$*.*
3. *If* $\Gamma\vdash_\ell \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q\triangleright\Delta$*, then* $\Gamma\vdash e:\mathsf{bool}^{\ell'}$ *and* $\Gamma\vdash_\ell P\triangleright\Delta$ *and* $\Gamma\vdash_\ell Q\triangleright\Delta$*.*
4. *If* $\Gamma\vdash_{\ell'}(\nu a^\ell)P\triangleright\Delta$*, then* $\Gamma,a^\ell:\langle \mathsf{L},G\rangle^\ell\vdash_{\ell'} P\triangleright\Delta$*.*
5. *If* $\Gamma\vdash_\ell X\langle e,c\rangle\triangleright\Delta$*, then* $\Gamma=\Gamma',X:S^{\ell'}\ T$ *and* $\Delta=\Delta',c:T$ *and* $\Gamma\vdash e:S^{\ell'}$ *and* $\ell\le\ell'\sqcap\mathsf{M}(T)$ *and* Δ' end *only.*
6. *If* $\Gamma\vdash_{\ell'}\mathsf{def}\ X(x^\ell,\alpha)=P\ \mathsf{in}\ Q\triangleright\Delta$*, then* $\Gamma,X:S^\ell\ T,x^\ell:S^\ell\vdash_{\ell\sqcap\mathsf{M}(T)} P\triangleright\{\alpha:T\}$ *and* $\Gamma,X:S^\ell\ T\vdash_{\ell'} Q\triangleright\Delta$*.*
7. *If* $\Gamma\vdash_\ell \bar{u}[n]\triangleright\Delta$*, then* $\Gamma=\Gamma',u:\langle \mathsf{L},G\rangle^\ell$ *and* $dom(\mathsf{L})=\{1,\ldots,n\}$ *and* $\Delta=\emptyset$*.*
8. *If* $\Gamma\vdash_\ell u[\mathtt{p}](\alpha).P\triangleright\Delta$*, then* $\Gamma=\Gamma',u:\langle \mathsf{L},G\rangle^\ell$ *and* $\Gamma\vdash_\ell P\triangleright\Delta,\alpha:G\upharpoonright\mathtt{p}$*.*
9. *If* $\Gamma\vdash_{\ell''} c!^{\ell'}\langle\Pi,e\rangle.P\triangleright\Delta$*, then*
   (a) *either* $\Delta=\Delta',c:!\langle\Pi,S^{\ell\downarrow\ell'}\rangle;T$ *and* $\Gamma\vdash e:S^\ell$ *and* $\Gamma\vdash_{\ell''} P\triangleright\Delta',c:T$ *and* $\ell''\le\ell'\le\ell$*;*
   (b) *or* $\Delta=\Delta',c:!\langle\Pi,\langle \mathsf{L},G\rangle^\ell\rangle;T$ *and* $\Gamma\vdash e:\langle \mathsf{L},G\rangle^\ell$ *and* $\Gamma\vdash_{\ell''} P\triangleright\Delta',c:T$ *and* $\ell''\le\ell'=\ell$*.*
10. *If* $\Gamma\vdash_{\ell''} c?^{\ell'}(\mathtt{p},x^{\ell'}).P\triangleright\Delta$*, then*
    (a) *either* $\Delta=\Delta',c:?(\mathtt{p},S^{\ell\downarrow\ell'});T$ *and* $\Gamma,x^{\ell'}:S^{\ell'}\vdash_{\ell''} P\triangleright\Delta',c:T$ *and* $\ell''=\ell'\le\ell$*;*
    (b) *or* $\Delta=\Delta',c:?(\mathtt{p},\langle \mathsf{L},G\rangle^\ell);T$ *and* $\Gamma,x^\ell:\langle \mathsf{L},G\rangle^\ell\vdash_{\ell''} P\triangleright\Delta',c:T$ *and* $\ell=\ell'=\ell''$*.*
11. *If* $\Gamma\vdash_\ell c!^{\ell'}\langle\langle\mathtt{q},c'\rangle\rangle.P\triangleright\Delta$*, then* $\Delta=\Delta',c:!^{\ell'}\langle\mathtt{q},T'\rangle;T,c':\vec{r}\delta;T'$ *and* $\Gamma\vdash_\ell P\triangleright\Delta',c:T$ *and* $\ell\le\ell'=\mathsf{M}(T')$*.*
12. *If* $\Gamma\vdash_{\ell'} c?^\ell((\mathtt{p},\alpha)).P\triangleright\Delta$*, then* $\Delta=\Delta',c:?^\ell(\mathtt{p},T');T$ *and* $\Gamma\vdash_{\ell'} P\triangleright\Delta',c:T,\alpha:T'$ *and* $\ell=\ell'=\mathsf{M}(T')$*.*
13. *If* $\Gamma\vdash_\ell c\oplus^{\ell'}\langle\Pi,\lambda_j\rangle.P\triangleright\Delta$*, then* $\Delta=\Delta',c:\oplus^{\ell'}\langle\Pi,\{\lambda_i:T_i\}_{i\in I}\rangle$ *and* $j\in I$ *and* $\Gamma\vdash_\ell P\triangleright\Delta',c:T_j$ *and* $\ell\le\ell'\le\bigsqcap_{i\in I}\mathsf{M}(T_i)$*.*
14. *If* $\Gamma\vdash_{\ell'} c\&^\ell(\mathtt{p},\{\lambda_i:P_i\}_{i\in I})\triangleright\Delta$*, then* $\Delta=\Delta',c:\&^\ell(\mathtt{p},\{\lambda_i:T_i\}_{i\in I})$ *and* $\Gamma\vdash_{\ell'} P_i\triangleright\Delta',c:T_i$ *for all* $i\in I$ *and* $\ell=\ell'\le\bigsqcap_{i\in I}\mathsf{M}(T_i)$*.*

Lemma 7 (Inversion Lemma for Queues and Q-sets).

1. *If* $\Gamma\vdash s:\varepsilon\triangleright\Theta$*, then* $\Theta=\emptyset$*.*
2. *If* $\Gamma\vdash s:h\cdot(\mathtt{p},\Pi,v^{\ell\downarrow\ell'})\triangleright\Theta$*, then* $\Gamma\vdash s:h\triangleright\Theta'$ *and*
   (a) *either* $\Theta\approx\Theta';\{s[\mathtt{p}]:!\langle\Pi,S^{\ell\downarrow\ell'}\rangle\}$ *and* $\Gamma\vdash v^\ell:S^\ell$*;*
   (b) *or* $v^\ell=a^\ell$ *and* $\Theta\approx\Theta';\{s[\mathtt{p}]:!\langle\Pi,\langle \mathsf{L},G\rangle^\ell\rangle\}$ *and* $\Gamma\vdash a:\langle \mathsf{L},G\rangle^\ell$*.*
3. *If* $\Gamma\vdash s:h\cdot(\mathtt{p},\mathtt{q},s'[\mathtt{p}'])\triangleright\Theta$*, then* $\Theta\approx\Theta',s'[\mathtt{p}']:T;\{s[\mathtt{p}]:!\langle\mathtt{q},T\rangle\}$ *and* $\Gamma\vdash s:h\triangleright\Theta'$*.*
4. *If* $\Gamma\vdash s:h\cdot(\mathtt{p},\Pi,\lambda^\ell)\triangleright\Theta$*, then* $\Theta\approx\Theta';\{s[\mathtt{p}]:\oplus^\ell\langle\Pi,\lambda\rangle\}$ *and* $\Gamma\vdash s:h\triangleright\Theta'$*.*
5. *If* $\Gamma\vdash_\Sigma\emptyset\triangleright\Theta$*, then* $\Sigma=\Theta=\emptyset$*.*
6. *If* $\Gamma\vdash_\Sigma H\cup\{s:h\}\triangleright\Theta$*, then* $\Sigma=\Sigma',s$ *and* $\Theta=\Theta_1,\Theta_2$ *and* $\Gamma\vdash_{\Sigma'} H\triangleright\Theta_1$*,* $\Gamma\vdash s:h\triangleright\Theta_2$*.*

Lemma 8 (Inversion Lemma for Configurations).

1. *If* $\Gamma\vdash_\Sigma <P\ ,\ H>\triangleright<\Delta\diamond\Theta>$*, then* $\Gamma\vdash_\ell P\triangleright\Delta$ *and* $\Gamma\vdash_\Sigma H\triangleright\Theta$ *and* $<\Delta\diamond\Theta>$ *is coherent.*

2. *If* $\Gamma \vdash_\Sigma C_1 \| C_2 \triangleright < \Delta \diamond \Theta >$, *then* $\Sigma = \Sigma_1, \Sigma_2$ *and* $\Delta = \Delta_1, \Delta_2$ *and* $\Theta = \Theta_1, \Theta_2$ *and* $\Gamma \vdash_{\Sigma_1} C_1 \triangleright < \Delta_1 \diamond \Theta_1 >$ *and* $\Gamma \vdash_{\Sigma_2} C_2 \triangleright < \Delta_2 \diamond \Theta_2 >$ *and* $< \Delta \diamond \Theta >$ *is coherent.*
3. *If* $\Gamma \vdash_\Sigma (\nu s) C \triangleright < \Delta \diamond \Theta >$, *then* $\Sigma = \Sigma' \setminus s$ *and* $\Delta = \Delta' \setminus s$ *and* $\Theta = \Theta' \setminus s$ *and* $\Gamma \vdash_{\Sigma'} C \triangleright < \Delta' \diamond \Theta' >$.
4. *If* $\Gamma \vdash_\Sigma (\nu a^\ell) C \triangleright < \Delta \diamond \Theta >$, *then* $\Gamma, a^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$.

The next lemma characterises the types due to messages occurring as heads of queues.

Lemma 9.

1. *If* $\Gamma \vdash s : h_1 \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'}) \cdot h_2 \triangleright \Theta$, *then* $\Gamma \vdash s : h_i \triangleright \Theta_i$ $(i = 1,2)$ *and*
   (a) *either* $\Theta \approx \Theta_1; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell \downarrow \ell'} \rangle\}; \Theta_2$ *and* $\Gamma \vdash v^\ell : S^\ell$;
   (b) *or* $v^\ell = a^\ell$ *and* $\Theta \approx \Theta_1; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell \downarrow \ell'} \rangle\}; \Theta_2$ *and* $\Gamma \vdash a^\ell : \langle \mathsf{L}, G \rangle^\ell$.
2. *If* $\Gamma \vdash s : h_1 \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^\ell) \cdot h_2 \triangleright \Theta$, *then* $\Theta \approx \Theta_1; \{s[\mathtt{p}] : !^\ell \langle \mathtt{q}, T \rangle\}; \Theta_2, s'[\mathtt{p}'] : T$ *and* $\Gamma \vdash s : h_i \triangleright \Theta_i$ $(i = 1,2)$.
3. *If* $\Gamma \vdash s : h_1 \cdot (\mathtt{p}, \Pi, \lambda^\ell) \cdot h_2 \triangleright \Theta$, *then* $\Theta \approx \Theta_1; \{s[\mathtt{p}] : \oplus^\ell \langle \Pi, \lambda \rangle\}; \Theta_2$ *and* $\Gamma \vdash s : h_i \triangleright \Theta_i$ $(i = 1,2)$.

The proof of this Lemma is standard, by induction on the length of queues.

Theorem 10 (Type Preservation under Equivalence). *If* $\Gamma \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$ *and* $C \equiv C'$, *then* $\Gamma \vdash_\Sigma C' \triangleright < \Delta \diamond \Theta >$.

Proof. By induction on the definition of $\equiv$.

Lemma 11 (Substitution Lemma).

1. *If* $\Gamma \vdash v^\ell : S^\ell$ *and* $\Gamma, x^{\ell'} : S^{\ell'} \vdash_{\ell''} P \triangleright \Delta$ *and* $\ell'' \leq \ell' \leq \ell$, *then* $\Gamma \vdash_{\ell''} P\{v^{\ell'}/x^{\ell'}\} \triangleright \Delta$.
2. *If* $\Gamma \vdash a^\ell : \langle \mathsf{L}, G \rangle^\ell$ *and* $\Gamma, x^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_{\ell'} P \triangleright \Delta$ *and* $\ell' \leq \ell$, *then* $\Gamma \vdash_{\ell'} P\{a^\ell/x^\ell\} \triangleright \Delta$.
3. *If* $\Gamma \vdash_\ell P \triangleright \Delta, \alpha : T$, *then* $\Gamma \vdash_\ell P\{s[\mathtt{p}]/\alpha\} \triangleright \Delta, s[\mathtt{p}] : T$.

Proof. By standard induction on P.

Theorem 12 (Subject Reduction). *Suppose* $\Gamma \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$ *and* $C \longrightarrow^* C'$. *Then* $\Gamma \vdash_\Sigma C' \triangleright < \Delta' \diamond \Theta' >$ *with* $< \Delta \diamond \Theta > \Rightarrow < \Delta' \diamond \Theta' >$.
Moreover if $C = < P , H >$, $C' = < P' , H' >$ *and* $\Gamma \vdash_\ell P \triangleright \Delta$, *then* $\Gamma \vdash_\ell P' \triangleright \Delta'$.

Proof. We prove the simpler statement (of which Theorem 12 is an immediate corollary):

$$\begin{array}{c} \textit{If } \Gamma \vdash_\Sigma < P , H > \triangleright < \Delta \diamond \Theta > \textit{ and } \Gamma \vdash_\ell P \triangleright \Delta \textit{ and } < P , H > \longrightarrow < P' , H' >, \textit{ then} \\ \Gamma \vdash_\Sigma < P' , H' > \triangleright < \Delta' \diamond \Theta' > \textit{ and } \Gamma \vdash_\ell P' \triangleright \Delta' \textit{ with } < \Delta \diamond \Theta > \Rightarrow < \Delta' \diamond \Theta' >. \end{array}$$

by induction on the derivation of $C \longrightarrow C'$, with a case analysis on the final rule. We only consider some paradigmatic cases. For simplicity we write only the part of the configuration which is reduced, omitting processes and queues that can be needed to type the configuration.

[Link]

$$a^\ell[1](\alpha_1).P_1 \mid \ldots \mid a^\ell[n](\alpha_n).P_n \mid \bar{a}^\ell[n] \longrightarrow (\nu s) < P_1\{s[1]/\alpha_1\} \mid \ldots \mid P_n\{s[n]/\alpha_n\} , s : \varepsilon >$$

By hypothesis we have

$$\Gamma \vdash_\Sigma < a^\ell[1](\alpha_1).P_1 \mid \ldots \mid a^\ell[n](\alpha_n).P_n \mid \bar{a}^\ell[n] , \emptyset > \triangleright < \Delta \diamond \Theta >$$

which implies by Lemma 8(1)

$$\Gamma \vdash_{\ell'} a^\ell[1](\alpha_1).P_1 \mid \ldots \mid a^\ell[n](\alpha_n).P_n \mid \bar{a}^\ell[n] \triangleright \Delta \tag{1}$$

$$\Gamma \vdash_\Sigma \emptyset \triangleright \Theta \tag{2}$$

From (2) by Lemma 7(5) we get $\Sigma = \Theta = \emptyset$.

From (1) by Lemma 6(2) we get $\Delta = \bigcup_{i=1}^n \Delta_i \cup \Delta'$ and

$$\Gamma \vdash_{\ell'} a^\ell[i](\alpha_i).P_i \triangleright \Delta_i \qquad (1 \leq i \leq n) \tag{3}$$

$$\Gamma \vdash_{\ell'} \bar{a}^\ell[n] \triangleright \Delta' \tag{4}$$

From (4) by Lemma 6(7) we get $\Gamma = \Gamma', a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell'}$ and $\Delta' = \emptyset$. Since Γ is a well formed environment, this implies $\ell = \ell'$.

From (3) by Lemma 6(8) we get

$$\Gamma \vdash_{\ell} P_i \triangleright \Delta_i, \alpha_i : G \upharpoonright i \quad (1 \leq i \leq n)$$

which implies by Lemma 11(3)

$$\Gamma \vdash_{\ell} P_i\{s[i]/\alpha_i\} \triangleright \Delta_i, s[i] : G \upharpoonright i \quad (1 \leq i \leq n).$$

Applying rules $\lfloor$CONC$\rfloor$ and $\lfloor$QINIT$\rfloor$ we derive

$$\Gamma \vdash_{\ell} P_1\{s[1]/\alpha_1\} | \ldots | P_n\{s[n]/\alpha_n\} \triangleright \bigcup_{i=1}^{n} (\Delta_i, s[i] : G \upharpoonright i) \tag{5}$$

$$\Gamma \vdash s : \varepsilon \triangleright \emptyset \tag{6}$$

From (6) using rules $\lfloor$QSINIT$\rfloor$ and $\lfloor$QSUNION$\rfloor$ we derive

$$\Gamma \vdash_{\{s\}} s : \varepsilon \triangleright \emptyset \tag{7}$$

Applying rule $\lfloor$CINIT$\rfloor$ to (5) and (7) we derive

$$\Gamma \vdash_{\{s\}} < P_1\{s[1]/\alpha_1\} | \ldots | P_n\{s[n]/\alpha_n\} \ , \ s : \varepsilon > \triangleright < \bigcup_{i=1}^{n} (\Delta_i, s[i] : G \upharpoonright i) \diamond \emptyset >$$

which implies by rule $\lfloor$GSRES$\rfloor$, taking into account that $\Delta = \bigcup_{i=1}^{n} \Delta_i$

$$\Gamma \vdash_{\emptyset} (\nu s) < P_1\{s[1]/\alpha_1\} | \ldots | P_n\{s[n]/\alpha_n\} \ , \ s : \varepsilon > \triangleright < \Delta \diamond \emptyset >$$

This concludes the proof for this case, since $\Rightarrow$ is reflexive.

[Send]

$$< s[\mathtt{p}]!^{\ell'}\langle \Pi, e \rangle.P \ , \ s : h > \longrightarrow < P \ , \ s : h \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'}) > \qquad (e \downarrow v^{\ell})$$

By hypothesis we have

$$\Gamma \vdash_{\Sigma} < s[\mathtt{p}]!^{\ell'}\langle \Pi, e \rangle.P \ , \ s : h > \triangleright < \Delta \diamond \Theta >$$

which implies by Lemma 8(1)

$$\Gamma \vdash_{\ell''} s[\mathtt{p}]!^{\ell'}\langle \Pi, e \rangle.P \triangleright \Delta \tag{8}$$

$$\Gamma \vdash_{\Sigma} s : h \triangleright \Theta \tag{9}$$

From (8) by Lemma 6(9) we get

1. either $\Delta = \Delta', s[\mathtt{p}] : !\langle \Pi, S^{\ell_0 \downarrow \ell'} \rangle; T$ and $\ell'' \leq \ell' \leq \ell_0$ and

$$\Gamma \vdash e : S^{\ell_0} \tag{10}$$

$$\Gamma \vdash_{\ell''} P \triangleright \Delta', s[\mathtt{p}] : T \tag{11}$$

From (10) by subject reduction on expressions we have:

$$\Gamma \vdash v^{\ell} : S^{\ell_0} \tag{12}$$

which implies $\ell_0 = \ell$ by our assumption on the typing of values.
From (9) by Lemma 7(5) and (6) we get $\Sigma = \{s\}$ and

$$\Gamma \vdash s : h \triangleright \Theta \tag{13}$$

Applying rule $\lfloor$QSEND$\rfloor$ on (12) and (13) we derive

$$\Gamma \vdash s : h \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'}) \triangleright \Theta; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell \downarrow \ell'} \rangle\}$$

which implies by rules $\lfloor$QSINIT$\rfloor$ and $\lfloor$QSUNION$\rfloor$:

$$\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'}) \triangleright \Theta; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell \downarrow \ell'} \rangle\} \tag{14}$$

Applying rule $\lfloor$CINIT$\rfloor$ on (11) and (14) we derive

$$\Gamma \vdash_{\{s\}} < P\ ,\ s : h \cdot (\mathtt{p}, \Pi, v^{\ell\downarrow\ell'}) > \triangleright < \Delta', s[\mathtt{p}] : T \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\} >$$

This concludes the proof, since

$$< \Delta', \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle;\ T\} \diamond \Theta > \ \Rightarrow\ < \Delta', s[\mathtt{p}] : T \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\} >$$

2. or $\Delta = \Delta', c : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell_0}\rangle; T$ and $\ell'' \leq \ell' = \ell_0$ and

$$\Gamma \vdash e : \langle \mathsf{L}, G\rangle^{\ell_0} \tag{15}$$
$$\Gamma \vdash_{\ell''} P \triangleright \Delta', c : T \tag{16}$$

From (15) by subject reduction on expressions we have:

$$\Gamma \vdash v^{\ell} : \langle \mathsf{L}, G\rangle^{\ell_0} \tag{17}$$

which implies $\ell = \ell_0$ by our assumption on the typing of values. From (9) by Lemma 7(5) and (6) we get $\Sigma = \{s\}$ and

$$\Gamma \vdash s : h \triangleright \Theta \tag{18}$$

Applying rule $\lfloor$QSERVSEND$\rfloor$ on (17) and (18) we derive

$$\Gamma \vdash s : h \cdot (\mathtt{p}, \Pi, v^{\ell}) \triangleright \Theta; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\}$$

which implies by rules $\lfloor$QSINIT$\rfloor$ and $\lfloor$QSUNION$\rfloor$:

$$\Gamma \vdash_{\{s\}} s : h \cdot (\mathtt{p}, \Pi, v^{\ell})\ \triangleright \Theta; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\} \tag{19}$$

Applying rule $\lfloor$CINIT$\rfloor$ on (11) and (19) we derive

$$\Gamma \vdash_{\{s\}} < P\ ,\ s : h \cdot (\mathtt{p}, \Pi, v^{\ell}) > \triangleright < \Delta', s[\mathtt{p}] : T \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\} >$$

This concludes the proof, since

$$< \Delta', \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle;\ T\} \diamond \Theta > \ \Rightarrow\ < \Delta', s[\mathtt{p}] : T \diamond \Theta; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\} >$$

[Rec]

$$< s[\mathtt{q}]?^{\ell}(\mathtt{p}, x^{\ell'}).P\ ,\ s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h > \longrightarrow < P\{v^{\ell'}/x^{\ell'}\}\ ,\ s : h >$$

By hypothesis we have

$$\Gamma \vdash_{\Sigma} < s[\mathtt{q}]?^{\ell}(\mathtt{p}, x^{\ell'}).P\ ,\ s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h > \triangleright < \Delta \diamond \Theta >$$

which implies by Lemma 8(1)

$$\Gamma \vdash_{\ell''} s[\mathtt{q}]?^{\ell}(\mathtt{p}, x^{\ell'}).P \triangleright \Delta \tag{20}$$
$$\Gamma \vdash_{\Sigma} s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h \triangleright \Theta \tag{21}$$

From (20) by Lemma 6(10) we get

1. either $\Delta = \Delta', s[\mathtt{q}] :?(\mathtt{p}, S^{\ell\downarrow\ell'}); T$ and $\ell'' = \ell' \leq \ell$ and

$$\Gamma, x^{\ell'} : S^{\ell'} \vdash_{\ell'} P \triangleright \Delta', s[\mathtt{q}] : T \tag{22}$$

From (21) by Lemma 7(5) and (6) we get $\Sigma = \{s\}$ and

$$\Gamma \vdash s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h \triangleright \Theta$$

which implies by Lemma 9(1) $\Gamma \vdash s : h \triangleright \Theta'$ and :

(a) either $\Theta \approx \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}; \Theta'$ and

$$\Gamma \vdash v^{\ell} : S_0^{\ell} \tag{23}$$

(b) or $\Theta \approx \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}_0, G_0\rangle^{\ell}\rangle\}; \Theta'$ and

$$\Gamma \vdash v^{\ell} : \langle \mathsf{L}_0, G_0\rangle^{\ell} \tag{24}$$

By the coherence of $< \Delta \diamond \Theta >$ only case (a) is possible with $S = S_0$, and therefore from (22) and (23) by Lemma 11(1) we have

$$\Gamma \vdash_{\ell'} P\{v^{\ell'}/x^{\ell'}\} \triangleright \Delta', s[\mathtt{q}] : T \tag{25}$$

From $\Gamma \vdash s : h \triangleright \Theta'$ using rules $\lfloor$QSInit$\rfloor$ and $\lfloor$QSUnion$\rfloor$ we derive:

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Theta' \tag{26}$$

Applying rule $\lfloor$CInit$\rfloor$ on (25) and (26) we derive

$$\Gamma \vdash_{\{s\}} < P\{v^{\ell'}/x^{\ell'}\}\ ,\ s : h > \triangleright < \Delta', s[\mathtt{q}] : T \diamond \Theta' >$$

This concludes the proof, since

$$< \Delta', s[\mathtt{q}] :?(\mathtt{p}, S^{\ell\downarrow\ell'}); T \diamond \{s[\mathtt{q}] : !\langle \mathtt{p}, S^{\ell\downarrow\ell'}\rangle\}; \Theta' > \ \Rightarrow\ < \Delta', s[\mathtt{q}] : T \diamond \Theta' >$$

2. or $\Delta = \Delta', c :?(\mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell}); T$ and $\ell = \ell'$ and

$$\Gamma, x^{\ell} : \langle \mathsf{L}, G\rangle^{\ell} \vdash_{\ell''} P \triangleright \Delta', c : T \tag{27}$$

From (21) by Lemma 7(5) and (6) we get $\Sigma = \{s\}$ and

$$\Gamma \vdash s : (\mathtt{p}, \mathtt{q}, v^{\ell\downarrow\ell'}) \cdot h \triangleright \Theta$$

which implies by Lemma 9(1) $\Gamma \vdash s : h \triangleright \Theta'$ and
(a) either $\Theta \approx \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}; \Theta'$ and (23)
(b) or $\Theta \approx \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}_0, G_0\rangle^{\ell}\rangle\}; \Theta'$ and (24).
By the coherence of $< \Delta \diamond \Theta >$ only case (b) is possible with $\mathsf{L}_0 = \mathsf{L}$ and $G_0 = G$, and therefore from (27) and (23) by Lemma 11(2) we have

$$\Gamma \vdash_{\ell'} P\{v^{\ell'}/x^{\ell'}\} \triangleright \Delta', s[\mathtt{q}] : T \tag{28}$$

From $\Gamma \vdash s : h \triangleright \Theta'$ using rules $\lfloor$QSInit$\rfloor$ and $\lfloor$QSUnion$\rfloor$ we derive:

$$\Gamma \vdash_{\{s\}} s : h \triangleright \Theta' \tag{29}$$

Applying rule $\lfloor$CInit$\rfloor$ on (28) and (29) we derive

$$\Gamma \vdash_{\{s\}} < P\{v^{\ell'}/x^{\ell'}\}\ ,\ s : h > \triangleright < \Delta', s[\mathtt{q}] : T \diamond \Theta' >$$

This concludes the proof, since

$$< \Delta', s[\mathtt{q}] :?(\mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell}); T \diamond \{s[\mathtt{q}] : !\langle \mathtt{p}, \langle \mathsf{L}, G\rangle^{\ell}\rangle\}; \Theta' > \ \Rightarrow\ < \Delta', s[\mathtt{q}] : T \diamond \Theta' >.$$

8.2. Properties of T-reachable configurations

The definition of coherence (Definition 7.6) does not require that each input has a corresponding output, each branching has a corresponding selection and vice versa. On the other hand, as usual for session types, this holds for configurations reachable from a typable initial configuration.

Definition 8.1 (T-reachable configurations).
A configuration C is T-reachable *if there is a typable initial configuration C_0 such that $C_0 \longrightarrow^* C$.*

Definition 8.2. *A configuration environment is* complete *if $< \Delta \diamond \Theta > (s[\mathtt{p}]) \upharpoonright \mathtt{q} \neq \mathsf{end}$ implies $s[\mathtt{q}] \in dom(\Delta) \cup dom(\Theta)$.*

Following [23], Section 3.3, notice that in Table 5 the first nine rules are computational rules and the last two rules are contextual.

Lemma 13. *If $(\nu \tilde{s})C$ is a T-reachable configuration and C does not contain session restrictions, then $\Gamma \vdash_{\{\tilde{s}\}} C \triangleright < \Delta \diamond \Theta >$ for some Γ, Δ, Θ and the configuration environment $< \Delta \diamond \Theta >$ is complete.*

Proof. The proof is by induction on the length n of the reduction sequence $C_0 \longrightarrow^* (\nu \tilde{s})C$, and then by induction on the inference of the last transition. The basic case $C_0 = (\nu \tilde{s})C$ is trivial. Suppose that $C_0 \longrightarrow^* (\nu \tilde{s'})C_{n-1} \longrightarrow (\nu \tilde{s})C$. By Theorem 12 and Lemma 8(3) we get
$\Gamma \vdash_{\{\tilde{s'}\}} C_{n-1} \triangleright < \Delta_{n-1} \diamond \Theta_{n-1} >$ for some $\Gamma, \Delta_{n-1}, \Theta_{n-1}$. Consider the last computational rule applied to infer the transition $(\nu \tilde{s'})C_{n-1} \longrightarrow (\nu \tilde{s})C$. We distinguish two cases, depending on whether the transition creates or not a new queue s.

Suppose $\tilde{s} = \tilde{s'} \cdot s$. Then the last computational rule applied is [Link], and the result holds by definition of projection of global type.

Suppose now $\tilde{s} = \tilde{s'}$. Then the last computational rule applied cannot be [Link] and $< \Delta \diamond \Theta >$ is obtained from $< \Delta_{n-1} \diamond \Theta_{n-1} >$ using the reduction relation of Definition 7.7. More precisely the reduction $< \Delta_{n-1} \diamond \Theta_{n-1} > \Rightarrow < \Delta \diamond \Theta >$ is obtained by applying one of the eight computational rules and the contextual rule of Definition 7.7. In all cases it is easy to check that if $< \Delta_{n-1} \diamond \Theta_{n-1} >$ is complete, then $< \Delta \diamond \Theta >$ is complete too.

Theorem 14. *The* **Q***-set of a* T*-reachable configuration is monotone.*

Proof. Toward a contradiction assume that $lev_{\downarrow}(\vartheta_1) \not\leq lev_{\downarrow}(\vartheta_2)$ and the message $(\mathtt{p}, \Pi_1, \vartheta_1)$ precedes the message $(\mathtt{p}, \Pi_2, \vartheta_2)$ in a queue belonging to a T-reachable configuration C and $\Pi_1 \cap \Pi_2 \neq \emptyset$. We only consider the case $\Pi_1 = \Pi_2 = \{\mathtt{q}\}$ and $\vartheta_i = v_i^{\ell_i \downarrow \ell'_i}$ where v_i is not a service name for $i = 1,2$, since the proof for the other cases is similar. In this case $lev_{\downarrow}(\vartheta_i) = \ell'_i$ for $i = 1,2$. Suppose $C = (\nu \tilde{s})C'$, where C' does not contain session restrictions, and let C_0 be the typable initial configuration from which C is T-reachable.

By Theorem 12 and Lemma 8(3) we get $\Gamma \vdash_{\{\tilde{s}\}} C' \triangleright < \Delta \diamond \Theta >$ for some Γ, Δ, Θ. Then, by Lemma 7(2) and Lemma 9(1), the queue environment Θ must contain $s[\mathtt{p}] : \mathbf{T};\ !\langle \mathtt{q}, S^{\ell_1 \downarrow \ell'_1} \rangle; \mathbf{T}';\ !\langle \mathtt{q}, S^{\ell_2 \downarrow \ell'_2} \rangle; \mathbf{T}''$ for some $S, \mathbf{T}, \mathbf{T}', \mathbf{T}''$. Lemma 13 ensures completeness of the configuration environment $< \Delta \diamond \Theta >$. We know that $< \Delta \diamond \Theta >$ is also coherent by Proposition 4; this implies that the process environment Δ will contain $s[\mathtt{q}] : T$ for some session type T such that $?(\mathtt{p}, S^{\ell_1 \downarrow \ell'_1})$ precedes $?(\mathtt{p}, S^{\ell_2 \downarrow \ell'_2})$ in T. Proposition 3 requires $\ell'_1 \leq \ell'_2$, and this contradicts $\ell'_1 \not\leq \ell'_2$.

8.3. Soundness for Access Control

We introduce a notion of join for message types and configuration types. The join of message types is always $\bot$. Since a configuration type is a message type, or a session type, or a message type followed by a session type, the join of a configuration type is defined to be the join of its building types. More formally:

$$\mathsf{J}(\mathscr{T}) = \begin{cases} \mathsf{J}(T) & \text{if } \mathscr{T} = \mathbf{T}; T \text{ or } \mathscr{T} = T \\ \bot & \text{if } \mathscr{T} = \mathbf{T} \end{cases}$$

Notice that the join of a configuration type is well defined also if output prefixes and message types cannot be always distinguished, since outputs do not contribute to the join.

Lemma 15. *Let C_0 be an initial configuration, and suppose $\Gamma \vdash_{\emptyset} C_0 \triangleright < \emptyset \diamond \emptyset >$ for some standard environment Γ such that $a^{\ell} : \langle \mathsf{L}, G \rangle^{\ell} \in \Gamma$. If $C_0 \longrightarrow^* (\nu s)C$ and s is an a-generated queue of C, then, assuming $\Gamma \vdash_{\{s\}} C \triangleright < \Delta \diamond \Theta >$, it holds that $\mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{p}])) \leq \mathsf{J}(G \upharpoonright \mathtt{p})$ for each participant $\mathtt{p} \in dom(\mathsf{L})$.*

Proof. The proof is by induction on the length n of the reduction sequence $\longrightarrow^*$, and then by induction on the inference of the last transition.

In the basic case, for $n = 1$, we have $C_0 \longrightarrow (\nu s)C$. Then the last computational rule applied is necessarily [Link], and since $< \Delta \diamond \Theta > (s[\mathtt{p}]) = G \upharpoonright \mathtt{p}$, we may immediately conclude.

Consider now the inductive case. Let $C_0 \longrightarrow^* C_{n-1} \longrightarrow (\nu s)C$. We distinguish two cases, according to whether the queue s is created in the last step (in which case it does not appear in C_{n-1}) or not.

If C_{n-1} does not contain the queue s then, as in the basic case, the last computational rule applied is [Link] and the result follows from the fact that $< \Delta \diamond \Theta > (s[\mathtt{p}]) = G \upharpoonright \mathtt{p}$.

Otherwise let $C_{n-1} = (\nu s)C'$. By Theorem 12 and Lemma 8(3), we have $\Gamma \vdash_{\{s\}} C' \triangleright < \Delta' \diamond \Theta' >$ for some Δ', Θ'. Then, by induction, $\mathsf{J}(< \Delta' \diamond \Theta' > (s[\mathtt{p}])) \leq \mathsf{J}(G \upharpoonright \mathtt{p})$. Now, if the last applied rule does not involve the queue s, we are done by induction. Otherwise, the last computational rule applied cannot be [Link]. In this case $< \Delta' \diamond \Theta' > \Rightarrow < \Delta \diamond \Theta >$ by applying one of the eight computational rules and the contextual rule of Definition 7.7. In all cases it is easy to verify that $\mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{p}])) \leq \mathsf{J}(< \Delta' \diamond \Theta' > (s[\mathtt{p}]))$, and we may conclude.

Theorem 16 (Access Control). *Let C_0 be an initial configuration, and suppose that $\Gamma \vdash_\emptyset C_0 \triangleright < \emptyset \diamond \emptyset >$ for some standard environment Γ such that $a^\ell : \langle \mathsf{L}, G \rangle^\ell \in \Gamma$. If $C_0 \longrightarrow^* (\nu s)C$, where the queue of name s in C is a-generated and contains the message $(\mathtt{p}, \mathtt{q}, \vartheta)$, then $lev_\uparrow(\vartheta) \leq \mathsf{L}(\mathtt{q})$.*

Proof. We only consider the case $\vartheta = v^{\ell \downarrow \ell'}$ where v is not a service name, since the proof for the other cases is similar. By Theorem 12 and Lemma 8(3) we get $\Gamma \vdash_{\{s\}} C \triangleright < \Delta \diamond \Theta >$ for some Δ, Θ. Then, by Lemma 7(2) and Lemma 9(1), the queue environment Θ must contain $s[\mathtt{p}] : \mathbf{T};\ !\langle \mathtt{q}, S^{\ell \downarrow \ell'} \rangle; \mathbf{T}'$ for some $S, \mathbf{T}, \mathbf{T}'$. Lemma 13 ensures completeness of the configuration environment $< \Delta \diamond \Theta >$. We know that $< \Delta \diamond \Theta >$ is also coherent by Proposition 4; this implies that the process environment Δ will contain $s[\mathtt{q}] : T$ for some session type T such that $?(\mathtt{p}, S^{\ell \downarrow \ell'})$ occurs in T. By definition of join and of configuration type (Definition 7.3) we have then $\ell \leq \mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{q}]))$ and by Lemma 15 we get $\mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{q}])) \leq \mathsf{J}(G \upharpoonright \mathtt{q})$. Then we may conclude, since the well-formedness of safety global types implies $\mathsf{J}(G \upharpoonright \mathtt{q}) \leq \mathsf{L}(\mathtt{q})$.

If we changed the `C`, `S`, `B` example by avoiding delegation and sending the (declassified) $\mathsf{CreditCard}$ to the seller, we would incur a violation of access control, since $lev_\uparrow(\mathsf{CreditCard}^{\top \downarrow \bot}) = \top$ and $\mathsf{L}(\mathtt{S}) = \bot$. In fact such a process would not be typable, as may be checked directly or derived from Theorem 16.

8.4. Soundness for Noninterference

In this section we prove the soundness of our type system for the security property. This result is based, as usual, on a Confinement Lemma, which states that if a process is typable with a "high" type, then it cannot affect any "low" queue. For each $\mathscr{L} \subseteq \mathscr{S}$, it is useful to distinguish the class of typable processes which are "essentially $\mathscr{L}$-typed", that is, which can only be typed with levels $\ell \in \mathscr{L}$. Following the terminology of [6], we call these processes *$\mathscr{L}$-bounded*. The following definitions and Lemmas are then quite standard.

Definition 8.3 ($\mathscr{L}$-Boundedness).
A process P is $\mathscr{L}$-bounded *in Γ if P is typable in Γ and $\Gamma \vdash_\ell P \triangleright \Delta$ implies $\ell \in \mathscr{L}$.*

On the set of typable processes, we call *$\mathscr{L}$-highness* the complement of non $\mathscr{L}$-boundedness.

Definition 8.4 ($\mathscr{L}$-highness).
A process P is $\mathscr{L}$-high in Γ if $\Gamma \vdash_\ell P \triangleright \Delta$ for some $\ell \notin \mathscr{L}$.

Lemma 17 (Confinement).
Let P be a $\mathscr{L}$-high process in Γ. Then, for any $\mathbf{Q}$-set H:

$$\langle P, H \rangle \longrightarrow^* (\nu \tilde{r}) \langle P', H' \rangle \text{ implies } H =_{\mathscr{L}} H' \text{ and } P' \text{ } \mathscr{L}\text{-high in } \Gamma.$$

Proof. We prove the result for a one-step reduction $< P\ ,\ H > \longrightarrow (\nu \tilde{r}) < P'\ ,\ H' >$, by induction on the inference of the transition. Then the general result will follow as a simple corollary, observing that $\Gamma \vdash_\ell P' \triangleright \Delta$ by Theorem 12 (Subject Reduction). We consider the most interesting base cases, leaving out the inductive cases which are easy.

- [Link] In this case, since $\Gamma \vdash_\ell P \triangleright \Delta$ is deduced using the typing rules $\lfloor$MINIT$\rfloor$, $\lfloor$MACC$\rfloor$ and $\lfloor$CONC$\rfloor$, we have $P = \bar{a}^\ell[n] \mid \prod_{\mathtt{p}=1}^n a^\ell[\mathtt{p}](\alpha_\mathtt{p}).Q_\mathtt{p}$, with $\Gamma \vdash a : \langle \mathsf{L}, G\rangle^\ell$, and the reduction has the form:

$$< \bar{a}^\ell[n] \mid \prod_{\mathtt{p}=1}^n a^\ell[\mathtt{p}](\alpha_\mathtt{p}).Q_\mathtt{p} \ , \ \emptyset > \longrightarrow (\nu s) < \prod_{\mathtt{p}=1}^n Q_\mathtt{p}\{s[\mathtt{p}]/\alpha_\mathtt{p}\} \ , \ s : \varepsilon >$$

for some fresh name s. Then we may conclude, since $\emptyset =_\mathscr{L} s : \varepsilon$ for any $\mathscr{L}$.

- [Send] In this case, since $\Gamma \vdash_\ell P \triangleright \Delta$ is deduced using the typing rule $\lfloor$SEND$\rfloor$, there exist ℓ_1, ℓ_2 such that $\ell \leq \ell_2 \leq \ell_1$ and $P = s[\mathtt{p}]!^{\ell_2}\langle \Pi, e\rangle.P'$, with $\Gamma \vdash e : S^{\ell_1}$. Here the reduction has the form:

$$< s[\mathtt{p}]!^{\ell_2}\langle \Pi, e\rangle.P' \ , \ s : h > \longrightarrow < P' \ , \ s : h \cdot (\mathtt{p}, \Pi, v^{\ell_1 \downarrow \ell_2}) >,$$

where $e \downarrow v^{\ell_1}$.
Now, $\ell \leq \ell_2$ implies $\ell_2 \notin \mathscr{L}$. From this we deduce that $s : h =_\mathscr{L} s : h \cdot (\mathtt{p}, \Pi, v^{\ell_1 \downarrow \ell_2})$.

- [Rec] In this case, since $\Gamma \vdash_\ell P \triangleright \Delta$ is deduced using the typing rule $\lfloor$RCV$\rfloor$, there exist ℓ_1, P'' such that $\ell \leq \ell_1$ and $P = s[\mathtt{q}]?^{\ell_1}(\mathtt{p}, x^\ell).P''$. Then $P' = P''\{v^\ell/x^\ell\}$ and the reduction is:

$$< s[\mathtt{q}]?^{\ell_1}(\mathtt{p}, x^\ell).P'' \ , \ s : (\mathtt{p}, \mathtt{q}, v^{\ell_1 \downarrow \ell}) \cdot h > \longrightarrow < P''\{v^\ell/x^\ell\} \ , \ s : h >$$

and $s : (\mathtt{p}, \mathtt{q}, v^{\ell_1 \downarrow \ell}) \cdot h =_\mathscr{L} s : h$.

- [DelRec] In this case, since $\Gamma \vdash_\ell P \triangleright \Delta$ is deduced by the typing rule $\lfloor$SREC$\rfloor$, there exists P'' such that $P = s[\mathtt{p}]?^\ell((\mathtt{q}, \alpha)).P''$. Then $P' = P''\{s'[\mathtt{p}']/\alpha\}$ and the reduction is:

$$< s[\mathtt{q}]?^\ell((\mathtt{p}, \alpha)).P'' \ , \ s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^\ell) \cdot h > \longrightarrow < P''\{s'[\mathtt{p}']/\alpha\} \ , \ s : h >$$

Since $\ell \notin \mathscr{L}$, we conclude that $s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^\ell) \cdot h =_\mathscr{L} s : h$.

- [Label] In this case, since $\Gamma \vdash_\ell P \triangleright \Delta$ is deduced by the typing rule $\lfloor$SEL$\rfloor$, we have $P = s[\mathtt{p}] \oplus^\ell \langle \Pi, \lambda\rangle.P'$ and the reduction is:

$$< s[\mathtt{p}] \oplus^\ell \langle \Pi, \lambda\rangle.P' \ , \ s : h > \longrightarrow < P' \ , \ s : h \cdot (\mathtt{p}, \Pi, \lambda^\ell) >$$

Again, since $\ell \notin \mathscr{L}$, we may conclude that $s : h =_\mathscr{L} s : h \cdot (\mathtt{p}, \Pi, \lambda^\ell)$.

Lemma 18 ($\mathscr{L}$-security of $\mathscr{L}$-high processes).
If P is $\mathscr{L}$-high in Γ, then P is $\mathscr{L}$-secure.

Proof. We show that the relation $\mathscr{R} = \{(P_1, P_2) \mid P_i \text{ is } \mathscr{L}\text{-high in } \Gamma \text{ for } i = 1, 2\}$ is a $\mathscr{L}$-bisimulation. Suppose that H_1, H_2 are such that $H_1 =_\mathscr{L} H_2$. Let now $\langle P_1, H_1\rangle \longrightarrow (\nu \tilde{r})\langle P_1', H_1'\rangle$. By Lemma 17, $H_1 =_\mathscr{L} H_1'$ and P_1' is $\mathscr{L}$-high in Γ. Then we may choose the empty move as the matching move for $< P_2 \ , \ H_2 >$, given that $H_1' =_\mathscr{L} H_1 =_\mathscr{L} H_2$ and $(P_1', P_2) \in \mathscr{R}$.

We next define the bisimulation relation that will be used in the proof of soundness.

Definition 8.5 (Bisimulation for soundness proof).
Given a down-closed set of security levels $\mathscr{L}$, the relation $\mathscr{R}_\Gamma^\mathscr{L}$ on processes is defined inductively by:

$P_1 \mathscr{R}_\Gamma^\mathscr{L} P_2$ if P_1, P_2 are typable in Γ and
i) either *P_1, P_2 are $\mathscr{L}$-high in Γ*

ii) or *P_1, P_2 are $\mathscr{L}$-bounded in Γ and one of the following holds:*

1. $P_1 = P_2$;
2. $P_i = a^\ell[\mathtt{p}](\alpha_\mathtt{p}).Q_\mathtt{p}^{(i)}$, and $\forall s$ fresh: $Q_\mathtt{p}^{(1)}\{s[\mathtt{p}]/\alpha_\mathtt{p}\} \mathscr{R}_\Gamma^\mathscr{L} Q_\mathtt{p}^{(2)}\{s[\mathtt{p}]/\alpha_\mathtt{p}\}$;
3. $P_i = s[\mathtt{p}]!^{\ell'}\langle \Pi, e\rangle.P_i'$, where $P_1' \mathscr{R}_\Gamma^\mathscr{L} P_2'$ and $\exists v, \exists \ell \geq \ell'$ such that $e \downarrow v^\ell$;
4. $P_i = s[\mathtt{q}]?^\ell(\mathtt{p}, x^{\ell'}).P_i'$, where $\forall v$: $P_1'\{v^{\ell'}/x^{\ell'}\} \mathscr{R}_\Gamma^\mathscr{L} P_2'\{v^{\ell'}/x^{\ell'}\}$;
5. $P_i = s[\mathtt{p}]!^\ell\langle\langle \mathtt{q}, s'[p']\rangle\rangle.P_i'$, where $P_1' \mathscr{R}_\Gamma^\mathscr{L} P_2'$;
6. $P_i = s[\mathtt{q}]?^\ell((\mathtt{p}, \alpha_i)).P_i'$, where $\forall \mathtt{p}', \forall s'$ fresh: $P_1'\{s'[\mathtt{p}']/\alpha_1\} \mathscr{R}_\Gamma^\mathscr{L} P_2'\{s'[\mathtt{p}']/\alpha_2\}$;
7. $P_i = \mathsf{if}\ e\ \mathsf{then}\ P_i'\ \mathsf{else}\ P_i''$ where $P_1' \mathscr{R}_\Gamma^\mathscr{L} P_2'$ and $P_1'' \mathscr{R}_\Gamma^\mathscr{L} P_2''$;

8. $P_i = s[\mathtt{p}] \oplus^{\ell} \langle \Pi, \lambda \rangle.Q_i$, *where* $Q_1 \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_2$;
9. $P_i = s[\mathtt{p}] \&^{\ell} (\mathtt{q}, \{\lambda_j : Q_j^{(i)}\}_{j \in J})$, *where* $\forall j \in J : Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_j^{(2)}$;
10. $P_i = (\nu r) P_i'$, *where* $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'$;
11. $P_i = \mathsf{def}\ D\ \mathsf{in}\ P_i'$, *where* $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'$;
12. $P_i = \prod_{j=1}^{m} Q_j^{(i)}$, *where* $\forall j\ (1 \leq j \leq m) :\ Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_j^{(2)}$ *follows from one of the previous clauses (including the clause of high processes).*

For the proof of soundness, it is convenient to use the following alternative definition of $\mathscr{L}$-bisimulation (where we need to put $\equiv$ in the matching reduction sequence for the case where it is the empty sequence and we have to apply $\equiv$ to obtain the same restriction as in the first process):

Definition 8.6 ($\mathscr{L}$-Bisimulation on processes (alternative formulation)).

A symmetric relation $\mathscr{R} \subseteq (\mathscr{P}r \times \mathscr{P}r)$ is a simple $\mathscr{L}$-bisimulation *if $P_1 \mathscr{R} P_2$ implies, for any pair of monotone* $\mathbf{Q}$*-sets H_1 and H_2 such that $H_1 =_{\mathscr{L}} H_2$ and each $< P_i , H_i >$ is saturated:*

$$\begin{array}{l} \textit{If } < P_1 , H_1 > \longrightarrow (\nu \tilde{r}) < P_1' , H_1' >, \textit{ then there exist } P_2', H_2' \textit{ such that} \\ \quad < P_2 , H_2 > \longrightarrow^{*} \equiv\ (\nu \tilde{r}) < P_2' , H_2' >, \textit{ where } H_1' =_{\mathscr{L}} H_2' \textit{ and } P_1' \mathscr{R} P_2'. \end{array}$$

Processes P_1, P_2 are $\mathscr{L}$-bisimilar, *$P_1 \simeq_{\mathscr{L}} P_2$, if $P_1 \mathscr{R} P_2$ for some $\mathscr{L}$-bisimulation $\mathscr{R}$.*

Theorem 19 (Noninterference). *If P is typable, then P is secure.*

Proof. The proof consists in showing that the relation $\mathscr{R}_{\Gamma}^{\mathscr{L}}$ presented in Definition 8.5 is a $\mathscr{L}$-bisimulation containing the pair (P,P). Note that we have $P \mathscr{R}_{\Gamma}^{\mathscr{L}} P$ by Clause 1. Suppose that $P_1 \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$, $\Gamma \vdash_{\ell_i} P_i \triangleright \Delta_i$ and H_1, H_2 are two monotone $\mathbf{Q}$-sets such that $H_1 =_{\mathscr{L}} H_2$ and $\langle P_i, H_i \rangle$ is saturated for $i = 1,2$. We want to show that each reduction $< P_1 , H_1 > \longrightarrow (\nu \tilde{r}) < P_1' , H_1' >$ is matched by a possibly empty sequence of reductions $< P_2 , H_2 > \longrightarrow^{*} \equiv (\nu \tilde{r}) < P_2' , H_2' >$ such that $H_1' =_{\mathscr{L}} H_2'$ and $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'$. Typability of the $\langle P_i', H_i' \rangle$ will be given by Lemma 12 (Subject Reduction), which we will use without mentioning it. We proceed by induction on the definition of $\mathscr{R}_{\Gamma}^{\mathscr{L}}$. Let us examine the two clauses of Definition 8.5:

i) Both P_1, P_2 are $\mathscr{L}$-high in Γ;

ii) Both P_1, P_2 are $\mathscr{L}$-bounded in Γ and $P_1 \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$ by one of the Clauses 1-12 of Definition 8.5.

Note that, since $\mathscr{L}$-highness is preserved by execution, as we go along executing processes we can pass from Clause *ii*) to Clause *i*), but not vice versa.

i) Let P_i be $\mathscr{L}$-high in Γ for $i = 1,2$. Consider a move $< P_1 , H_1 > \longrightarrow (\nu \tilde{r}) < P_1' , H_1' >$. By the Confinement Lemma (Lemma 17) $H_1' =_{\mathscr{L}} H_1$ and P_1' is $\mathscr{L}$-high in Γ. This move can then be matched by the empty move of $< P_2 , H_2 >$, namely $< P_2 , H_2 > \equiv (\nu \tilde{r}) < P_2 , H_2 >$, where $H_1' =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$. Then we may conclude, since $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$ by Clause *i*) again.

ii) Suppose now that the P_i's are $\mathscr{L}$-bounded. We consider some interesting cases.

- Let $P_i = s[\mathtt{p}]!^{\ell'} \langle \Pi, e \rangle.P_i'$, where $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'$ and $\exists v, \exists \ell \geq \ell'$ such that $e \downarrow v^{\ell}$. In this case the reduction is obtained by rule [Send]. Applicability of rule [Send] assures that there is a queue $s : h_1$ in H_1. Since $\langle P_2, H_2 \rangle$ is saturated, there will be a queue $s : h_2$ in H_2. Then, assuming $H_i = K_i \cup s : h_i$, we have $H_1' = K_1 \cup s : h_1 \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'})$, and the matching move will be $< P_2 , H_2 > \longrightarrow (\nu \tilde{r}) < P_2' , H_2' >$, where $H_2' = K_2 \cup s : h_2 \cdot (\mathtt{p}, \Pi, v^{\ell \downarrow \ell'})$. Indeed, if $\ell' \notin \mathscr{L}$ then $H_1' =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2 =_{\mathscr{L}} H_2'$. If $\ell' \in \mathscr{L}$ then $H_1' =_{\mathscr{L}} H_2'$ follows from $K_1 =_{\mathscr{L}} K_2$ and $s : h_1 =_{\mathscr{L}} s : h_2$.
- If $P_i = s[\mathtt{q}]?^{\ell}((\mathtt{p}, \alpha_i)).P_i'$, then the reduction is obtained by rule [DelRec]. Since $\Gamma \vdash_{\ell_i} P_i \triangleright \Delta_i$ is deduced by the typing rule $\lfloor$SREC$\rfloor$, we know that $\ell = \ell_i$. Moreover $\ell_i \in \mathscr{L}$ because P_i is $\mathscr{L}$-bounded, hence $\ell \in \mathscr{L}$. Applicability of rule [DelRec] ensures that a message $(\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell})$ must occur at the head of queue s in H_1. Given that $H_1 =_{\mathscr{L}} H_2$, the same message must occur in H_2. Since H_2 is monotone, the message $(\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell})$ must occur at the head of queue s also in H_2. Then, assuming $H_i = K_i \cup s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell}) \cdot h_i$, we get:

  $< P_1 , K_1 \cup s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell}) \cdot h_1 > \longrightarrow < P_1'\{s'[\mathtt{p}']/\alpha_1\} , K_1 \cup s : h_1 >$

  $< P_2 , K_2 \cup s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']^{\ell}) \cdot h_2 > \longrightarrow < P_2'\{s'[\mathtt{p}']/\alpha_2\} , K_2 \cup s : h_2 >$

  We may then conclude since by hypothesis $P_1'\{s'[\mathtt{p}']/\alpha_1\} \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'\{s'[\mathtt{p}']/\alpha_2\}$, and $H_1 =_{\mathscr{L}} H_2$ implies $K_1 \cup s : h_1 =_{\mathscr{L}} K_2 \cup s : h_2$.

- If $P_i = \mathsf{def}\ D\ \mathsf{in}\ P_i'$, then either $< P_1\ ,\ H_1 >$ reduces by applying rule [Def] or $< P_1'\ ,\ H_1 >$ reduces.
  In the first case let D be $X(x^\ell, \alpha) = Q$ and $P_1' = X\langle e, s[\mathtt{p}]\rangle \mid Q_1$. From $P_1' \mathscr{R}_\Gamma^{\mathscr{L}} P_2'$, by Clauses 12 and 1 of Definition 8.5 we get $P_2' = X\langle e, s[\mathtt{p}]\rangle \mid Q_2$ and $Q_1 \mathscr{R}_\Gamma^{\mathscr{L}} Q_2$. Then, if $e \downarrow v^\ell$ we get
  $< P_1\ ,\ H_1 > \longrightarrow < \mathsf{def}\ D\ \mathsf{in}\ (Q\{v^\ell/x^\ell\}\{s[\mathtt{p}]/\alpha\} \mid Q_1)\ ,\ H_1 >$ and
  $< P_2\ ,\ H_2 > \longrightarrow < \mathsf{def}\ D\ \mathsf{in}\ (Q\{v^\ell/x^\ell\}\{s[\mathtt{p}]/\alpha\} \mid Q_2)\ ,\ H_2 >$, and we may conclude using Clause 12.
  In the second case we can apply the induction hypothesis to $P_1' \mathscr{R}_\Gamma^{\mathscr{L}} P_2'$.
- Let $P_i = \prod_{j=1}^m Q_j^{(i)}$, where $\forall j\ (1 \le j \le m):\ Q_j^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_j^{(2)}$ follows from one of the previous clauses (including the clause of high processes). Then the move of $< P_1\ ,\ H_1 >$ comes either from a single component $Q_j^{(1)}$, or from a group of components synchronising among themselves to open a session. Without loss of generality, we may assume that in the first case the moving component is $Q_1^{(1)}$, and in the second case the group of moving components is $\bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}$. Let us examine the two cases.
  (a) If $Q_1^{(1)}$ moves alone, then we have a move $< P_1\ ,\ H_1 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(1)} \mid \prod_{j=2}^n Q_j^{(1)}\ ,\ H_1' >$, which is deduced from $< Q_1^{(1)}\ ,\ H_1 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(1)}\ ,\ H_1' >$. Since $Q_1^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_1^{(2)}$, by induction we have a move $< Q_1^{(2)}\ ,\ H_2 > \longrightarrow^* \equiv (\nu\tilde{r}) < Q_1'^{(2)}\ ,\ H_2' >$ with $Q_1'^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_1'^{(2)}$ and $H_1' =_{\mathscr{L}} H_2'$, whence we may infer $< P_2\ ,\ H_2 > \longrightarrow^* \equiv (\nu\tilde{r}) < Q_1'^{(2)} \mid \prod_{j=2}^n Q_j^{(2)}\ ,\ H_2' >$, and thus we conclude by Clause 12 again.
  (b) Let now $P_1 = \bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)} \mid \prod_{j=n+2}^m Q_j^{(1)}$ and suppose that a new session is created through a synchronisation of all processes in $\bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}$.
  Let then $< P_1\ ,\ H_1 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=n+2}^m Q_j^{(1)}\ ,\ H_1 \cup \{s : \varepsilon\} >$. This move is deduced from $< \bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}\ ,\ H_1 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}\ ,\ H_1 \cup \{s : \varepsilon\} >$.
  We distinguish two cases, according to whether $\ell \in \mathscr{L}$ or $\ell \notin \mathscr{L}$:
  * If $\ell \in \mathscr{L}$, by the typing rule $\lfloor$MACC$\rfloor$ each $Q_j^{(1)}$ with $1 \le j \le n+1$ must be $\mathscr{L}$-bounded. In this case $Q_1^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_1^{(2)}$ follows from Clause 1 and for each j such that $2 \le j \le n+1$ $Q_j^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_j^{(2)}$ follows from Clause 2. Hence $Q_1^{(2)} = \bar{a}^\ell[n]$, $Q_j^{(2)} = a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(2)}$ and $R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mathscr{R}_\Gamma^{\mathscr{L}} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$. Then $< \bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(2)}\ ,\ H_2 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}\ ,\ H_2 \cup \{s : \varepsilon\} >$, whence we deduce $< P_2\ ,\ H_2 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=n+2}^m Q_j^{(2)}\ ,\ H_2 \cup \{s : \varepsilon\} >$. Since $R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mathscr{R}_\Gamma^{\mathscr{L}} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$ for $2 \le j \le n+1$, we may conclude by Clause 12 again.
  * If $\ell \notin \mathscr{L}$, then by the typing rule $\lfloor$MACC$\rfloor$ each $Q_j^{(1)}$ with $1 \le j \le n+1$ must be $\mathscr{L}$-high in Γ. Then also each $Q_j^{(2)}$ must be $\mathscr{L}$-high in Γ because $Q_j^{(1)} \mathscr{R}_\Gamma^{\mathscr{L}} Q_j^{(2)}$ follows necessarily from Clause i) (since in all the other clauses both processes must be $\mathscr{L}$-bounded). Hence $< \prod_{j=1}^{n+1} Q_j^{(2)}\ ,\ H_1 >$ may simulate the move $< \bar{a}^\ell[n] \mid \prod_{j=2}^{n+1} a^\ell[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}\ ,\ H_1 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}\ ,\ H_1 \cup \{s : \varepsilon\} >$ by the empty move, since $H_1 \cup \{s : \varepsilon\} =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$ (because empty queues are not observed) and by Lemma 17 we know that $\prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$ is $\mathscr{L}$-high in Γ. Since $Q_j^{(2)}$ $\mathscr{L}$-high in Γ for $1 \le j \le n+1$ implies $\prod_{j=1}^{n+1} Q_j^{(2)}$ $\mathscr{L}$-high in Γ, we conclude $\prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$ $\mathscr{R}_\Gamma^{\mathscr{L}}$ $\prod_{j=1}^{n+1} Q_j^{(2)}$ by Clause i). Then, recomposing with the remaining components, $< P_2\ ,\ H_2 >$ may also reply by the empty move to the move: $< P_1\ ,\ H_1 > \longrightarrow (\nu s) < \prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=n+2}^m Q_j^{(1)}\ ,\ H_1' >$, since $\prod_{j=2}^{n+1} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$ $\mid$ $\prod_{j=n+2}^m Q_j^{(1)}$ $\mathscr{R}_\Gamma^{\mathscr{L}}$ $\prod_{j=2}^m Q_j^{(2)}$ by Clause 12 again.

Note that in the proof of non-interference we used both the saturation and the monotonicity conditions as defined in Section 5. We claim that these conditions are reasonable, since they hold for all T-reachable configurations, i.e. for all configurations that can be obtained by reducing typable user processes with the empty **Q**-set, as shown in Lemma 1 and Theorem 14.

We conclude with an observation about the compositionality of our notion of security. It is clear that security is not preserved by input prefixing in case the input is "high". In the absence of restrictions on the application of our semantic rules, our security notion is not preserved by parallel composition either. Indeed, the three processes discussed in Section 6.5, Section 6.6, Section 6.7, are not secure, although all their components are secure. Let us just recall here the first of these examples, completed with security levels on services:

Consider the following process:

$$s[2]?(1,x^{\top}).\bar{b}^{\ell}[2] \mid b^{\ell}[1](\beta_1).\beta_1!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0} \mid b^{\ell}[2](\beta_2).\beta_2?(1,y^{\bot}).\mathbf{0}$$

As argued in Section 6.5, this process is insecure while all its components are secure. Note that this process is untypable, whatever the choice of ℓ. Indeed, in a well-formed environment Γ, the service b^{ℓ} must be given a safety global type $\langle \mathsf{L}, G\rangle^{\ell}$ and then the typing rule $\lfloor$MAcc$\rfloor$ requires both participant bodies $\beta_1!\langle 2,\mathsf{true}^{\bot}\rangle.\mathbf{0}$ and $\beta_2?(1,y^{\bot}).\mathbf{0}$ to be typable at level ℓ. This is only possible if $\ell = \bot$. But then the first component cannot be typed, since the typing rule $\lfloor$Rcv$\rfloor$ requires $\bar{b}^{\ell}[2]$ to be typable at level $\top$.

On the other hand, if we swapped the levels $\top$ and $\bot$ in the above process, then we would obtain:

$$s[2]?(1,x^{\bot}).\bar{b}^{\ell}[2] \mid b^{\ell}[1](\beta_1).\beta_1!\langle 2,\mathsf{true}^{\top}\rangle.\mathbf{0} \mid b^{\ell}[2](\beta_2).\beta_2?(1,y^{\top}).\mathbf{0}$$

which is both secure and typable for $\ell = \top$ (recall that the level of a service is the meet of its global type).

Indeed, as an easy consequence of our soundness Theorem, security is preserved by parallel composition if the components are typable with respect to the same standard environment:

Corollary 20 (Security compositionality for typable processes).
Let P and Q be typable with respect to the same standard environment. If P and Q are secure, then also $P \mid Q$ is secure.

Proof. If P and Q are typable with respect to the same standard environment, then also $P \mid Q$ is typable with respect to that environment, hence by the previous theorem it is also secure.

9. Conclusion and future work

In this work, we have investigated the integration of security requirements into session types. Interestingly, there appears to be an influence of session types on security.

For instance, it is well known that one of the causes of insecure information flow in a concurrent scenario is the possibility of different termination behaviours in the branches of a high conditional. In our calculus, we may distinguish three termination behaviours: (proper) termination, deadlock and divergence. Now, the classical session types of [29] already exclude some combinations of these behaviours in conditional branches. For instance, a non-trivial divergence (whose body contains some communication actions) in one branch cannot coexist with a non-trivial termination in the other branch. Moreover, session types prevent *local deadlocks* due to a bad matching of the communication behaviours of participants in the same session. By adding to classical session types the interaction typing of [2], we would also exclude most of the *global deadlocks* due to a bad matching of the protocols of two interleaved sessions. However, this typing still does not prevent deadlocks due to inverse session calls (for instance, a partner that calls service a and then service b, while the dual partner calls first b and then a). We plan to study a strengthening of interaction typing that would rule out also this kind of deadlock taking inspiration from [17].

The form of declassification considered in this work is admittedly quite simple. To justify our choice, let us first remark that a local declassification construct, such as that proposed in [7] for a classical language with sequential composition, would not be appropriate for our calculus. Indeed, such a construct specifies which additional information flows - with respect to the security policy in force - are authorised within its scope. If we adopted this construct in our calculus, where the only form of sequential composition is prefixing, the scope of a declassification for an input process would span over its whole continuation. In other words, after a declassification it would never be possible to restore the original policy. This is why we chose here a simpler and more punctual form of declassification, which applies to single values rather than to whole subprocesses. On the other hand, we retained from [7] the idea that declassification should be constrained by the access control policy, since a declassified value may only be received by a trusted participant, namely one whose security level is greater than or equal to the original level of the value.

A specificity of our declassification mechanism is that a value v is declassified from ℓ to ℓ' while being transmitted from one participant to another. Moreover, this value is treated as an ℓ'-value by the sender (the value $v^{\ell\downarrow\ell'}$ is $\mathscr{L}$-observable for any $\mathscr{L}$ containing ℓ') while it is treated in a mixed way by the receiver, who receives it as if it were a ℓ-value but then uses it freely as a ℓ'-value. It might be objected that this is a rather abstract and implicit way to perform

declassification, and that the role played by the bisimulation in controlling declassification is not obvious. Indeed, in the *receiver* the usage of the declassified value is not constrained by the bisimulation (since $v^{\ell\downarrow\ell'}$ is already an ℓ'-value when it is read from the queue[4]); it is only constrained by the access control mechanism, which requires the receiver to be ℓ-*trusted*, and therefore assumed to make good use of the original ℓ-value by processing it with a function that safely downgrades it to level ℓ' (what is sometimes called a "permitted function" or a "downgrading policy" [21]). A more explicit solution would consist in replacing $v^{\ell\downarrow\ell'}$ by $f_{\ell\downarrow\ell'}(v)$ in the queue, where f is some permitted function (say, an encryption function) used by the sender to safely transmit the value as an ℓ'-level one - here the subscript $\ell\downarrow\ell'$ indicates that f takes an ℓ-level argument and yields an ℓ'-level result. Then the trusted receiver could retrieve v as an ℓ-level value (using the decryption function), and subsequently process it with his own permitted function $g_{\ell\downarrow\ell'}$ to produce the declassified ℓ'-level result. Here the function f would be used by the sender to protect the value v while it transits in the queue, whereas the function g would be used by the receiver to actually perform the declassification.). On the other hand, in the *sender* the usage of $v^{\ell\downarrow\ell'}$ is constrained by the bisimulation (and a fortiori by the type system), since by observing $v^{\ell\downarrow\ell'}$ as an ℓ'-level value in the queue we prevent the sender to receive any ℓ-level value before performing the declassified send. This means in particular that the sender cannot have previously received v^{ℓ} from some other participant. In other words, the sender must *own* the high value v^{ℓ} in order to be able to transmit it to the receiver with the authorisation to declassify it. Moreover, since the declassified send of v^{ℓ} cannot depend on any other high value, the bisimulation ensures that the sender discloses no more than the value v^{ℓ} itself. To conclude, declassification is controlled by a combined effect of the bisimulation in the sender and of access control in the receiver. The bisimulation itself has no specific clause to deal with declassification. Indeed, we take here the stand that, since our declassification applies to single values and not to security levels, it can be implemented without modifying the definition of bisimulation, thus departing from other approaches ([27], [1]).

Our notion of security could be strengthened by allowing empty queues corresponding to low services to be observed. Then the initiation of a low session would always be observable, no matter whether a low message is exchanged or not in the session. In this way we could avoid requiring saturation of **Q**-sets, since the empty **Q**-set would no longer be $\mathscr{L}$-equal to a **Q**-set consisting of an empty queue corresponding to a low service. Indeed, let us look back at the example that motivated the introduction of the saturation condition, at page 9. If low empty queues were observable, the bisimulation would never test the process $P = s[1]!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0}$ in combination with the two **Q**-sets $\emptyset$ and $\{s : \varepsilon\}$, since these would no more be equivalent for the low observer. However, notice that this would require adding security levels to queues, since the empty queue that is created for a high service should remain unobservable. As a matter of fact, we did explore this alternative definition, in the hope to obtain compositionality for our security property (in its robust variant defined in the Appendix). However, it turned out that adding levels to queues was still not sufficient for compositionality. This is why we adopted the simpler (and weaker) form of observation here.

To enhance the practical use of the present calculus we plan to develop a type inference algorithm, which applied to a typable process returns its type together with the required security levels of session participants in order to guarantee access control.

Acknowledgments We would like to thank Nobuko Yoshida for her encouragement to engage in this work, Bernard Serpette for enlightening remarks and the anonymous referees for insightful comments, which allowed us to improve the presentation of the paper and to clarify some important points.

References

[1] A. Almeida Matos and G. Boudol. On Declassification and the Non-Disclosure Policy. *Journal of Computer Security*, 17:549–597, 2009.

[2] L. Bettini, M. Coppo, L. D'Antoni, M. De Luca, M. Dezani-Ciancaglini, and N. Yoshida. Global Progress in Dynamically Interleaved Multiparty Sessions. In *Proc. CONCUR'08*, volume 5201 of *LNCS*, pages 418–433. Springer, 2008.

[3] K. Bhargavan, R. Corin, P.-M. Deniélou, C. Fournet, and J. J. Leifer. Cryptographic Protocol Synthesis and Verification for Multiparty Sessions. In *Proc. CSF'09*, pages 124–140. IEEE Computer Society, 2009.

[4] M. Boreale, R. Bruni, R. De Nicola, and M. Loreti. Sessions and Pipelines for Structured Service Programming. In *Proc. FMOODS '08*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.

[4] Note that this is an obliged choice, given our definition of bisimulation. If $v^{\ell\downarrow\ell'}$ were treated as an ℓ-value in the queue, then the bisimulation would systematically fail in the receiver, as soon as the value is actually declassified and used as an ℓ'-value.

[5] A. Bossi, R. Focardi, C. Piazza, and S. Rossi. Verifying persistent security properties. *Computer Languages, Systems & Structures*, 30(3-4):231 – 258, 2004.

[6] G. Boudol and I. Castellani. Noninterference for Concurrent Programs and Thread Systems. *Theoretical Computer Science*, 281(1):109–130, 2002.

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

[8] S. Capecchi, I. Castellani, and M. Dezani-Ciancaglini. Information Flow Safety in Multiparty Sessions. In *Proc. EXPRESS'11*, volume 64 of *EPTCS*, pages 16–31, 2011.

[9] S. Capecchi, I. Castellani, M. Dezani-Ciancaglini, and T. Rezk. Session Types for Access and Information Flow Control. In *Proc. CONCUR'10*, volume 6269 of *LNCS*, pages 237–252. Springer, 2010.

[10] R. De Nicola and M. Hennessy. Testing Equivalence for Processes. In *Proc. ICALP'83*, volume 154, pages 548–560. Springer-Verlag, 1983.

[11] D. E. R. Denning. *Cryptography and Data Security*. Addison-Wesley, 1982.

[12] M. Dezani-Ciancaglini and U. de' Liguoro. Sessions and Session Types: an Overview. In *Proc. WSFM'09*, volume 6194 of *LNCS*, pages 1–28. Springer, 2010.

[13] R. Focardi and R. Gorrieri. Classification of Security Properties (Part I: Information Flow). In *Proc. FOSAD'00*, volume 2171 of *LNCS*, pages 331–396. Springer, 2001.

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

[15] K. Honda and N. Yoshida. A Uniform Type Structure for Secure Information Flow. In *Proc. POPL'02*, pages 81–92. ACM Press, 2002.

[16] K. Honda, N. Yoshida, and M. Carbone. Multiparty Asynchronous Session Types. In *Proc. POPL'08*, pages 273–284. ACM Press, 2008.

[17] N. Kobayashi. A Type System for Lock-Free Processes. *Information and Computation*, 177:122–159, 2002.

[18] N. Kobayashi. Type-Based Information Flow Analysis for the Pi-Calculus. *Acta Informatica*, 42(4–5):291–347, 2005.

[19] M. Kolundzija. Security Types for Sessions and Pipelines. In *Proc. WSFM'08*, volume 5387 of *LNCS*, pages 175–190. Springer, 2009.

[20] A. Lapadula, R. Pugliese, and F. Tiezzi. Regulating Data Exchange in Service Oriented Applications. In *Proc. FSEN'07*, volume 4767 of *LNCS*, pages 223–239. Springer, 2007.

[21] P. Li and S. Zdancewic. Downgrading policies and relaxed noninterference. In *Proc. POPL'05*, pages 158–170. ACM Press, 2005.

[22] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. CUP, 1999.

[23] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[24] J. Planul, R. Corin, and C. Fournet. Secure Enforcement for Global Process Specifications. In *Proc. CONCUR'09*, volume 5710 of *LNCS*, pages 511–526. Springer, 2009.

[25] A. Sabelfeld and A. C. Myers. Language-Based Information-Flow Security. *IEEE Journal on Selected Areas in Communications*, 21(1):5–19, Jan. 2003.

[26] A. Sabelfeld and D. Sands. Probabilistic Noninterference for Multi-threaded Programs. In *Proc. CSFW'00*, pages 200–214. IEEE Computer Society, 2000.

[27] A. Sabelfeld and D. Sands. Dimensions and Principles of Declassification. In *Proc. CSFW'05*, pages 255–269. IEEE Computer Society, 2005.

[28] G. Smith and D. Volpano. Secure Information Flow in a Multi-threaded Imperative Language. In *Proc. POPL'98*, pages 355–364. ACM Press, 1998.

[29] K. Takeuchi, K. Honda, and M. Kubo. An Interaction-based Language and its Typing System. In *Proc. PARLE'94*, volume 817 of *LNCS*, pages 398–413. Springer, 1994.

[30] D. Volpano, C. Irvine, and G. Smith. A Sound Type System for Secure Flow Analysis. *Journal of Computer Security*, 4(2,3):167–187, 1996.

Appendix

Appendix A. Robust security

We define an alternative notion of $\mathscr{L}$-bisimulation, which leads to a more discriminating security property, which is closer to typability. We call these new notions *robust*. The idea of robust $\mathscr{L}$-bisimulation stems from the observation that, while the asynchronous communications among participants can be observed by looking at the changes they induce on the **Q**-sets, there is no means to observe the behaviour of a potential session participant, as long as it is not triggered by the initiator together with a complete set of matching participants. This means that an "incomplete process" like $a^{\perp}[1](\alpha).\alpha!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}$ is always considered bisimilar to $\mathbf{0}$. However, it is clear that it behaves differently when plugged into a complying context. For instance, if we put $a^{\perp}[1](\alpha).\alpha!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}$ in parallel with the initiator and the missing partner we get:

$$< \bar{a}^{\perp}[2] \mid a[1](\alpha).\alpha!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0} \mid a^{\perp}[2](\beta).\beta?(1,x^{\perp}).\mathbf{0}\ ,\ \emptyset > \longrightarrow (\nu s)(< s[2]?(1,x^{\perp}).\mathbf{0}\ ,\ \{s:(1,2,\mathsf{true}^{\perp})\} >)$$

while if we put $\mathbf{0}$ in the same context we get $< \bar{a}^{\perp}[2] \mid \mathbf{0} \mid a^{\perp}[2](\beta).\beta?(1,x^{\perp}).\mathbf{0}\ ,\ \emptyset > \not\longrightarrow$.

Since session initiation is a synchronous interaction, which is blocking for all session participants, it would not make sense to use the **Q**-sets to keep track of it. We therefore introduce a notion of *tester*, a sort of behaviourless process which triggers session participants in order to reveal their potential behaviour. Intuitively, testers play the same role as "high contexts" in previous work on security. They also bear a strong analogy with the notion of test introduced by De Nicola and Hennessy in the 80's [10], although they are much simpler in our case, since they are only used to explore potential participant behaviours, and not to test active processes. Moreover, they are not aimed at establishing a testing equivalence, but merely at rendering the bisimulation more contextual.

Formally, a *tester* is a parallel composition of session initiators and degenerate session participants, which cannot reduce by itself.

Definition Appendix A.1. *The syntax of* pre-testers *is*

$$M ::= \bar{a}^{\ell}[n] \mid a^{\ell}[\mathtt{p}](\alpha).\mathbf{0} \mid M \mid M$$

A tester *is an irreducible pre-tester.*

To run processes together with testers, we introduce an asymmetric (non commutative) *triggering* operator $P \curvearrowleft M$, whose behaviour is given by the two new "external link" rules [ExtLink1] and [ExtLink2] in Table A.13, which only differ for the presence or not of the initiator inside the process. Note that rule [ExtLink1] also allows $P \curvearrowleft M$ to move alone, in case $k = n$. From now on we denote by $\longrightarrow$ the reduction obtained by adding the rules [ExtLink1] and [ExtLink2] to those in Table 5. Then the moves of $P \curvearrowleft M$ can take place in any static context. Let us stress that in all cases the tester M disappears after the reduction. This is because testers are not interesting in themselves but only insofar as they may reveal the ability of processes to contribute to a session initiation. A term $P \curvearrowleft M$ is read "P triggered by M".

We are now ready for defining our bisimulation. A relation $\mathscr{R}$ on processes is a *robust $\mathscr{L}$-bisimulation* if, whenever two related processes are put in parallel with the same tester and then coupled with $\mathscr{L}$-equal monotone **Q**-sets yielding saturated configurations, then the reduction relation preserves both the relation $\mathscr{R}$ on processes and the $\mathscr{L}$-equality of **Q**-sets:

$$(a^{\ell}[i_1](\alpha_{i_1}).P_{i_1} \mid \ldots \mid a^{\ell}[i_k](\alpha_{i_k}).P_{i_k} \mid \bar{a}^{\ell}[n]) \ \curvearrowleft\ (a^{\ell}[i_{k+1}](\alpha_{i_{k+1}}).\mathbf{0} \mid \ldots \mid a^{\ell}[i_n](\alpha_{i_n}).\mathbf{0}) \longrightarrow$$
$$(\nu s) < P_{i_1}\{s[i_1]/\alpha_{i_1}\} \mid \ldots \mid P_{i_k}\{s[i_k]/\alpha_{i_k}\}\ ,\ s:\varepsilon >$$
[ExtLink1]

$$(a^{\ell}[i_1](\alpha_{i_1}).P_{i_1} \mid \ldots \mid a^{\ell}[i_k](\alpha_{i_k}).P_{i_k}) \ \curvearrowleft\ (a^{\ell}[i_{k+1}](\alpha_{i_{k+1}}).\mathbf{0} \mid \ldots \mid a^{\ell}[i_n](\alpha_{i_n}).\mathbf{0} \mid \bar{a}^{\ell}[n]) \longrightarrow$$
$$(\nu s) < P_{i_1}\{s[i_1]/\alpha_{i_1}\} \mid \ldots \mid P_{i_k}\{s[i_k]/\alpha_{i_k}\}\ ,\ s:\varepsilon >$$
[ExtLink2]

Table A.13: External link rules.

Definition Appendix A.2 (Robust $\mathscr{L}$-Bisimulation on processes).
A symmetric relation $\mathscr{R} \subseteq (\mathscr{P}r \times \mathscr{P}r)$ is a robust $\mathscr{L}$-bisimulation *if $P_1 \mathscr{R} P_2$ implies, for any tester M and any pair of monotone* $\mathbf{Q}$*-sets H_1 and H_2 such that $H_1 =_{\mathscr{L}} H_2$ and each $< P_i , H_i >$ is saturated:*

If $< P_1 \triangleright M , H_1 > \longrightarrow (\nu \tilde{r}) < P_1' , H_1' >$, then either $H_1' =_{\mathscr{L}} H_2$ and $P_1' \mathscr{R} P_2$, or there exist P_2', H_2' such that $< P_2 \triangleright M , H_2 > \longrightarrow^* (\nu \tilde{r}) < P_2' , H_2' >$, where $H_1' =_{\mathscr{L}} H_2'$ and $P_1' \mathscr{R} P_2'$.

Processes P_1, P_2 are robustly $\mathscr{L}$-bisimilar, *$P_1 \simeq_{\mathscr{L}} P_2$, if $P_1 \mathscr{R} P_2$ for some robust $\mathscr{L}$-bisimulation $\mathscr{R}$.*

Let us illustrate the use of testers with some examples. Note that although "complete processes" may reduce autonomously without the help of any tester, we do not allow the empty process as a tester. Indeed, its use may be simulated by any tester whose names are disjoint from those of the tested process, what we shall call a *fresh tester*. In fact, in Definition Appendix A.2, the quantification over M may be restricted in practice only to those testers that make the process react, by triggering some of its waiting participants, together with one fresh tester that forces the process to progress by itself, if possible, and get stuck otherwise.

Example 21. *Consider the three processes:*

$$\begin{array}{lcl} P_1 & = & \mathbf{0} \\ P_2 & = & a^{\perp}[1](\alpha).\alpha!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0} \\ P_3 & = & \bar{a}^{\perp}[2] \mid a^{\perp}[1](\alpha).\alpha!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0} \mid a^{\perp}[2](\beta).\beta?(1, x^{\perp}).\mathbf{0} \end{array}$$

Let $\mathscr{L} = \{\perp\}$. It is easy to see that no pair of P_i is in the $\mathscr{L}$-bisimilarity relation. Assume in all cases the starting $\mathbf{Q}$*-sets to be $\emptyset$. To distinguish P_2 from P_3, which is a complete process and thus can progress by itself, we may use the fresh tester $\bar{b}^{\perp}[2]$, which will just stay idle in parallel with P_3 while this moves to a state P_3' by opening a session s, generating the* $\mathbf{Q}$*-set $\{s : \varepsilon\}$. Since the tester $\bar{b}^{\perp}[2]$ cannot make P_2 react, P_2 will respond by staying put, thus keeping unchanged the* $\mathbf{Q}$*-set $\emptyset =_{\mathscr{L}} \{s : \varepsilon\}$. So far so good. But now, if P_2 and P_3' are tested again with $\bar{b}^{\perp}[2]$ and coupled with the* $\mathbf{Q}$*-set $\{s : \varepsilon\}$, then P_3' will produce the* $\mathbf{Q}$*-set $\{s : (1, 2, \mathsf{true}^{\perp})\}$ while P_2 remains stuck on the* $\mathbf{Q}$*-set $\{s : \varepsilon\} \neq_{\mathscr{L}} \{s : (1, 2, \mathsf{true}^{\perp})\}$.*

To distinguish P_1 from P_2, we may use the tester $\bar{a}^{\perp}[2] \mid a^{\perp}[2](\beta).\mathbf{0}$, which produces no effect on P_1 but will activate P_2, allowing it to move to a new state P_2' by opening a session s, generating the $\mathbf{Q}$*-set $\{s : \varepsilon\}$. From this point onwards, we proceed as for P_3' above.*

To distinguish P_1 from P_3 we may use the same fresh tester $\bar{b}^{\perp}[2]$ and a similar argumentation.

Testers may also be used to explore some deadlocked processes, when the deadlock is caused by inverted service calls to different services in dual components, for instance. However, as expected, the bisimulation will not equate such a deadlocked process with a correct process where the two calls occur in the same order in the two components.

Example 22. *Consider the following processes, where s is a previously opened session on some service:*

$$\begin{array}{lll} P_1 = & \bar{a}^{\perp}[2] & \mid a^{\perp}[1](\alpha_1).b^{\perp}[1](\beta_1).s[1]!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0} \\ & \bar{b}^{\perp}[2] & \mid b^{\perp}[2](\beta_2).a^{\perp}[2](\alpha_2).\mathbf{0} \\ P_2 = & \bar{a}^{\perp}[2] & \mid a^{\perp}[1](\alpha_1).b^{\perp}[1](\beta_1).s[1]!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0} \\ & \bar{b}^{\perp}[2] & \mid a^{\perp}[2](\alpha_2).b^{\perp}[2](\beta_2).\mathbf{0} \end{array}$$

Process P_1 is deadlocked because the "service calls" to a and b occur in reverse order in its two components. Instead, the calls occur in the same order in the components of P_2. Hence P_2 may progress by itself and thus, as above, it is sufficient to test it with a fresh tester to distinguish it from the blocked process P_1.

The notions of robust $\mathscr{L}$-security and robust security are now defined as usual:

Definition Appendix A.3 (Robust $\mathscr{L}$-Security). *A process P is* robustly $\mathscr{L}$-secure *if $P \simeq_{\mathscr{L}} P$.*

Definition Appendix A.4 (Robust Security). *A process P is* robustly secure *if it is robustly $\mathscr{L}$-secure for every $\mathscr{L}$.*

It is easy to see that robust $\mathscr{L}$-bisimilarity is more discriminating than $\mathscr{L}$-bisimilarity, as shown by the process $a^{\perp}[1](\alpha_1).s[1]!\langle 2, \mathsf{true}^{\perp}\rangle.\mathbf{0}$, which is $\mathscr{L}$-bisimilar to $\mathbf{0}$ but not robustly $\mathscr{L}$-bisimilar to $\mathbf{0}$. As a consequence, the property of robust security is stronger than security, as shown by the following example.

Example 23. *Consider the process:*

$$s[1]?(2,x^{\top}).a^{\top}[1](\alpha).s[1]!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}$$

This processes is secure but not robustly secure. Note that when put in parallel with the initiator and the missing partner of $a^{\top}[1](\alpha).\mathbf{0}$*, then it becomes insecure too. Note that this process is not typable.*

We prove now that typability implies robust security. The proof will make use of the same relation $\mathscr{R}_{\Gamma}^{\mathscr{L}}$ used for security and presented in Definition 8.5. Note that it is not possible here to use an alternative definition of $\mathscr{L}$-bisimulation similar to Definition 8.6, since in case $P_2 \triangleleft M$ does an empty move, then M does not disappear. Hence we need to use the original Definition Appendix A.2 in the proof.

Lemma 24 (Robust confinement).
Let P be a $\mathscr{L}$-high process in Γ. Then, for any $\mathbf{Q}$-set H:

$$\langle P \triangleleft M, H\rangle \longrightarrow^{*} (\nu\tilde{r})\langle P', H'\rangle \text{ implies } H =_{\mathscr{L}} H' \text{ and } P' \ \mathscr{L}\text{-high in } \Gamma.$$

Proof. Simple variation of the proof of Lemma 17 (Confinement Lemma). Note that the residual of $\langle P \triangleleft M, H\rangle$ after the first step of the reduction is a pure process.

Theorem 25 (Soundness for robust security). *If P is typable, then P is robustly secure.*

Proof. The proof consists in showing that the relation $\mathscr{R}_{\Gamma}^{\mathscr{L}}$ presented in Definition 8.5 is a $\mathscr{L}$-bisimulation containing the pair (P,P). Note that we have $P \mathscr{R}_{\Gamma}^{\mathscr{L}} P$ by Clause 1. Suppose that $P_1 \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$, $\Gamma \vdash_{\ell_i} P_i \triangleright \Delta_i$ and H_1, H_2 are two monotone $\mathbf{Q}$-sets such that $H_1 =_{\mathscr{L}} H_2$ and $\langle P_i, H_i\rangle$ is saturated for $i = 1,2$. We want to show that for any tester M, and for each reduction $< P_1 \triangleleft M \ , \ H_1 > \longrightarrow (\nu\tilde{r}) < P_1' \ , \ H_1' >$, either we have $H_1' =_{\mathscr{L}} H_2$ and $P_1' \mathscr{R} P_2$, or $< P_2 \triangleleft M \ , \ H_2 > \longrightarrow^{*} (\nu\tilde{r}) < P_2' \ , \ H_2' >$, where $H_1' =_{\mathscr{L}} H_2'$ and $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2'$ (typability of the $\langle P_i', H_i'\rangle$ will be given by Lemma 12 (Subject Reduction), which we will use without mentioning it). We proceed by induction on the definition of $\mathscr{R}_{\Gamma}^{\mathscr{L}}$. Let us examine the two clauses of Definition 8.5:

i) P_1, P_2 are $\mathscr{L}$-high in Γ (i.e. both P_1, P_2 are not $\mathscr{L}$-bounded in Γ);

ii) Both P_1 and P_2 are $\mathscr{L}$-bounded in Γ and $P_1 \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$ by one of the Clauses 1-12 of Definition 8.5.

Note that, since $\mathscr{L}$-highness is preserved by execution, as we go along executing processes we can pass from Clause *ii*) to Clause *i*), but not vice versa.

i) Let P_1, P_2 be $\mathscr{L}$-high in Γ. Consider a move $< P_1 \triangleleft M \ , \ H_1 > \longrightarrow (\nu\tilde{r}) < P_1' \ , \ H_1' >$. By Lemma 24, we know that $H_1' =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$ and P_1' is $\mathscr{L}$-high in Γ. Then we conclude, since $P_1' \mathscr{R}_{\Gamma}^{\mathscr{L}} P_2$ by Clause *i*) again. *ii*) Suppose now that P_1 and P_2 are $\mathscr{L}$-bounded. We consider some interesting cases. Note that a tester can only interact with the process in Clauses 2, 11 and 12. Hence we do not need to consider testers in the other cases. Indeed, we will omit these cases here, since they are treated exactly as in the proof of Theorem 19.

- If $P_i = a^{\ell}[\mathtt{p}](\alpha_{\mathtt{p}}).Q_{\mathtt{p}}^{(i)}$, let $M = \bar{a}^{\ell}[n] \mid \prod_{\mathtt{q}\in\{1,\ldots,n\},\mathtt{q}\neq\mathtt{p}} a^{\ell}[\mathtt{q}](\alpha_{\mathtt{q}}).\mathbf{0}$, where n is the arity of a. Then we can only apply rule [Link], so we get $< P_1 \triangleleft M \ , \ H_1 > \longrightarrow (\nu s) < Q_{\mathtt{p}}^{(1)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} \ , \ H_1 \cup \{s : \varepsilon\} >$, where s is fresh. Now, this move can be matched by $< P_2 \triangleleft M \ , \ H_2 > \longrightarrow (\nu s) < Q_{\mathtt{p}}^{(2)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} \ , \ H_2 \cup \{s : \varepsilon\} >$, since by hypothesis $Q_{\mathtt{p}}^{(1)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_{\mathtt{p}}^{(2)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}$, and $H_1 =_{\mathscr{L}} H_2$ implies $H_1 \cup \{s : \varepsilon\} =_{\mathscr{L}} H_2 \cup \{s : \varepsilon\}$.
- Let $P_i = \prod_{j=1}^{m} Q_j^{(i)}$, where $\forall j \ (1 \leq j \leq n) : \ Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_j^{(2)}$ follows from one of the previous clauses (including the clause of high processes). Then the move of $< P_1 \triangleleft M \ , \ H_1 >$ comes either from a single component $Q_j^{(1)}$, or from a group of components synchronising among themselves and possibly with the tester to open a session. Without loss of generality, we may assume that in the first case the moving component is $Q_1^{(1)}$, and in the second case the group of moving components is $\prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}$ or $\bar{a}^{\ell}[n] \mid \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}$, for some $k \leq n$. Let us examine the two cases.
  (a) If $Q_1^{(1)}$ moves alone, then the move $< P_1 \triangleleft M \ , \ H_1 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(1)} \mid \prod_{j=2}^{m} Q_j^{(1)} \ , \ H_1' >$, is deduced from $< Q_1^{(1)} \triangleleft M \ , \ H_1 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(1)} \ , \ H_1' >$. Since $Q_1^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_1^{(2)}$, by induction we have either $H_1' =_{\mathscr{L}} H_2$ and $Q_1'^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_1^{(2)}$, and therefore $Q_1'^{(1)} \mid \prod_{j=2}^{m} Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_1^{(2)} \mid \prod_{j=2}^{m} Q_j^{(2)}$ by Clause 12, or $< Q_1^{(2)} \triangleleft M \ , \ H_2 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(2)} \ , \ H_2' >$ with $Q_1'^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_1'^{(2)}$ and $H_1' =_{\mathscr{L}} H_2'$, whence we infer $< P_2 \triangleleft M \ , \ H_2 > \longrightarrow (\nu\tilde{r}) < Q_1'^{(2)} \mid \prod_{j=2}^{m} Q_j^{(2)} \ , \ H_2' >$ and we may conclude by Clause 12 again.

(b) Here we assume that $P_1 = \prod_{j=1}^{k} Q_j^{(1)} \mid \prod_{j=k+1}^{m} Q_j^{(1)}$ and that all the processes in $\prod_{j=1}^{k} Q_j^{(1)}$ synchronise, possibly with the help of the tester.

Let us consider the case where $Q_j^{(1)} = a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)}$ for $1 \le j \le k$ (the case that includes the initiator is similar, and would use the rule [ExtLink1] instead of [ExtLink2]). Let M be a tester which allows the application of rule [ExtLink2]. Suppose that the move of $< P_1 \triangleleft M \, , \, H_1 >$ is $< P_1 \triangleleft M \, , \, H_1 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=k+1}^{m} Q_j^{(1)} \, , \, H_1 \cup \{s : \varepsilon\} >$, deduced from $< \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)} \triangleleft M \, , \, H_1 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \, , \, H_1 \cup \{s : \varepsilon\} >$.

We distinguish two cases, according to whether $\ell \in \mathscr{L}$ or $\ell \notin \mathscr{L}$:

* If $\ell \in \mathscr{L}$, then by the typing rule $\lfloor$MACC$\rfloor$ no $Q_j^{(1)}$ with $1 \le j \le k$ can be high. In this case, for each j such that $1 \le j \le k$, we know that $Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_j^{(2)}$ follows from Clause 2, and hence $Q_j^{(2)} = a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(2)}$ and $R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mathscr{R}_{\Gamma}^{\mathscr{L}} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$ (note that if we were in the case including an initiator $Q_1^{(1)} = \bar{a}^{\ell}[n] = Q_1^{(2)}$, then $Q_1^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_1^{(2)}$ would follow from Clause 1 instead). In this case, $< \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(2)} \triangleleft M \, , \, H_2 >$ may simulate $< \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)} \triangleleft M \, , \, H_1 >$ by consuming the same part of M: $< \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(2)} \triangleleft M \, , \, H_2 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \, , \, H_2 \cup \{s : \varepsilon\} >$. Then $< P_2 \triangleleft M \, , \, H_2 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=k+1}^{m} Q_j^{(2)} \, , \, H_2 \cup \{s : \varepsilon\} >$. Since $R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mathscr{R}_{\Gamma}^{\mathscr{L}} R_{\mathtt{p}_j}^{(2)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\}$, we may conclude by Clause 12 again.

* If $\ell \notin \mathscr{L}$, then the typing rule $\lfloor$MACC$\rfloor$ requires that each component $Q_j^{(1)}$ with $1 \le j \le k$ is high. Then also each $Q_j^{(2)}$ must be high because $Q_j^{(1)} \mathscr{R}_{\Gamma}^{\mathscr{L}} Q_j^{(2)}$ follows necessarily from Clause i) (since in all the other clauses both processes must be $\mathscr{L}$-bounded). In this case, in reply to the move $< \prod_{j=1}^{k} a^{\ell}[\mathtt{p}_j](\alpha_{\mathtt{p}_j}).R_{\mathtt{p}_j}^{(1)} \triangleleft M \, , \, H_1 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \, , \, H_1 \cup \{s : \varepsilon\} >$, by Lemma 24 we have $H_1 \cup \{s : \varepsilon\} =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$ and $\prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mathscr{R}_{\Gamma}^{\mathscr{L}} \prod_{j=1}^{k} Q_j^{(2)}$.

  Then, recomposing with the remaining components, in reply to the move: $< P_1 \triangleleft M \, , \, H_1 > \longrightarrow (\nu s) < \prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=k+1}^{m} Q_j^{(1)} \, , \, H_1' >$ we have $H_1 \cup \{s : \varepsilon\} =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$ and, by Clause 12 again: $\prod_{j=1}^{k} R_{\mathtt{p}_j}^{(1)}\{s[\mathtt{p}_j]/\alpha_{\mathtt{p}_j}\} \mid \prod_{j=k+1}^{m} Q_j^{(1)} \; \mathscr{R}_{\Gamma}^{\mathscr{L}} \; \prod_{j=1}^{m} Q_j^{(2)}$.

Appendix B. The Client, Seller, Bank example at a glance

We recap here the definition, semantics and typing of our Client, Seller, Bank example, introduced in Section 2 and used as a running example throughout the paper.

Syntax: process definition

$$
\begin{array}{lcl}
\mathtt{I} & = & \bar{a}^{\perp}[2] \mid \bar{b}[\perp]2 \\
\mathtt{C} & = & a^{\perp}[1](\alpha_1).\alpha_1!^{\perp}\langle 2, \mathsf{Title}^{\perp}\rangle.\alpha_1!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\ \alpha_1\&^{\perp}(2, \{\mathsf{ok} : \alpha_1?^{\perp}(2, date^{\perp}).\mathbf{0}, \mathsf{ko} : \mathbf{0}\}) \\
\mathtt{S} & = & a^{\perp}[2](\alpha_2).\alpha_2?^{\perp}(1, x^{\perp}).b^{\perp}[2](\beta_2).\beta_2!^{\perp}\langle\langle 1, \alpha_2\rangle\rangle.\beta_2?^{\perp}((1, \eta)). \\
 & & \quad \beta_2\&^{\perp}(1, \{\mathsf{ok} : \eta \oplus^{\perp} \langle 1, \mathsf{ok}\rangle.\eta!^{\perp}\langle 1, \mathsf{Date}^{\perp}\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^{\perp} \langle 1, \mathsf{ko}\rangle.\mathbf{0}\}) \\
\mathtt{B} & = & b^{\perp}[1](\beta_1).\beta_1?^{\perp}((2, \zeta)).\zeta?^{\top}(2, cc^{\perp}).\beta_1!^{\perp}\langle\langle 2, \zeta\rangle\rangle. \\
 & & \quad \mathsf{if}\ \mathit{valid}(cc^{\perp})\ \mathsf{then}\ \beta_1 \oplus^{\perp} \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ \beta_1 \oplus^{\perp} \langle 2, \mathsf{ko}\rangle.\mathbf{0}
\end{array}
$$

Semantics: some reductions

$$
\begin{array}{l}
\mathtt{I} \mid \mathtt{C} \mid \mathtt{S} \mid \mathtt{B} \quad \longrightarrow \\
(\nu s_a)(< \bar{b}^{\perp}[2] \mid s_a[1]!^{\perp}\langle 2, \mathsf{Title}^{\perp}\rangle.\cdots \mid s_a[2]?^{\perp}(1,x^{\perp}).\cdots \mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a : \varepsilon\} >) \\
\quad \longrightarrow \\
(\nu s_a)(< \bar{b}^{\perp}[2] \mid s_a[1]!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\cdots \mid s_a[2]?^{\perp}(1,x^{\perp}).\cdots \mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a : (1,2,\mathsf{Title}^{\perp\downarrow\perp})\} >) \\
\quad \longrightarrow \\
(\nu s_a)(< \bar{b}^{\perp}[2] \mid s_a[1]!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\cdots \mid b^{\perp}[2](\beta_2).\cdots \mid b^{\perp}[1](\beta_1).\cdots,\ \{s_a : \varepsilon\} >) \\
\quad \longrightarrow \\
(\nu s_a)(\nu s_b)(< s_a[1]!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\cdots \mid s_b[2]!^{\perp}\langle\langle 1, s_a[2]\rangle\rangle.\cdots \mid s_b[1]?^{\perp}((2,\zeta)).\cdots,\ \{s_a : \varepsilon, s_b : \varepsilon\} >) \\
\quad \longrightarrow \\
(\nu s_a)(\nu s_b)(< s_a[1]!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.\cdots \mid s_b[2]?^{\perp}((1,\eta)).\cdots \mid s_b[1]?^{\perp}((2,\zeta)).\cdots,\ \{s_a : \varepsilon, s_b : (2,1,s_a[2]^{\perp})\} >) \\
\quad \longrightarrow \\
(\nu s_a)(\nu s_b)(< s_a[1]!^{\perp}\langle 2, \mathsf{CreditCard}^{\top}\rangle.P_{\mathtt{C}} \mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}} \mid s_a[2]?^{\top}(2,cc^{\perp}).P_{\mathtt{B}},\ \{s_a : \varepsilon, s_b : \varepsilon\} >) \\
\quad \longrightarrow \\
(\nu s_a)(\nu s_b)(< P_{\mathtt{C}} \mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}} \mid s_a[2]?^{\top}(1,cc^{\perp}).P_{\mathtt{B}},\ \{s_a : (1,2,\mathsf{CreditCard}^{\top\downarrow\perp}), s_b : \varepsilon\} >)
\end{array}
$$

where the continuation processes $P_{\mathtt{C}}, P_{\mathtt{S}}, P_{\mathtt{B}}$ are defined by:

$$
\begin{array}{l}
P_{\mathtt{C}} = s_a[1]\&^{\perp}(2, \{\mathsf{ok} : s_a[1]?^{\perp}(2, date^{\perp}).\mathbf{0}, \mathsf{ko} : \mathbf{0}\}) \\
P_{\mathtt{S}} = s_b[2]\&^{\perp}(1, \{\mathsf{ok} : \eta \oplus^{\perp} \langle 1, \mathsf{ok}\rangle.\eta!^{\perp}\langle 1, \mathsf{Date}^{\perp}\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^{\perp} \langle 1, \mathsf{ko}\rangle.\mathbf{0}\}) \\
P_{\mathtt{B}} = s_b[1]!^{\perp}\langle\langle 2, s_a[2]\rangle\rangle.\mathsf{if}\ valid(cc^{\perp})\ \mathsf{then}\ s_b[1] \oplus^{\perp} \langle 2, \mathsf{ok}\rangle.\mathbf{0}\ \mathsf{else}\ s_b[1] \oplus^{\perp} \langle 2, \mathsf{ko}\rangle.\mathbf{0}
\end{array}
$$

Typing of processes

The following process P is a fragment of the component S:

$$P = \beta_2\&^{\perp}(1, \{\mathsf{ok} : \eta \oplus^{\perp} \langle 1, \mathsf{ok}\rangle.\eta!^{\perp}\langle \mathsf{Date}^{\perp}, 1\rangle.\mathbf{0}, \mathsf{ko} : \eta \oplus^{\perp} \langle 1, \mathsf{ko}\rangle.\mathbf{0}\})$$

Then we can derive:

$$\vdash_{\perp} P \triangleright \{\beta_2 : \&^{\perp}(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\}), \eta : T'\}$$

and hence

$$\vdash_{\perp} \beta_2!^{\perp}\langle\langle 1, \alpha_2\rangle\rangle.\beta_2?^{\perp}((1,\eta)).P \triangleright \{\alpha_2 : \Rsh\delta; T, \beta_2 : !^{\perp}\langle 1, T\rangle; ?^{\perp}(1, T'); \&^{\perp}(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\})\}$$

where

$$T = ?(1, \mathsf{Number}^{\top\downarrow\perp}); T' \qquad T' = \oplus^{\perp}\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle$$

Typing of configurations

We can type the configuration

$$< P_{\mathtt{C}} \mid s_b[2]?^{\perp}((1,\eta)).P_{\mathtt{S}} \mid s_a[2]?^{\top}(1,cc^{\perp}).P_{\mathtt{B}},\ \{s_a : (1,2,\mathsf{CreditCard}^{\top\downarrow\perp}), s_b : \varepsilon\} >$$

by the configuration environment:

$$< \{s_a[1] : T_{a,1}, s_a[2] : T_{a,2}, s_b[1] : T_{b,1}, s_b[2] : T_{b,2}\} \diamond \{\, s_a[1] : !\langle 2, \mathsf{Number}^{\top\downarrow\perp}\rangle\} >$$

where:

$$
\begin{array}{l}
T_{a,1} = \&^{\perp}(2, \{\mathsf{ok} : ?(2, \mathsf{String}^{\perp\downarrow\perp}); \mathsf{end}, \mathsf{ko} : \mathsf{end}\}) \\
T_{a,2} = ?(1, \mathsf{Number}^{\top\downarrow\perp}); \oplus^{\perp}\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle \\
T_{b,1} = !^{\perp}\langle 2, \oplus^{\perp}\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle; \oplus^{\perp}(2, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\}) \\
T_{b,2} = ?^{\perp}(1, \oplus^{\perp}\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp\downarrow\perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle); \&^{\perp}(1, \{\mathsf{ok} : \mathsf{end}, \mathsf{ko} : \mathsf{end}\})
\end{array}
$$