Session Types for Access and Information Flow Control

Sara Capecchi, Ilaria Castellani, Mariangiola Dezani-Ciancaglini, Tamara Rezk

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Session Types for Access and Information Flow Control

Sara Capecchi — Ilaria Castellani — Mariangiola Dezani-Ciancaglini — Tamara Rezk

N° 7368

August 25, 2010

ISSN 0249-6399 ISRN INRIA/RR--7368--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SOPHIA ANTIPOLIS - MÉDITERRANÉE

Session Types for Access and Information Flow Control

Sara Capecchi*, Ilaria Castellani†, Mariangiola Dezani-Ciancaglini* , Tamara Rezk†

Theme :
Équipe-Projet INDES

Rapport de recherche n° 7368 — August 25, 2010 — 45 pages

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 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.

Key-words: Concurrency, communication-centred computing, session types, access control, secure information flow.

Work partially funded by the INRIA Sophia Antipolis COLOR project MATYSS, by the ANR-SETI-06-010 and ANR-08-EMER-010 grants, and by the MIUR Projects DISCO and IPODS.

* Dipartimento di Informatica, Università di Torino, corso Svizzera 185, 10149 Torino, Italy
† INRIA Sophia Antipolis Méditerranée

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Types de session pour le contrôle d'accès et la sécurité du flux d'information

Résumé : Nous étudions un calcul pour modéliser des sessions à partenaires multiples, avec délégation, où les données et les partenaires sont équipés de niveaux de sécurité. Nous présentons un système de types pour ce calcul, garantissant à la fois la sûreté des sessions, une forme de contrôle d'accès et la sécurité du flux d'information, tout en permettant la déclassification des données lors de leur transmission. En particulier, ce système de types permet de prévenir les fuites d'information dues aux primitives spécifiques du langage, que sont l'ouverture de session, la sélection, le branchement et la délégation. Nous illustrons l'utilisation de notre système de types par une série d'exemples, qui révèlent une interaction intéressante entre les contraintes apparaissant dans les systèmes de types classiques pour la sécurité et celles utilisées dans les types de session pour assurer des propriétés telles que l'absence d'erreurs de communication au cours d'une session et la conformité d'une session à un protocole donné.

Mots-clés : Concurrence, calcul centré sur la communication, types de session, contrôle d'accès, sécurité du flux d'information.

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 harness 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 [24]. We refer to [10] 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, 19]. These papers propose a compiler which, given a multiparty session description, implements cryptographic protocols that guarantee *session execution integrity*. The question of ensuring *access control* in binary sessions has been recently addressed in [16] 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 [12], 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 [20] for a review) and process calculi [11, 13, 15].

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 [22], 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, namely those permitted by the access control policy. Our work reveals an interesting interplay between the constraints of security type systems and those of 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. The Appendix summarises various features of the running example introduced in Section 2.

This paper is the full version of [8], including complete definitions and proofs.

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 its use for declassification. The question of secure information flow will only be marginal here. It will be discussed in Sections 5 and 6.

A client C sends the title of a book to a bookseller S. Then S *delegates* to a bank B both the reception of the credit card number of C and the control of its validity. This delegation is crucial for assuring the secrecy of the credit card number, which should be read by B but not by S. Then B notifies S about the result of the control: for this 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 untrusted party S but *really* - thanks to delegation - to the trusted party B;
4. B delegates back to S the conversation with C;
5. B selects the answer ok or ko for S depending on the validity 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.

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

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

1.	$\mathtt{C} \to \mathtt{S} : \left\langle \mathsf{String}^{\bot} \right\rangle$	
2.	$\mathtt{S} \Rsh \delta$	$\mathtt{S} \to \mathtt{B} : \langle T \rangle$
3.	$\mathtt{C} \to \mathtt{S} : \left\langle \mathsf{Number}^{\top\downarrow\bot} \right\rangle$	
4.	$\mathtt{S} \Rsh \delta$	$\mathtt{B} \to \mathtt{S} : \langle T' \rangle$
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.

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 we omit unnecessary levels on operators and use $\bot$ to mean "public" and $\top$ to mean "secret"). A session is a particular activation of a service, involving a number of parties with predefined roles. Here processes C and S communicate by opening a session on service a, while processes 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 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. Then S waits for a channel ζ from B. Henceforth, S communicates using both channels β_2 and ζ: on channel β_2 he waits for one of the labels ok or ko, which he then forwards to C on ζ, sending also a date if the label is ok. Forgetting 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, $\mathtt{S} \Rsh \delta$ 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, 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, it is easy to see that the *session type* T is:

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

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

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

To formalise access control, we will 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 S in both services a and b, and $\top$ with B in service b. Notice that C may indifferently have level $\top$ or $\bot$, since it 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 $(\mathscr{S},\sqsubseteq)$ 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 $\ell'\leq\ell$ (in case $\ell'=\ell$, we will write simply ℓ instead of $\ell\downarrow\ell$). Let us point out that declassification of services is not allowed in our calculus, both for technical reasons (to be explained in some detail in Section 7) and for lack of evidence of its usefulness. 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 p, q,... to denote the *participants* of a session. In an n-ary session (a session corresponding to an n-ary service) p, 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 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.

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

Table 3: Syntax of processes, expressions and queues.

As in [14], 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 by in Table 3, together with the *runtime syntax* of the calculus (sessions, channels with role, queues etc.), 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 $s : h$, and to replace the channel α in each process $P_\mathtt{p}$ by the channel with role $s[\mathtt{p}]$. This is the only synchronous interaction of the calculus, since all the communications that 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 input/output 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 $\mathbf{Q}$-set H, possibly restricted with respect to service and session names, or a parallel composition $C \| C$ of configurations. In a configuration $(\nu s) < P\ ,\ H >$, all occurrences of $s[\mathtt{p}]$ in P and H and of s in H are bound. By abuse of notation we often write P instead of $< P\ ,\ \emptyset >$.

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 [17]. Among the rules for queues, we have one for commuting independent messages and another one for splitting a message for multiple recipients.

Note that 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 $(\nu a^\ell) C$, 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 e=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

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

$$(\nu r)P \mid Q \equiv (\nu r)(P \mid Q)$$

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

$$\mathsf{def}\ D\ \mathsf{in}\ (\nu r)P \equiv (\nu r)\mathsf{def}\ D\ \mathsf{in}\ P$$

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

$$\mathsf{def}\ D\ \mathsf{in}\ (\mathsf{def}\ D'\ \mathsf{in}\ P) \equiv \mathsf{def}\ D\ \mathsf{and}\ D'\ \mathsf{in}\ P$$

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

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

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

$$P \equiv Q \text{ and } H \equiv K \quad \Rightarrow \quad <P\ ,\ H> \equiv <Q\ ,\ K>$$

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

$$(\nu rr')C\ \equiv\ (\nu r'r)C$$

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

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

Table 4: Structural equivalence.

is why the value needs to be recorded with both levels in the queue. Recall that in case v is a service, then $\ell' = \ell$ (services cannot be declassified).

The rules [Rec], [DelRec] and [Branch] perform the corresponding complementary operations. Rules [ScopP] and [ScopC] are standard contextual rules. In the latter, Barendregt convention ensures that the names in $\tilde{s}$ are disjoint from those in $\tilde{r}$ and do not appear in C''. 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:

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

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

$$P \longrightarrow P' \quad \Rightarrow \quad \mathsf{def}\ D\ \mathsf{in}\ P \longrightarrow \mathsf{def}\ D\ \mathsf{in}\ P'$$
[Defin]

$$P \longrightarrow P' \quad \Rightarrow \quad (\nu a^{\ell})P \longrightarrow (\nu a^{\ell})P'$$
[ScopP]

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

Table 5: Reduction rules.

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

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

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

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

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

5 Information Flow Security in Sessions

We turn now to the question of ensuring *secure information flow* [9] within sessions. We shall be interested in the property of *noninterference* (NI) [12], combined with a limited form of *declassification* [1], which may only take place during the transmission of a basic value. The property of NI requires that there is no flow of information from objects of a given level to objects of lower or incomparable level [25, 23, 20]. 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 $v^{\ell\downarrow\ell'}$, λ^{ℓ} or $s[\mathtt{p}]^{\ell}$. In the rest of this section, we will focus on the observation model, which will be based on bisimulation as is now standard for concurrent processes [23, 21].

We assume that the observer can see the content of messages in session queues. To fix ideas, one may view the observer as a kind of buffer through which messages may transit while reaching or leaving a session queue. We do not want to go as far as allowing an observer to take part in a session, since that could affect the behaviour of other processes. In other words, we assume a passive rather than an active observer.

What matters for security is observation relative to a given set of levels. Given a downward-closed subset $\mathscr{L}$ of $\mathscr{S}$, a $\mathscr{L}$-observer will only be able to see messages whose level belongs to $\mathscr{L}$. A notion of $\mathscr{L}$-equality $=_{\mathscr{L}}$ on **Q**-sets is then introduced, representing indistinguishability of **Q**-sets by a $\mathscr{L}$-observer. Based on $=_{\mathscr{L}}$, a notion of $\mathscr{L}$-bisimulation $\simeq_{\mathscr{L}}$ will formalise indistinguishability of processes by a $\mathscr{L}$-observer.

Formally, a queue $s : h$ is $\mathscr{L}$-observable if it contains some message with a 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 messages with a level in $\mathscr{L}$. This equality is based on a $\mathscr{L}$-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:*

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

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

$$(\mathrm{p},\Pi,\vartheta)\Downarrow\mathscr{L} = \begin{cases}(\mathrm{p},\Pi,\vartheta) & \text{if } lev_\downarrow(\vartheta)\in\mathscr{L},\\ \varepsilon & \text{otherwise.}\end{cases}$$
$$\varepsilon\Downarrow\mathscr{L} = \varepsilon$$
$$(m\cdot h)\Downarrow\mathscr{L} = m\Downarrow\mathscr{L}\cdot h\Downarrow\mathscr{L}$$
$$\emptyset\Downarrow\mathscr{L} = \emptyset$$
$$(H\cup\{s:h\})\Downarrow\mathscr{L} = \begin{cases}H\Downarrow\mathscr{L}\cup\{s:h\Downarrow\mathscr{L}\} & \text{if } h\Downarrow\mathscr{L}\neq\varepsilon,\\ H\Downarrow\mathscr{L} & \text{otherwise.}\end{cases}$$

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 $H\Downarrow\mathscr{L} = K\Downarrow\mathscr{L}$.*

The idea of $\mathscr{L}$-bisimulation is to test processes by running them in conjunction with $\mathscr{L}$-equal queues. Now, when testing processes with queues we cannot allow arbitrary queues, otherwise we could get insecurity for 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 $P = s[1]!\langle 2,\mathsf{true}^\perp\rangle.\mathbf{0}$ does not react in the same way when combined with the two **Q**-sets $\{s:\varepsilon\} =_{\{\perp\}} \emptyset$. Indeed, while P reduces when combined with $\{s:\varepsilon\}$, it is stuck when combined with $\emptyset$. Formally:

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

Hence a $\mathscr{L}$-observer with $\mathscr{L} = \{\perp\}$ would detect an insecurity by looking at the resulting **Q**-sets $\{s:(1,2,\mathsf{true}^\perp)\} \neq_{\{\perp\}} \emptyset$.

A natural requirement is that the **Q**-sets always 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 which occurs in P has a corresponding queue in H. We use $\mathrm{fsn}(P)$ to denote the set of free session names which occur in P.

Definition 5.4 *A configuration $\langle P,H\rangle$ is* saturated *if P does not contain restrictions of session names and all session names in P have corresponding queues in H, i.e. $s\in fsn(P)$ implies $s\in qn(H)$.*

It is easy to check that saturation is preserved by reduction, since only rule [Link] creates new sessions, always 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 is initial *if the process is a closed user process, and the* **Q***-set is empty.*

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 happens with inputs. For example if

$$\begin{array}{lcl} Q & = & s[2]?(1,x^{\perp}).\mathbf{0}, \\ H_1 & = & \{s:(1,2,\mathsf{true}^{\perp})\}, \\ H_2 & = & \{s:(1,2,\mathsf{true}^{\top})\cdot(1,2,\mathsf{true}^{\perp})\} \end{array}$$

we have $< Q\ ,\ H_1 > \longrightarrow < \mathbf{0}\ ,\ \{s:\varepsilon\} >$, while $< Q\ ,\ H_2 > \not\longrightarrow$. Note that $H_1 =_{\{\perp\}} H_2$ and $\{s:\varepsilon\} \neq_{\{\perp\}} H_2$. The problem here is that the top level message is transparent for the $\{\perp\}$-equality, but it prevents the process from reading the bottom level message. Note that the problem comes from the fact that both messages have the same receiver and the same sender, since for example if $H_3 = \{s:(3,2,\mathsf{true}^{\top})\cdot(1,2,\mathsf{true}^{\perp})\}$ then $< Q\ ,\ H_3 > \longrightarrow < \mathbf{0}\ ,\ \{s:(3,2,\mathsf{true}^{\top})\} >$, thanks to the structural equivalence on queues defined in Table 4.

Our way out for this problem is to ask that the security levels of messages with the same sender and common receivers never decrease along a sequence. In this way we allow the **Q**-sets H_1 and H_3, but forbid H_2. More formally:

Definition 5.7 *A queue 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 the queue and* $\Pi_1 \cap \Pi_2 \neq \emptyset$*. A* **Q***-set is* monotone *if it contains only monotone queues.*

Since a queue can contain the same message more than once, a message can precede an identical message. Also, it may be that each of two messages precedes the other, since an occurrence of one of them may be placed between two occurrences of the other. Note that in this case monotonicity implies that the two messages have the same level.

Notice that 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:

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

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 the process $a[2](\beta).\beta?(1,x^{\top}).\beta?(1,y^{\perp}).\mathbf{0}$, since a high input will not be allowed to be followed by a low action, as will be explained in detail in the next section. In particular, the process $s[1]!\langle 2,\mathsf{true}^{\top}\rangle.s[1]!\langle 2,\mathsf{true}^{\perp}\rangle.\mathbf{0}$ will also 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 the desired process bisimulation. A relation on processes is a $\mathscr{L}$*-bisimulation* if, whenever two related processes are coupled with $\mathscr{L}$-equal monotone **Q**-sets yielding saturated configurations, both the relation and the $\mathscr{L}$-equality of **Q**-sets are preserved by execution:

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 (in case of extrusion of the private service a^ℓ) or a fresh session name s (if the last applied rule is [Link]), and in the last 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.

Security is now defined, as usual, to be the reflexive kernel of $\simeq_{\mathscr{L}}$:

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

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 $< P , H_P >$.

6 Examples of Information Flow Security in Sessions

In this section we illustrate the various kinds of flow that can occur in our calculus, through 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 in all other objects. In queues, we will use v^ℓ as a shorthand for $v^{\ell\downarrow\ell}$. For the sake of simplicity, we assume here just two security levels $\bot$ and $\top$ (also called low and high). In all examples, we suppose $H_1 = \{s : (1, 2, \mathsf{true}^\top)\}$ and $H_2 = \{s : (1, 2, \mathsf{false}^\top)\}$.

6.1 High input should not be followed by low actions

A simple example of insecure flow, which is not specific to our calculus but arises in all process calculi with values and a conditional construct, is the following (assuming session s has four participants):

$$s[2]?(1, x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]!\langle 3, \mathsf{true}^\top\rangle.\mathbf{0}\ \mathsf{else}\ \mathbf{0} \\ \mid\ s[3]?(2, z^\top).s[3]!\langle 4, \mathsf{true}^\bot\rangle.\mathbf{0} \mid\ s[4]?(3, y^\bot).\mathbf{0}$$

This process is insecure because, depending on the high value received for $x^\top$ on channel $s[2]$, that is, on whether the **Q**-set is H_1 or H_2, the low value $\mathsf{true}^\bot$ will be emitted or not on channel $s[3]$, leading to $H_1' = \{s : (3, 4, \mathsf{true}^\bot)\} \neq_{\mathscr{L}} H_2' = \{s : \varepsilon\}$ if $\mathscr{L} = \{\bot\}$. This shows that a high input should not be followed by a low output. Note that the reverse is not true, since output is not blocking: if we swapped the polarities of input and output in the third participant (and adjusted them accordingly in the other participants), then the resulting process would be secure.

Let us point out that this process is not typable in a classical session type system, since the session types of the conditional branches are not the same (as we shall see in the next section, this is a classical requirement in session types). However, it would become typable if the second branch of the conditional were replaced by the deadlocked process $(\nu b)b[1](\beta_1).s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0}$. The expert reader will notice that by adding to our type system the interaction typing of [2] (which enforces global progress) we would rule out also this second process. On the other hand, the interaction typing does not prevent deadlocks due to inverse session calls, as for instance:

$$\begin{array}{ll}\bar{b}[2] & \mid\ b[1](\beta_1).c[1](\gamma_1).s[2]!\langle 3,\mathsf{true}^\top\rangle.\mathbf{0}\\ \bar{c}[2] & \mid\ c[2](\gamma_2).b[2](\beta_2).\mathbf{0}\end{array}$$

Clearly, this deadlock could be used to implement the insecure flow in our example.

6.2 Need for levels on services.

Consider the following process:

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

This process is insecure because, depending on the high value received for $x^\top$, it will initiate or not a session on service b, which performs a low value exchange. To rule out this kind of leak we annotate service names with security levels which 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 appears in the branch of a $\top$-conditional, and hence it will not be allowed to perform the exchange of the value $\mathsf{true}^\perp$.

6.3 Need for levels on selection and branching

Consider the following process:

$$\begin{array}{l}s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]\oplus\langle 3,\lambda\rangle.\mathbf{0}\ \mathsf{else}\ s[2]\oplus\langle 3,\lambda'\rangle.\mathbf{0}\\ \mid\ s[3]\&(2,\{\lambda : s[3]!\langle 4,\mathsf{true}^\perp\rangle.\mathbf{0},\lambda' : s[3]!\langle 4,\mathsf{false}^\perp\rangle.\mathbf{0}\})\\ \mid\ s[4]?(3,y^\perp).\mathbf{0}\end{array}$$

This process is insecure because a selection in one participant, which depends on a high value, causes the corresponding branching participant to emit two different low values. 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 branches.

6.4 Need for levels on delegated channels.

Consider the following process:

$$\begin{array}{l}s[2]?(1,x^\top).\mathsf{if}\ x^\top\ \mathsf{then}\ s[2]!\langle\langle 3,s'[1]\rangle\rangle.s[2]!\langle\langle 4,s''[1]\rangle\rangle.\mathbf{0}\ \mathsf{else}\ s[2]!\langle\langle 3,s''[1]\rangle\rangle.s[2]!\langle\langle 4,s'[1]\rangle\rangle.\mathbf{0}\\ \mid\ s[3]?((2,\eta)).\eta!\langle 2,\mathsf{true}^\perp\rangle.\mathbf{0}\mid\ s[4]?((2,\eta')).\eta'!\langle 2,\mathsf{false}^\perp\rangle.\mathbf{0}\\ \mid\ s'[2]?(1,x^\perp).\mathbf{0}\ \mid\ s''[2]?(1,y^\perp).\mathbf{0}\end{array}$$

This process is insecure because, depending on the high value received for $x^\top$, the participants 3 and 4 of s will be delegated to participate in sessions s' and s'', or viceversa, feeding the queues of s' and s'' with different low values. 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.

6.5 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 level and their declassified level. The reason for recording also the declassified 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 5.2). 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 $v^{\ell\downarrow\ell'}$, λ^{ℓ} or $s[\mathtt{p}]^{\ell}$.

6.6 Examples of secure processes

It is easy to check that both the processes $P = s[1]!\langle 2,\mathtt{true}^{\top}\rangle.s[1]!\langle 2,\mathtt{true}^{\bot}\rangle.\mathbf{0}$ and $Q = s[2]?(1,x^{\top}).s[2]?(1,y^{\bot}).\mathbf{0}$ are secure, as well as their parallel composition $P \mid Q$. We let the reader verify that our $\mathtt{C},\mathtt{S},\mathtt{B}$ process of Section 4.1 is also 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:

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

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 $\{\mathtt{q}\}$. We use S^{ℓ} as short for $S^{\ell\downarrow\ell}$, called a *trivial declassification*.

Type $\mathtt{p} \to \Pi : \{\lambda_i : G_i\}^{\ell}_{i\in I}$, 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 $\mathtt{p} \,\text{↱}\delta.G$ says that the role of $\mathtt{p}$ is delegated to another participant; this construct does not appear in the original global types of [14]. 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.

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

The condition $\ell \leq \bigsqcap_{i\in I} \mathsf{M}(G_i)$ 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*
		|	$!\langle \Pi, \langle \mathsf{L}, G \rangle^{\ell} \rangle;T$	*servsend*
		|	$!^{\ell}\langle \mathtt{q}, T \rangle;T'$	*delsend*
		|	$\oplus^{\ell}\langle \Pi, \{\lambda_i : T_i\}_{i\in I} \rangle$	*selection*
		|	$\mu\mathbf{t}.T$	*recursive*
		|	$\text{↱}\delta;T$	*delegation*
		|	$?(\mathtt{p}, S^{\ell\downarrow\ell'});T$	*receive*
		|	$?(\Pi, \langle \mathsf{L}, G \rangle^{\ell});T$	*servreceive*
		|	$?^{\ell}(\mathtt{p}, T);T'$	*delreceive*
		|	$\&^{\ell}(\mathtt{p}, \{\lambda_i : T_i\}_{i\in I})$	*branching*
		|	$\mathbf{t}$	*variable*
		|	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 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 6.3. Recursive types are considered modulo fold/unfold.

We say that $!\langle \Pi, S^{\ell\downarrow\ell'}\rangle$, $!\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle$, $!^{\ell}\langle \mathtt{q}, T\rangle$, $?(\mathtt{p}, S^{\ell\downarrow\ell'})$, $?(\Pi, \langle \mathsf{L}, G\rangle^{\ell})$, $?^{\ell}(\mathtt{p}, T)$ 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 [14]. The *projection of G onto* q, denoted $(G \upharpoonright \mathtt{q})$, gives participant q's view of the protocol described by G.

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

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

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

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

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

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

$$(\mu \boldsymbol{t}.G') \upharpoonright \mathtt{q} = \mu \boldsymbol{t}.(G' \upharpoonright \mathtt{q}) \qquad \boldsymbol{t} \upharpoonright \mathtt{q} = \boldsymbol{t} \qquad \mathsf{end} \upharpoonright \mathtt{q} = \mathsf{end}$$

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.

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*):

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

Notice that in the definition of $\mathsf{J}(T)$ we cannot omit ℓ in the cases of selection and branching, because we could have $\mathsf{M}(T) \not\leq \mathsf{J}(T)$. For example $\mathsf{M}(\mu \mathbf{t}.\mathbf{t}) = \top$ and $\mathsf{J}(\mu \mathbf{t}.\mathbf{t}) = \bot$.

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:

A safety global type $\langle \mathsf{L}, G\rangle^{\ell}$ *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 $\Gamma, x^{\ell} : -$ (respectively $\Gamma, a^{\ell} : \langle \mathsf{L}, G\rangle^{\ell'}$ and $\Gamma, X : S^{\ell}\ T$) only if $x^{\ell''}$ (respectively $a^{\ell''}$ and X) does not belong to the domain of Γ, that is, we don't want Γ to contain the same variable or service name with two different security levels.

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:

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

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:

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

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.

Rule $\lfloor$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 $\ell_1 \sqcup \ell_2$ using rule $\lfloor$SUBS$\rfloor$.

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

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

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

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

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

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

$$\frac{\Gamma, x^{\ell'} : S^{\ell'} \vdash_{\ell'} P \triangleright \Delta, c : T \qquad \ell' \leq \ell}{\Gamma \vdash_{\ell'} c?^{\ell}(\mathsf{p}, x^{\ell'}).P \triangleright \Delta, c : ?(\mathsf{p}, S^{\ell \downarrow \ell'}); T} \lfloor \textsc{Rcv} \rfloor$$

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

$$\frac{\Gamma, x^\ell : \langle \mathsf{L}, G \rangle^\ell \vdash_\ell P \triangleright \Delta, c : T}{\Gamma \vdash_\ell c?^{\ell}(\mathsf{p}, x^\ell).P \triangleright \Delta, c : ?(\mathsf{p}, \langle \mathsf{L}, G \rangle^\ell); T} \lfloor \textsc{ServRcv} \rfloor$$

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

$$\frac{\Gamma \vdash_\ell P \triangleright \Delta, c : T, \alpha : T' \quad \ell = \mathsf{M}(T')}{\Gamma \vdash_\ell c?^{\ell}((\mathsf{p}, \alpha)).P \triangleright \Delta, c : ?(\mathsf{p}, T'); T} \lfloor \textsc{Rec} \rfloor$$

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

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

Table 8: Typing rules for processes.

$\lfloor$IF$\rfloor$ This rule requires that the two branches P, Q of a conditional be typed with the same process environment, and with the same security level as the tested expression. So the level of the conditional is the level of the tested expression, which cannot be modified, while process P and Q could also have higher levels.

$\lfloor$NRES$\rfloor$ This rule types the restriction of a process: the use of the restricted name a 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 $\ell'' \leq \sqcap \ell'$.

$\lfloor$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 p-th participant is the 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

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

$\lfloor$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. 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 both the process B and the session b between S and B.

⌊SERVSEND⌋ 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 ⌊SEND⌋ the hypothesis $\ell' \leq \ell$ is not really constraining, since P can always be downgraded to ℓ' using rule ⌊SUBS⌋.

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

⌊DELEG⌋ 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.

⌊SREC⌋ This rule is the dual of the ⌊DELEG⌋ rule.

⌊SEL⌋ 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.

⌊BRANCH⌋ This rule is the dual of the ⌊SEL⌋ 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 $c : T \in \Delta$ and $\tau_1 \in \{S_1^{\ell_1 \downarrow \ell}, \langle \mathsf{L}_1, G_1\rangle^{\ell}, ?^{\ell}(\mathsf{p}_1, T_1)\}$ and $\tau_2 \in \{S_2^{\ell_2 \downarrow \ell'}, \langle \mathsf{L}_2, G_2\rangle^{\ell'}, ?^{\ell'}(\mathsf{p}_2, T_2)\}$. If τ_1 precedes τ_2 in T, 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:

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

we can derive:

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

and hence:

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

where

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

$T' = \oplus^{\perp}\langle 1, \{\mathsf{ok} : !\langle 1, \mathsf{String}^{\perp \downarrow \perp}\rangle; \mathsf{end}, \mathsf{ko} : \mathsf{end}\}\rangle$

7.6 Typing queues and Q-sets

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

$$\begin{array}{rrcll} \text{Message} & \mathbf{T} & ::= & !\langle \Pi, S^{\ell\downarrow\ell'}\rangle & \textit{message value send} \\ & & \mid & !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle & \textit{message service send} \\ & & \mid & !^{\ell}\langle \mathsf{q}, T\rangle & \textit{message delegation} \\ & & \mid & \oplus^{\ell}\langle \Pi, \lambda\rangle & \textit{message selection} \\ & & \mid & \mathbf{T};\mathbf{T}' & \textit{message sequence} \end{array}$$

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* $!^{\ell}\langle \Pi, T\rangle$ express the communication to all $\mathsf{p} \in \Pi$ of a value of type S^{ℓ}declassified to level ℓ', of a service of type $\langle \mathsf{L}, G\rangle^{\ell}$ and of a channel of type T with $\ell = \mathsf{M}(T)$, respectively. 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 $\oplus^{\bot}\langle \{1,3\}, \mathsf{ok}\rangle$ is the message type for the message $(2, \{1,3\}, \mathsf{ok}^{\bot})$.

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 $Z \in \{T, S^{\ell\downarrow\ell'}, \langle \mathsf{L}, G\rangle^{\ell}, \lambda\}$).

- $\natural\langle \Pi, Z\rangle;\ \natural'\langle \Pi', Z\rangle;\mathbf{T} \approx\ \natural'\langle \Pi', Z\rangle;\ \natural\langle \Pi, Z\rangle;\mathbf{T} \quad$ if $\Pi \cap \Pi' = \emptyset$
- $\natural\langle \Pi, Z\rangle;\mathbf{T} \approx\ \natural\langle \Pi', Z\rangle;\ \natural\langle \Pi'', Z\rangle;\mathbf{T} \quad$ if $\Pi = \Pi' \cup \Pi'', \Pi' \cap \Pi'' = \emptyset$

Table 9: Equivalence relation on message types.

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

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

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

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

$$\frac{\Gamma \vdash s : h \triangleright \Theta \qquad \Theta \approx \Theta'}{\Gamma \vdash s : h \triangleright \Theta'}\ \lfloor \textsc{QCong} \rfloor$$

Table 10: Typing rules for single queues.

Typing judgments for queues have the shape

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

$$\frac{}{\Gamma \vdash_\emptyset \emptyset \triangleright \emptyset}\ \lfloor \text{QSINIT} \rfloor \qquad \frac{\Gamma \vdash_\Sigma H \triangleright \Theta \qquad \Gamma \vdash s : h \triangleright \Theta'}{\Gamma \vdash_{\Sigma,s} H \cup \{s : h\} \triangleright \Theta, \Theta'}\ \lfloor \text{QSUNION} \rfloor$$

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

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

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

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

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

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. The composition ";" of queue environments is defined by:

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

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 $\textit{dom}(\Theta) \cap \textit{dom}(\Theta') = \emptyset$, 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*.

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

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

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 7.2 *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:*

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

Configuration types can be projected on a participant $\mathtt{q}$.

Definition 7.3 *The* projection of the configuration type $\mathscr{T}$ onto $\mathtt{q}$*, denoted by $\mathscr{T}\upharpoonright\mathtt{q}$, is defined by:*

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

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

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

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

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

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

$$(?^{\ell}(\mathtt{p},T);T')\upharpoonright\mathtt{q} = \begin{cases} ?^{\ell}T;T'\upharpoonright\mathtt{q} & \text{if } \mathtt{q}=\mathtt{p}\\ T'\upharpoonright\mathtt{q} & \text{otherwise.}\end{cases}$$

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

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

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

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

Definition 7.4 *The* duality relation *between projections of configuration types is the minimal symmetric relation which satisfies:*

$$\mathsf{end}\bowtie\mathsf{end} \qquad \boldsymbol{t}\bowtie\boldsymbol{t} \qquad T\bowtie T' \implies \mu\boldsymbol{t}.T\bowtie\mu\boldsymbol{t}.T'$$
$$\forall i\in I\ T_i\bowtie T_i' \implies \oplus^{\ell}\{\lambda_i:T_i\}_{i\in I}\bowtie\&^{\ell}\{\lambda_i:T_i'\}_{i\in I}$$
$$\mathscr{T}\bowtie T\implies !S^{\ell\downarrow\ell'};\mathscr{T}\bowtie ?S^{\ell};T \qquad \mathscr{T}\bowtie T\implies !\langle\mathsf{L},G\rangle^{\ell};\mathscr{T}\bowtie ?\langle\mathsf{L},G\rangle^{\ell};T$$
$$\mathscr{T}\bowtie T\implies !^{\ell}T';\mathscr{T}\bowtie ?^{\ell}T';T$$
$$\exists i\in I\ \lambda=\lambda_i\ \&\ \mathscr{T}\bowtie T_i\implies\oplus^{\ell}\lambda;\mathscr{T}\bowtie\&^{\ell}\{\lambda_i:T_i\}_{i\in I}$$

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

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

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

Table 12: Typing rules for configurations.

The above definitions are needed to state coherence of 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 7.5 *A 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*

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

Typing rules assure that configurations are always typed with coherent environments. 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 $\Delta \setminus s$ and $\Theta \setminus s$ are defined as follows:

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

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:

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

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

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

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

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

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

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.

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. By induction on the definition of $\Rightarrow$ (Definition 7.6).

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* $\Gamma \vdash_\ell$ if e then P 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* $\ell = \ell' \sqcap \mathsf{M}(T)$ *and* Δ' end *only.*
6. *If* $\Gamma \vdash_{\ell'} \mathsf{def}\ X(x^\ell, \alpha) = P$ 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* $\ell'' \leq \ell' \leq \ell$.
   (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* $\ell'' \leq \ell' = \ell$.
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* $\ell'' = \ell' \leq \ell$.
    (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* $\ell = \ell''$.
11. *If* $\Gamma \vdash_\ell c!^{\ell'}\langle\langle \mathtt{q}, c'\rangle\rangle.P \triangleright \Delta$*, then* $\Delta = \Delta', c : !^{\ell'}\langle \mathtt{q}, T'\rangle; T, c' : \overleftarrow{\delta}; T'$ $\Gamma \vdash_\ell P \triangleright \Delta', c : T$ *and* $\ell \leq \ell'$.
12. *If* $\Gamma \vdash_{\ell'} c?^\ell((\mathtt{p}, \alpha)).P \triangleright \Delta$*, then* $\Delta = \Delta', c : ?(\mathtt{p}, T'); T$ *and* $\Gamma \vdash_{\ell'} P \triangleright \Delta', c : T, \alpha : T'$ *and* $\ell = \ell'$.
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* $\ell = \ell'$.
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* $\ell = \ell'$.

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* $\Theta = \Theta'; \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}$ *and* $\Gamma \vdash s : h \triangleright \Theta'$ *and* $\Gamma \vdash v^{\ell} : S^{\ell}$.

3. *If* $\Gamma \vdash s : h \cdot (\mathtt{p}, \Pi, a^{\ell}) \triangleright \Theta$, *then* $\Theta = \Theta'; \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\}$, $\Gamma \vdash s : h \triangleright \Theta'$ *and* $\Gamma \vdash a : \langle \mathsf{L}, G\rangle^{\ell}$.

4. *If* $\Gamma \vdash s : h \cdot (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']) \triangleright \Theta$, *then* $\Theta = \Theta'; s'[\mathtt{p}'] : T; \{s[\mathtt{p}] : !\langle \mathtt{q}, T\rangle\}$ *and* $\Gamma \vdash s : h \triangleright \Theta'$.

5. *If* $\Gamma \vdash s : h \cdot (\mathtt{p}, \Pi, \lambda^{\ell}) \triangleright \Theta; \{s[\mathtt{p}] : \oplus^{\ell}\langle \Pi, \lambda\rangle\}$, *then* $\Theta = \Theta'; \{s[\mathtt{p}] : \oplus^{\ell}\langle \Pi, \lambda\rangle\}$ *and* $\Gamma \vdash s : h \triangleright \Theta'$.

6. *If* $\Gamma \vdash_{\Sigma} \emptyset \triangleright \Theta$, *then* $\Sigma = \Theta = \emptyset$.

7. *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* $\Sigma = \Sigma_1 \cup \Sigma_2$ *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* $\Sigma_1 \cap \Sigma_2 = \emptyset$ *and* $< \Delta \diamond \Theta >$ *is coherent.*

3. *If* $\Gamma \vdash_{\Sigma} (\nu s) C \triangleright < \Delta \diamond \Theta >$, *then* $\Sigma = \Sigma' \backslash s$ *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. The proof is standard, by induction on the length of queues.

Lemma 9 *1. If* $\Gamma \vdash s : (\mathtt{p}, \Pi, v^{\ell\downarrow\ell'}) \cdot h \triangleright \Theta$, *then* $\Theta = \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}; \Theta'$ *and* $\Gamma \vdash s : h \triangleright \Theta'$ *and* $\Gamma \vdash v^{\ell} : S^{\ell}$.

2. *If* $\Gamma \vdash s : (\mathtt{p}, \Pi, a^{\ell}) \cdot h \triangleright \Theta$, *then* $\Theta = \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}, G\rangle^{\ell}\rangle\}; \Theta'$ *and* $\Gamma \vdash s : h \triangleright \Theta'$ *and* $\Gamma \vdash a^{\ell} : \langle \mathsf{L}, G\rangle^{\ell}$.

3. *If* $\Gamma \vdash s : (\mathtt{p}, \mathtt{q}, s'[\mathtt{p}']) \cdot h \triangleright \Theta$, *then* $\Theta = \{s[\mathtt{p}] : !\langle \mathtt{q}, T\rangle\}; \Theta'$ *and* $\Gamma \vdash s : h \triangleright \Theta'$ *and* $s'[\mathtt{p}'] : T \in \Theta$.

4. *If* $\Gamma \vdash s : (\mathtt{p}, \Pi, \lambda) \cdot h \triangleright \Theta$, *then* $\Theta = \{s[\mathtt{p}] : \oplus^{\ell}\langle \Pi, \lambda\rangle\}; \Theta'$ *and* $\Gamma \vdash s : h \triangleright \Theta'$.

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* $< \Delta \diamond \Theta > \;\Rightarrow\; < \Delta' \diamond \Theta' >$.

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

If $\Gamma \vdash_\Sigma C \triangleright < \Delta \diamond \Theta >$ *and* $C \longrightarrow C'$, *then* $\Gamma \vdash_\Sigma C' \triangleright < \Delta' \diamond \Theta' >$ *with* $< \Delta \diamond \Theta > \;\Rightarrow\; < \Delta' \diamond \Theta' >$.

By induction on the derivation of $C \longrightarrow C'$, with a case analysis on the final rule. We only consider some paradigmatic cases.

[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(6) we get $\Sigma = \Theta = \emptyset$.

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

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

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

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

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

which implies by Lemma 11(3)

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

Applying rule $\lfloor$Conc$\rfloor$ and $\lfloor$QInit$\rfloor$ we derive

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

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

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

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

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

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

This concludes the proof for this case, since $< \Delta \diamond \Theta > \ \Rightarrow \ < \Delta \diamond \Theta >$.

[Send]

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

By hypothesis we have

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

which implies by Lemma 8(1)

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

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

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

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

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

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

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(6) and (7) we get $\Sigma = \{s\}$ and

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

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

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

which implies by rules $\lfloor$QSInit$\rfloor$ and $\lfloor$QSUnion$\rfloor$:

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

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

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

This concludes the proof, since

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

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(6) and (7) 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(6) and (7) 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 $\Theta = \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}; \Theta'$ and

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

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

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

The coherence of $< \Delta \diamond \Theta >$ gives $S = S_0$, 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(6) and (7) 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) implies $\Gamma \vdash s : h \triangleright \Theta'$ and either $\Theta = \{s[\mathtt{p}] : !\langle \Pi, S^{\ell\downarrow\ell'}\rangle\}; \Theta'$ and (23) or $\Theta = \{s[\mathtt{p}] : !\langle \Pi, \langle \mathsf{L}_0, G_0^{\ell}\rangle\rangle\}; \Theta'$ and (24).
The coherence of $< \Delta \diamond \Theta >$ gives $\mathsf{L}_0 = \mathsf{L}$ 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 7.5) 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 $s[\mathtt{q}] \in dom(\Delta) \cup dom(\Theta)$.*

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]. If the last rule is one of the output rules [Send], [Deleg] or [Sel], then $< \Delta \diamond \Theta >$ is obtained from $< \Delta_{n-1} \diamond \Theta_{n-1} >$ by shifting an element of Δ_{n-1} to Θ_{n-1}. Then we conclude by induction, since $s[\mathtt{q}] \in dom(\Delta) \cup dom(\Theta)$ if and only if $s[\mathtt{q}] \in dom(\Delta_{n-1}) \cup dom(\Theta_{n-1})$. If the last rule applied is one of the input rules [Rcv], [Srec] or [Branch], then if an input type for $s[\mathtt{p}]$ involving $\mathtt{q}$, for instance $?(\mathtt{q}, S^{\ell \downarrow \ell'})$, is erased from Δ_{n-1}, a matching message type $!\langle \mathtt{p}, S^{\ell \downarrow \ell'} \rangle$ for $s[\mathtt{q}]$ is erased at the same time from Θ_{n-1}, and thus we may conclude using induction.

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 $\vartheta_i = v_i^{\ell_i \downarrow \ell'_i}$ 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

To prove that our type system ensures access control it is convenient to distinguish two classes of operational rules: *computational rules*, which apply to a redex and correspond to an actual computation step (these are the first nine rules in Table 5), and *contextual rules*, which allow a term to be reduced within a static context (these are the last three rules in Table 5) [18].

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:

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

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]. If the last rule is one of the output rules [Send], [Deleg] or [Sel], then $< \Delta \diamond \Theta >$ is obtained from $< \Delta' \diamond \Theta' >$ by shifting an element of Δ' to Θ', and thus $\mathsf{J}(< \Delta \diamond \Theta >) = \mathsf{J}(< \Delta' \diamond \Theta' >)$. If the last rule applied is one of the input rules [Rcv], [Srec] or [Branch], then an input type is erased from Δ' and a message type is erased from Θ', hence $\mathsf{J}(< \Delta \diamond \Theta >) \leq \mathsf{J}(< \Delta' \diamond \Theta' >)$ 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 $\vartheta = v^{\ell \downarrow \ell'}$, 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 7.2) we have then $\ell \leq \mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{q}]))$ and by Lemma 15 we get $\mathsf{J}(< \Delta \diamond \Theta > (s[\mathtt{q}])) \leq \mathsf{J}(G \upharpoonright \mathtt{q})$, where G is the global type associated with the service a. Then we may conclude, since the well-formedness of safety global types implies $\mathsf{J}(G \upharpoonright \mathtt{q}) \leq \mathsf{L}(\mathtt{q})$.

8.4 Soundness for Noninterference

We start by showing the confinement lemma, which says that a process of level ℓ can only add or remove messages of level greater than or equal to ℓ in the queues. Note that, for any level $\ell \in \mathscr{S}$, the complement $\mathscr{L} = \{\ell' \mid \ell \not\leq \ell'\}$ of the upward-closure of ℓ is a downward-closed set containing all levels lower than or incomparable with ℓ.

Lemma 17 (Confinement)
Let $\Gamma \vdash_\ell P \triangleright \Delta$ and $\mathscr{L} = \{\ell' \mid \ell \not\leq \ell'\}$. Then, for any **Q***-set H such that $\langle P, H\rangle$ is typable:*

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

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. 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, ℓ_2, P'' such that $\ell \leq \ell_2 \leq \ell_1$ and $P = s[\mathtt{q}]?^{\ell_1}(\mathtt{p}, x^{\ell_2}).P''$. Then $P' = P''\{v^{\ell_2}/x^{\ell_2}\}$ and the reduction is:
  $< s[\mathtt{q}]?^{\ell_1}(\mathtt{p}, x^{\ell_2}).P''\ ,\ s : (\mathtt{p}, \mathtt{q}, v^{\ell_1\downarrow\ell_2}) \cdot h > \longrightarrow < P''\{v^{\ell_2}/x^{\ell_2}\}\ ,\ s : h >$
  As in the previous case, $\ell \leq \ell_2$ implies $\ell_2 \notin \mathscr{L}$. Hence $s : (\mathtt{p}, \mathtt{q}, v^{\ell_1\downarrow\ell_2}) \cdot h =_{\mathscr{L}} s : h$.

- [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 $P = s[\mathrm{p}] \oplus^\ell \langle \Pi, \lambda \rangle.P'$ and the reduction is:
$$< s[\mathrm{p}] \oplus^\ell \langle \Pi, \lambda \rangle.P' \ , \ s : h > \longrightarrow < P' \ , \ s : h \cdot (\mathrm{p}, \Pi, \lambda^\ell) >$$
Again, since $\ell \notin \mathscr{L}$, we may conclude that $s : h =_{\mathscr{L}} s : h \cdot (\mathrm{p}, \Pi, \lambda^\ell)$.

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 *syntactic $\mathscr{L}$-highness* the complement of non $\mathscr{L}$-boundedness, which amounts to typability with a level not belonging to $\mathscr{L}$.

A closely related property is that of *semantic $\mathscr{L}$-highness*, the property of processes that do not manipulate any messages of level in $\mathscr{L}$ during their execution. Formally:

Definition 8.4 (Semantic $\mathscr{L}$-highness)
The set of semantically high processes with respect to $\mathscr{L}$, denoted $\mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$, is the largest set such that $P \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$ implies:

$$\textit{If } (\nu \tilde{r})\langle P, H \rangle \longrightarrow (\nu \tilde{r'})\langle P', H' \rangle \textit{ then } H =_{\mathscr{L}} H' \textit{ and } P' \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}.$$

Note that a semantically $\mathscr{L}$-high process is not necessarily syntactically $\mathscr{L}$-high. For instance a conditional whose condition is always true and whose first branch is of level $\top$ will be semantically $\mathscr{L}$-high for all downward-closed $\mathscr{L} \subset \mathscr{S}$ (proper inclusion) even if its second branch could emit some message of level $\bot$. On the other hand syntactic $\mathscr{L}$-highness implies semantic $\mathscr{L}$-highness, as stated next.

Lemma 18 (Syntactic $\mathscr{L}$-highness implies semantic $\mathscr{L}$-highness)
Let $\Gamma \vdash_\ell P \triangleright \Delta$ for some $\ell \notin \mathscr{L}$. Then $P \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$.

Proof. If $\Gamma \vdash_\ell P \triangleright \Delta$ for some $\ell \notin \mathscr{L}$ then, by the Confinement lemma (Lemma 17), P may only add or remove $\mathbf{Q}$-set messages of level ℓ' such that $\ell \leq \ell'$. This means that the $\mathbf{Q}$-set remains $\mathscr{L}$-equal to itself during the whole execution.

Lemma 19 (Security of semantically $\mathscr{L}$-high processes)
Let $P \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$. Then $P \simeq_{\mathscr{L}} P$.

Proof. We show that the relation $\mathscr{R} = \{(P_1, P_2) \mid P_i \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}\}$ is a $\mathscr{L}$-bisimulation. Suppose that H_1, H_2 are such that $H_1 =_{\mathscr{L}} H_2$ and $\langle P_i, H_i \rangle$ are typable for $i = 1, 2$. Let now $(\nu \tilde{r})\langle P_1, H_1 \rangle \longrightarrow (\nu \tilde{r'})\langle P_1', H_1' \rangle$. By definition of $\mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$, $H_1 =_{\mathscr{L}} H_1'$ and $P_1' \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$. Then we may choose the empty move as the matching move for $(\nu \tilde{r})\langle P_2, H_2 \rangle$, 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 downward-closed set of security levels $\mathscr{L} \subseteq \mathscr{S}$, the relation $\mathscr{R}^{\mathscr{L}}_{\Gamma}$ on processes is defined inductively as follows:

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

i) either $P_i \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$ *for* $i = 1,2$

ii) or P_i *is* $\mathscr{L}$*-bounded in* Γ *for* $i = 1,2$ *and one of the following holds:*

1. $P_1 = P_2$;
2. $P_i = \bar{a}^{\ell}[n] \mid \prod_{\mathtt{p}=1}^{n} a^{\ell}[\mathtt{p}](\alpha_{\mathtt{p}}).Q^{(i)}_{\mathtt{p}}$, *and* $\forall \mathtt{p}, \forall s$ *fresh,* $Q^{(1)}_{\mathtt{p}}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} \mathscr{R}^{\mathscr{L}}_{\Gamma} Q^{(2)}_{\mathtt{p}}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}$;
3. $P_i = s[\mathtt{p}]!^{\ell'}\langle \Pi, e_i \rangle.P'_i$, *where* $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$ *and* $\exists v, \exists \ell \geq \ell'$ *such that* $e_i \downarrow v^{\ell}$;
4. $P_i = s[\mathtt{q}]?^{\ell}(\mathtt{p}, x^{\ell'}).P'_i$, *where* $P'_1\{v^{\ell'}/x^{\ell'}\} \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2\{v^{\ell''}/x^{\ell'}\}$;
5. $P_i = s[\mathtt{p}]!^{\ell}\langle\langle \mathtt{q}, s'[p'] \rangle\rangle.P'_i$, *where* $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$;
6. $P_i = s[\mathtt{q}]?^{\ell}((\mathtt{p}, \alpha_i)).P'_i$, *where* $\forall \mathtt{p}', \forall s'$ *fresh,* $P'_1\{s'[\mathtt{p}']/\alpha_1\} \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2\{s'[\mathtt{p}']/\alpha_2\}$;
7. $P_i = \mathsf{if}\ e\ \mathsf{then}\ P'_i\ \mathsf{else}\ P''_i$ *where* $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$ *and* $P''_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P''_2$;
8. $P_i = s[\mathtt{p}] \oplus^{\ell} \langle \Pi, \lambda \rangle.Q_i$, *where* $Q_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} Q_2$;
9. $P_i = s[\mathtt{p}]\&^{\ell}(\mathtt{q}, \{\lambda_j : Q^i_j\}_{j \in J})$, *where* $Q^1_j \mathscr{R}^{\mathscr{L}}_{\Gamma} Q^2_j$ *for every* $j \in J$;
10. $P_i = (\nu r)P'_i$, *where* $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$;
11. $P_i = \mathsf{def}\ D\ \mathsf{in}\ P'_i$, *where* $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$;
12. $P_i = \prod_{j=1}^{n} Q^{(i)}_j$, *where* $\forall j\ Q^{(1)}_j \mathscr{R}^{\mathscr{L}}_{\Gamma} Q^{(2)}_j$ *follows from one of the previous cases (including the case of semantically high processes).*

Theorem 20 (Noninterference) *If P is typable, then $P \simeq_{\mathscr{L}} P$ for all down-closed $\mathscr{L}$.*

Proof. The proof consists in showing that the relation $\mathscr{R}^{\mathscr{L}}_{\Gamma}$ presented in Definition 8.5 is a $\mathscr{L}$-bisimulation containing the pair (P,P). Note that we have $P \mathscr{R}^{\mathscr{L}}_{\Gamma} P$ by definition (Clause 1). We proceed by induction on the definition of $\mathscr{R}^{\mathscr{L}}_{\Gamma}$. Suppose that $\Gamma \vdash_{\ell_i} P_i \triangleright \Delta_i$ 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 reduction sequence $< P_2 , H_2 > \longrightarrow^* (\nu \tilde{r}) < P'_2 , H'_2 >$ such that $H'_1 =_{\mathscr{L}} H'_2$ and $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P'_2$. We distinguish two cases:

i) Both P_i are not $\mathscr{L}$-bounded in Γ;

ii) Both P_i are $\mathscr{L}$-bounded in Γ;

Suppose that the P_i's are not $\mathscr{L}$-bounded. Then by Lemma 18 we have $P_i \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$, for $i = 1,2$, and thus the move $< P_1 , H_1 > \longrightarrow (\nu \tilde{r}) < P'_1 , H'_1 >$ can be matched by the empty move of $< P_2 , H_2 >$, where $H'_1 =_{\mathscr{L}} H_1 =_{\mathscr{L}} H_2$, and $P'_1 \in \mathscr{H}^{\mathscr{L}}_{\mathtt{sem}}$ implies $P'_1 \mathscr{R}^{\mathscr{L}}_{\Gamma} P_2$ by Clause i) of Definition 8.5. Suppose now that the P_i's are $\mathscr{L}$-bounded. We consider some interesting cases.

- If $P_i = \bar{a}^{\ell}[n] \mid \prod_{\mathtt{p}=1}^{n} a^{\ell}[\mathtt{p}](\alpha_{\mathtt{p}}).Q^{(i)}_{\mathtt{p}}$, then we can only reduce by applying rule [Link], so we get $< P_1 , H_1 > \longrightarrow (\nu s) < \prod_{\mathtt{p}=1}^{n} Q^{(1)}_{\mathtt{p}}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} , H_1 \cup \{s : \varepsilon\} >$ and $< P_2 , H_2 > \longrightarrow (\nu s) < \prod_{\mathtt{p}=1}^{n} Q^{(2)}_{\mathtt{p}}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\} , H_2 \cup \{s : \varepsilon\} >$, where s is

fresh. Since by hypothesis $Q_{\mathtt{p}}^{(1)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}\mathscr{R}_\Gamma^{\mathscr{L}}\, Q_{\mathtt{p}}^{(2)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}$ for each $\mathtt{p}$, we have $\prod_{\mathtt{p}=1}^{n} Q_{\mathtt{p}}^{(1)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}\mathscr{R}_\Gamma^{\mathscr{L}} \prod_{\mathtt{p}=1}^{n} Q_{\mathtt{p}}^{(2)}\{s[\mathtt{p}]/\alpha_{\mathtt{p}}\}$ by Clause 12 of Definition 8.5 and moreover $H_1 =_{\mathscr{L}} H_2$ implies $H_1 \cup \{s:\varepsilon\} =_{\mathscr{L}} H_2 \cup \{s:\varepsilon\}$.

- Let $P_i = s[\mathtt{p}]!^{\ell'}\langle \Pi, e_i\rangle.P_i'$, where $P_1'\mathscr{R}_\Gamma^{\mathscr{L}} P_2'$ and $\exists v, \exists \ell \geq \ell'$ such that $e_i \downarrow v^{\ell}$. 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 $P_1'\mathscr{R}_\Gamma^{\mathscr{L}} P_2'$.

- If $P_i = \prod_{j=1}^{n} Q_j^{(i)}$, where $\forall j\ Q_j^{(1)}\mathscr{R}_\Gamma^{\mathscr{L}} Q_j^{(2)}$ follows from one of the previous cases (including the case of semantically high processes), then we have a reduction:
  $< P_1\ ,\ H_1 > \longrightarrow < Q_1'^{(1)} \mid \prod_{j=2}^{n} Q_j^{(1)}\ ,\ H_1' >$, which is deduced from
  $< Q_1^{(1)}\ ,\ H_1 > \longrightarrow < Q_1'^{(1)}\ ,\ H_1' >$. Since $Q_1^{(1)}\mathscr{R}_\Gamma^{\mathscr{L}} Q_1^{(2)}$ we get by induction
  $< Q_1^{(2)}\ ,\ H_2 > \longrightarrow < Q_1'^{(2)}\ ,\ H_2' >$ with $Q_1'^{(1)}\mathscr{R}_\Gamma^{\mathscr{L}} Q_1'^{(2)}$ and $H_1' =_{\mathscr{L}} H_2'$, and thus we may conclude using 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.

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 [24] 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 does not prevent deadlocks due to inverse session calls. We plan to study a strengthening of interaction typing that would rule out also this kind of deadlock.

The form of declassification considered in this work is admittedly quite simple. However, it already illustrates the connection between declassification and access control, since a declassified value may only be received by a participant whose level is greater than or equal to the original level of the value. This means that declassification is constrained by the access control policy, as in [7]. We plan to extend declassification also to data which are not received from another participant, by allowing declassification of a tested expression, as in this variant of the $\mathtt{B}$ process of our example in Section 2:

$$\mathtt{B}' \;=\; \ldots \;\mathsf{if}\; \{cc^{\top} = \mathit{secret}^{\top}\}^{\top\downarrow\perp} \;\mathsf{then}\; \beta_1 \oplus^{\perp} \langle 2, \mathsf{ok}\rangle.\mathbf{0} \;\mathsf{else}\; \beta_1 \oplus^{\perp} \langle 2, \mathsf{ko}\rangle.\mathbf{0}$$

Again, this declassification would be controlled by requiring $\mathtt{B}'$ to have level $\top$.

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 to require saturation of **Q**-sets, since an empty **Q**-set would no longer be $\mathscr{L}$-equal to a **Q**-set which contains an empty queue where messages of levels belonging to $\mathscr{L}$ can be put. However, notice that this would require adding security levels on queues, since the empty queue that is created when initiating a session corresponding to a high service should not be observable. This is why we chose a simpler (and weaker) form of observation here.

Note finally that, since our syntax is annotated with security levels, it would be possible to equip our calculus with a *monitored semantics* as in [5]. The induced notion of safety would be less restrictive than typability since it would allow, for instance, a high conditional with a low branch provided this branch is never taken.

Acknowledgments We would like to thank Nobuko Yoshida for her encouragement to engage in this work, and the anonymous referees of [8] for helpful feedback.

References

[1] A. Almeida Matos and G. Boudol. On Declassification and the Non-Disclosure Policy. In *Journal of Computer Security*, volume 17, pages 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. Nicola, and M. Loreti. Sessions and Pipelines for Structured Service Programming. In *Proc. FMOODS '08*, volume 5051 of *LNCS*, pages 19–38. Springer, 2008.

[5] G. Boudol. Secure Information Flow as a Safety Property. In *Proc. FAST'08*, volume 5491 of *LNCS*, pages 20–34. Springer, 2009.

[6] G. Boudol and I. Castellani. Noninterference for concurrent programs and thread systems. *Theoretical Comput. Sci.*, 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, M. Dezani-Ciancaglini, and T. Rezk. Session Types for Access and Information Flow Control. In *Proc. CONCUR'10*, LNCS. Springer, 2010.

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

[10] 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.

[11] 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.

[12] 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.

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

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

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

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

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

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

[19] 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.

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

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

[22] A. Sabelfeld and D. Sands. Dimensions and Principles of Declassification. In *Proc. CSFW'05*. IEEE Computer Society, 2005.

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

[24] 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.

[25] 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.

A 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

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

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:

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

Typing of processes

The following process P is a fragment of the component $\mathtt{S}$:

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

Then we can derive:

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

and hence

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

where

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

$T' = \oplus^{\perp}\langle 1,\{\mathsf{ok}:!\langle 1,\mathsf{String}^{\perp\downarrow\perp}\rangle;\mathsf{end},\mathsf{ko}:\mathsf{end}\}\rangle$

Typing of configurations

We can type the configuration

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

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:

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

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

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

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

Centre de recherche INRIA Sophia Antipolis – Méditerranée
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399