The Journal of Systems and Software 120 (2016) 133–153

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# Privacy protection by typing in ubiquitous computing systems

François Siewe[a,*], Hongji Yang[b]

[a] School of Computer Science and Informatics, De Montfort University, Leicester, LE1 9BH, United Kingdom
[b] Centre for Creative Computing, Bath Spa University, Bath, BA2 9BN, United Kingdom

ARTICLE INFO

*Article history:*
Received 2 November 2015
Revised 4 May 2016
Accepted 24 July 2016
Available online 26 July 2016

*Keywords:*
Privacy
Type system
Ubiquitous computing
Pervasive systems
Security
Type-checking
Simulation

ABSTRACT

Ubiquitous computing systems collect and share a great deal of information upon the users and their environment; including private or highly sensitive personal information. Unless users are confident enough that their privacy is protected, many will be deterred from using such systems. This paper proposes a privacy type system that controls the behaviour of concurrent, context-aware and mobile processes to ensure that private information is not accidentally disclosed. We prove the subject reduction property and the soundness of the proposed type system; which guarantee that a well-typed process cannot accidentally disclose private information. We demonstrate the pragmatics of our approach with a case study.

© 2016 Elsevier Inc. All rights reserved.

## 1. Introduction

Thanks to the advances in technologies, the vision of ubiquitous computing (ubicomp, for short) (Weiser, 1991) is increasingly becoming a reality with the proliferation of smart handheld devices such as smart phones and tablet computers capable of providing the user with relevant information and services anytime and anywhere. These smart devices are equipped with a variety of sensors to collect information upon the user context and share this information via a network to enable timely adaptation to changes in the user context. These may include private or highly sensitive personal information such as the user's location, activity, or personal health information that must be protected from falling into the wrong hands. For example, a smart phone equipped with a GPS (Global Positioning System) receiver can sense the user location and activity (e.g. walking or driving) and eventually share them with other devices or services via the Internet. According to eMarketer's prediction (eMa, 2015), by 2017 the number of mobile phone users will surpass 5 billion worldwide; making it more urgent than ever to develop mechanisms for protecting user privacy in ubicomp systems.

However, privacy is a subjective concept based on personal perceptions of risk and benefit (Schilit et al., 2003). In general, people are likely to disclose personal information in exchange of services if they believe the benefit outweighs the potential cost of this information being misused (Schilit et al., 2003; Hong et al., 2004). This is not to say that because ubicomp carries many benefits to the user –in term of calm, invisible, context-aware and adaptive computing– privacy is not an issue in ubicomp. Surely, the threat of violating individual's privacy is more severe than ubicomp systems not being used or accepted. Ubicomp is so intrusive to our everyday life; collects all kind of information about us in a completely unobtrusive manner, anywhere and anytime; and shares this information with the environment without our knowledge most of the time. Surely, there is a high risk that private information be disclosed accidentally; and more so in large-scale distributed ubicomp systems. Unless a mechanism is put in place that gives users enough confidence that their privacy is protected, many will be deterred from using such systems. To address this problem, type systems are a powerful technique to constrain the behaviours of a system so that certain errors and security violations do not occur at run-time. A modelling language with a well-defined formal semantics will help to understand the behaviours of ubicomp systems and to reason about the system requirements prior to implementation.

The Calculus of Context-aware Ambients (CCA, for short) (Siewe et al., 2011) is a formal language for modelling ubicomp systems. The main features of the calculus include *context-awareness*, *mobility* and *concurrency*. The concept of *ambient*, inherited from (Cardelli and Gordon, 2000), represents an abstraction of a place where computation may happen. An ambient may contain other ambients called child ambients organised into a tree structure. Such a hierarchy can be used to model any entity in a ubicomp

* Corresponding author.
*E-mail addresses:* fsiewe@dmu.ac.uk (F. Siewe), h.yang@bathspa.ac.uk (H. Yang).

http://dx.doi.org/10.1016/j.jss.2016.07.037
0164-1212/© 2016 Elsevier Inc. All rights reserved.

system –whether physical, logical, mobile or immobile– as well as the environment (or context) of that entity (Siewe et al., 2011). For example the user location, the user profile, the mobile devices carried by the user, and the nearby resources can be modelled using the concept ambient (see Al-Doori et al., Al-Sammarraie et al. (2010); Almutairi and Siewe (2013)). In addition, CCA models are fully executable (using the CCA interpreter ccaPL), which is very useful for analysing the behaviour of ubicomp systems. Hence the reasons why CCA is chosen for this work. In CCA ambients communicate by message passing, can be mobile (or immobile), and can be aware of the presence of other ambients. These interactions among ambients, if not properly regulated, may lead to unwanted disclosure of private information, whether directly or indirectly.

This paper proposes a novel type system that constrains the behaviour of CCA processes to ensure that private information is not accidentally disclosed. In this type system, ambients are assigned to groups and have control over who can access their context information, who can share them (through message passing) and with whom. This enables the users to control how their context information is collected, stored and shared. Type checking guarantees that a well-typed process cannot violate the privacy of any ambient. The main contributions of this work are fivefold:

- A syntax of the types is proposed (Section 5.1); its innovative features include notations for describing privacy types. Mobility types and exchange types can also be specified. Privacy types are used to specify the *privacy requirements* of ambients. The privacy type of an ambient specifies the groups of the ambients that can sense that ambient context information, the groups of the ambients that can share that information with a third party, and the groups that third party must belong to. In this way, the flow of context information can be controlled to avoid accidental disclosure of private context information. Type annotations are added to the syntax of CCA in Section 3 and to its semantics in Section 4.
- A formalisation of the proposed type system using typing rules (Section 5.2); only processes that can be typed using these rules are *well-typed*. These rules are used to check statically (i.e. at compile time) the well-typedness of processes.
- The subject reduction property of the proposed type system is formally established (Section 6). This property states that a well-typed process can only reduce to well-typed processes; i.e. well-typedness is preserved by the reduction relation.
- The type soundness (aka safety) property is also formally established (Section 7); therefore it is guaranteed that well-typed processes do not violate the privacy requirements of any ambient in the system, nor give rise to run-time errors during reduction.
- The pragmatics of the proposed type system is illustrated using a case study of an infostation-based mobile communication (IMC) system where the identity and the location information of the sender must not be disclosed (Section 8). The privacy requirements of this system are captured using the type system (Section 8.1). The full specification of the IMC system in CCA is formally proved to be well-typed (Section 8.2). Finally the *reliability* of the IMC system is demonstrated through simulations in ccaPL (the CCA interpreter) which show that the users can communicate anonymously without the risk of revealing their location information (Section 8.3).

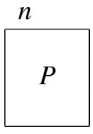
## 2. Motivation and examples

This section illustrates how the concept of *privacy type* proposed in this paper can be used to capture and analyse privacy requirements in ubicomp. Smart homes are a typical example of ubicomp systems where household furniture and appliances interact with each other via a home network to provide the inhabitants personalised services and comfort. The main services a smart home can provide include the intelligent climate control service that controls the heating, ventilation and air-conditioning (HVAC) system to assure good thermal comfort and appropriate indoor air quality; the lighting control service to help save energy; the multi-room audi-visual entertainment service; and the security service which controls the CCTV cameras installed in the smart home. At the tip of a smart phone and miles away from home, one can switch on or off light in the living room; access live CCTV cameras video footage or simply be notified automatically of any intrusion into her property; or attend remotely to an elderly person under her care. Protecting the privacy of the users in such complex and heterogeneous systems is a difficult task.

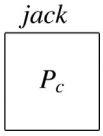
Consider the following scenario. Jack lives with his brother Peter and their grandad Jacob in a smart home. Jacob is an elderly person and suffers from dementia; he often forgets taking his medication and so needs assistance most of the time. The smart home has a number of rooms, each equipped with a location system that enables it to determine the identity of anyone present in the room in order to provide personalised services. The rooms are connected to a home network and to the Internet; and can share with other devices in the network the identity of anyone present in the room. This makes it easy to Jack and Peter to monitor at any time the location of Jacob. However, Jack and Peter would like their own location information to be private; especially not to be disclosed over the Internet. To enforce this privacy requirement in the smart home system, the flow of context information must be controlled across the home network to ensure that private information are not accidentally disclosed. The following shows how this can be done using the proposed privacy type system in the Calculus of Context-aware Ambients (CCA).

*System Modelling in* CCA

In CCA, the concept of ambient is used to conceptualise any entity in a ubicomp system. An ambient has a name and its behaviour is modelled as a process. For example, an ambient of name $n$ and behaviour $P$ is denoted by the process $n[P]$ (textual representation). This ambient can also be represented graphically as follows:




It is assumed that the user carries a mobile device that can be used to identify her. Depending on the location determination system in the rooms, this may be as simple as an RFID tag, or the blue-tooth of the user's smart phone. Any user of a smart home can be represented as an ambient, e.g.




An ambient may contain other ambients which are its *child ambients*. A parent ambient represents the *location* of its child ambients. Ambients that have the same parent are called *sibling* ambients. Such a tree structure can be used to model any ubicomp system. Suppose Jack's smart home has one living room with a TV set, one kitchen, one bath room and three bedrooms. Fig. 1 depicts the smart home model in CCA when Jack is in the living room, Peter in the Kitchen and Jacob in his bedroom. The corresponding textual representation is the following process, where the symbol '|' denotes the parallel composition of processes and each $P_i$ describes

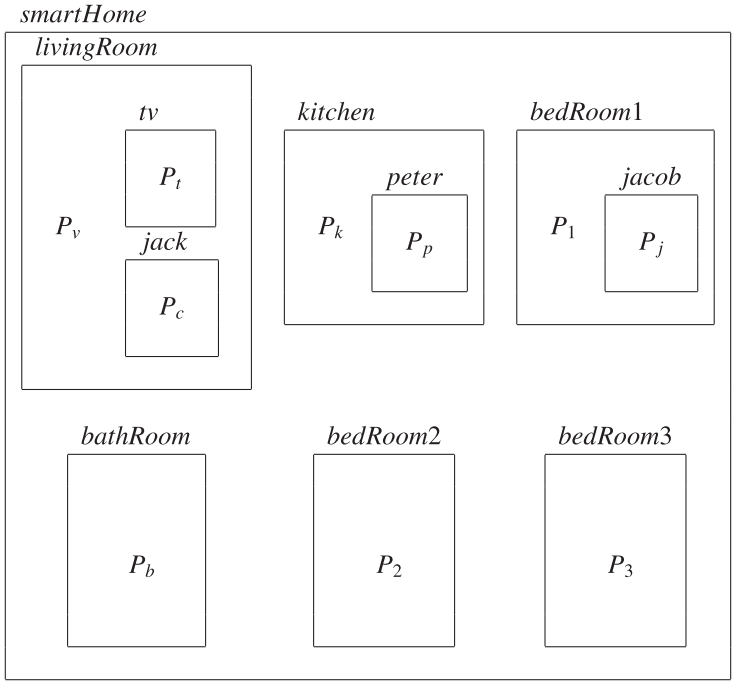



**Fig. 1.** A smart home model in CCA.

the behaviour of the ambient it belongs to.

$$
\begin{aligned}
&\mathit{smartHome}[\\
&\quad \mathit{livingRomm}[P_v \mid \mathit{tv}[P_t] \mid \mathit{jack}[P_c]]\\
&\quad \mid \mathit{kitchen}[P_k \mid \mathit{peter}[P_p]]\\
&\quad \mid \mathit{bedRoom1}[P_1 \mid \mathit{jacob}[P_j]]\\
&\quad \mid \mathit{bathRoom}[P_b]\\
&\quad \mid \mathit{beRoom2}[P_2]\\
&\quad \mid \mathit{beRoom3}[P_3]\\
&]
\end{aligned}
$$

*Context Modelling in* CCA

The context of an ambient in a system is characterised by the surrounding environment of that ambient in that system. This is obtained by replacing that ambient in the system by a place holder called the *hole* and denoted by $\odot$. For example, the context of Jack in the smart home system is depicted in Fig. 2. In CCA a context-expression is a predicate over context, i.e. a formula that describes a situation in the system's context; the formula holds if that situation occurs in the system's context. For example, the context-expression '$\diamond(\mathit{livingRoom}[\mathit{jack}[\mathtt{true}] \mid \mathtt{true}] \mid \mathtt{true})$' holds if the ambient Jack is in the living room, where the spatial operator $\diamond$ is called the *somewhere modality*. Indeed, a context-expression $\diamond\kappa$ holds for a context if the context-expression $\kappa$ holds somewhere in that context. More details about context expression are given in Section 3. Context-expressions are used in CCA to guard capabilities. A *context-guarded prefix* has the form $\kappa?M.P$, where $\kappa$ is a context-expression, $M$ a capability (i.e. an atomic action that a process can perform) and $P$ a continuation process. It waits until the environment satisfies the context-expression $\kappa$, executes $M$ and then continues like $P$. Such a process is said to be *context-aware*. For example the process '$\diamond(\mathit{livingRoom}[\mathit{jack}[\mathtt{true}] \mid \mathtt{true}] \mid \mathtt{true})?\mathtt{send}().\mathbf{0} \mid \mathtt{recv}().\mathbf{0}$' waits until Jack enters the living room, sends a signal and then terminates.

*Communication in* CCA

Ambients can exchange messages using the capability `send` for output and `recv` for input. The data structure for messages is a record type of the form $\{l_1 : W_1, \ldots, l_\ell : W_\ell\}$, where $l_i$ is a field of type $W_i$, $1 \leq i \leq \ell$. For example, Jack in the living room can interact with the sibling ambient TV as depicted in Fig. 3, where $\rightarrow$ represents a reduction step. We write '$::\mathtt{recv}$' to receive from any sibling and '$\mathit{tv} :: \mathtt{send}$' to send to a specific ambient named *tv*. The process $P_t\{\mathit{bbc1}/y\}$ denotes the process obtained by replacing each free occurrence of the name $y$ in $P_t$ by *bbc*1.

*Mobility in* CCA

Jack can move from the living room to the kitchen by performing the mobility capability '`out`' to move out of its current parent (which is the living room) and then the mobility capability '`in` *kitchen*' to move into the (sibling ambient) kitchen, as shown in Fig. 4.

*Privacy Types*

The mobility capabilities enable an ambient to move from one location to another, while communication capabilities are used to exchange messages between two ambients by the message passing mechanism. Therefore in the proposed type system, the type of an ambient $n$ as depicted in Eq. (1) is composed of the *group* $g$ that ambient belongs to; that ambient's *mobility type* $X$ (i.e. whether it is mobile or immobile); its *exchange type* $T$ (i.e. the type of information it can send or receive); and its *privacy type* $U$. It is assumed an infinite set of group names, denoted by '$*$'. The mobility type and the exchange type are presented in details in Section 5. For this scenario, only the privacy type will be discussed.

$$n : \mathtt{amb}(g)[X, T, U] \tag{1}$$

An ambient's privacy type has the form $\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$, where $\mathcal{R}$ is the set of groups of ambients that are allowed to read the context of that ambient; $\mathcal{O}$ is the set of groups of ambients that are allowed to share that ambient's context information; and $\mathcal{I}$ is the set of groups of ambients that ambient's context information can be shared with. Thus, each ambient has control over who can read their context information, who can share their context information with a third party, and who that third party might be. Two extreme cases can be identified: i) the privacy type $\mathtt{priv}[*, *, *]$

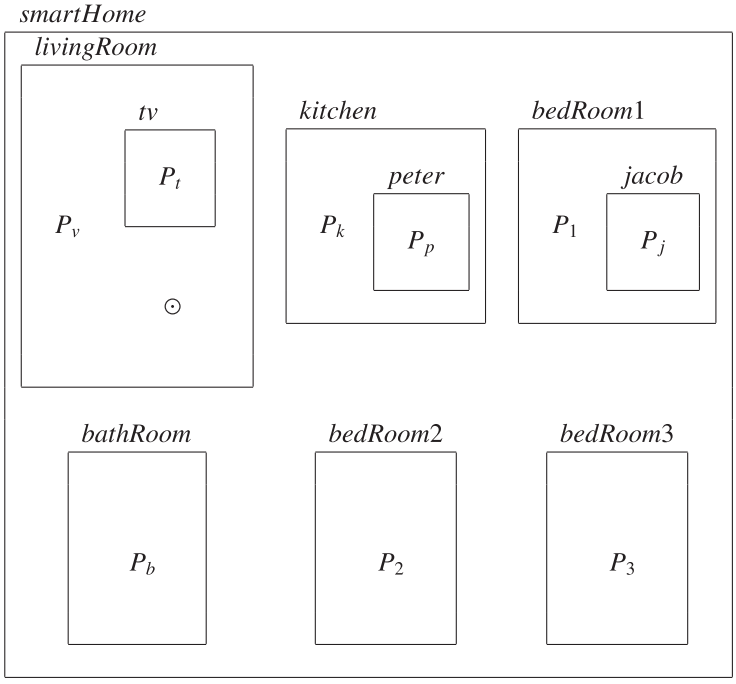



**Fig. 2.** The context of the ambient jack in the smart home system.

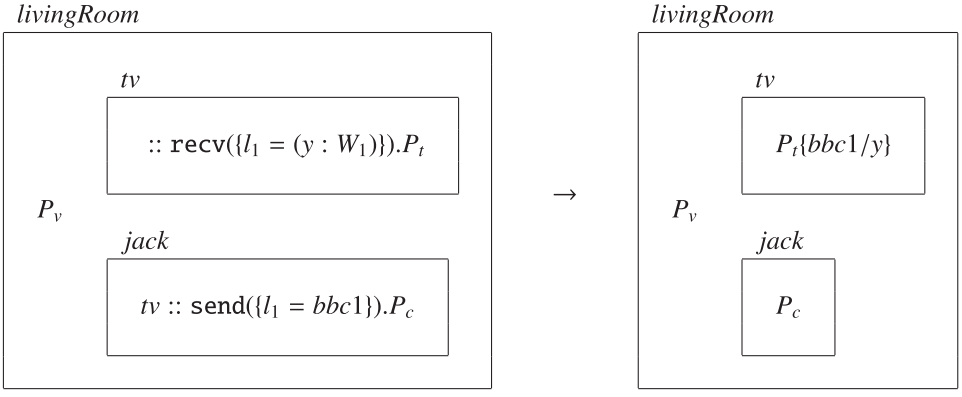



**Fig. 3.** Ambient communications.

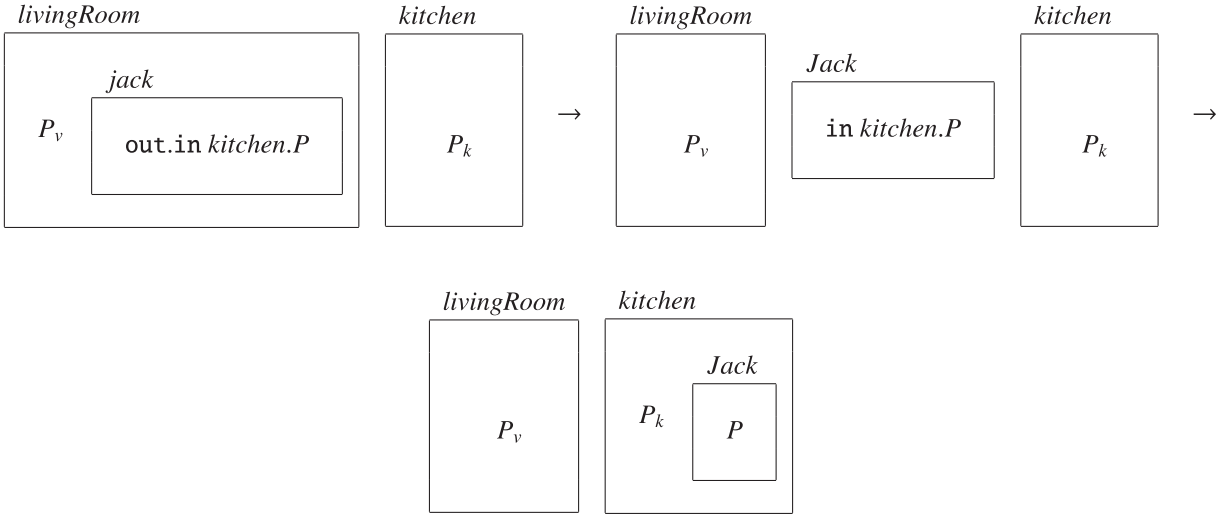



**Fig. 4.** Mobility in CCA.

**Table 1**
Syntax of CCA.

| | | | | | | |
|---|---|---|---|---|---|---|
| $P, Q$ | ::= | **Process** | $\kappa$ | ::= | | **Context Expressions** |
| | $\mathbf{0}$ | inactivity | | | `true` | true |
| | $P\|Q$ | parallel composition | | | $n = m$ | name match |
| | $(\nu\, n{:}\ W)P$ | name restriction | | | $\bullet$ | hole |
| | $(\nu\, g : \mathtt{gr})\ P$ | group restriction | | | $\neg\kappa$ | negation |
| | $!P$ | replication | | | $\kappa_1 \mid \kappa_2$ | parallel composition |
| | $n[P]$ | ambient | | | $\kappa_1 \wedge \kappa_2$ | conjunction |
| | $\kappa?M.P$ | context-guarded prefix | | | $n[\kappa]$ | location context |
| | $x \triangleright (y_1, \ldots, y_\ell).P$ | process abstraction | | | $\oplus\kappa$ | spatial next modality |
| | `if` $\kappa_1?M_1.P_1$ ... $\kappa_\ell?M_\ell.P_\ell$ `fi` | selection | | | $\diamond\kappa$ | somewhere modality |
| $M$ | ::= | **Capabilities** | $\alpha$ | ::= | | **Locations** |
| | `del` $n$ | delete ambient $n$ | | | $\uparrow$ | any parent |
| | `in` $n$ | move into ambient $n$ | | | $n\uparrow$ | parent $n$ |
| | `out` | move out of parent | | | $\downarrow$ | any child |
| | $\alpha\ x(z_1, \ldots, z_\ell)$ | process abstraction call | | | $n\downarrow$ | child $n$ |
| | $\alpha$ `recv`$(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\})$ | input | | | $: :$ | any sibling |
| | $\alpha$ `send`$(\{l_1 = z_1, \ldots, l_\ell = z_\ell\})$ | output | | | $n: :$ | sibling $n$ |
| | | | | | $\epsilon$ | locally |

of *public* ambients, i.e. everyone can read and share their context information; ii) the privacy type `priv`$[\emptyset, \emptyset, \emptyset]$ of *discreet* ambients, i.e. no one can read or share their context information. Ambients that are not discreet nor public take their privacy types in between these two extremes. For example let *f, g* and *h* be the groups of the ambients representing Jack's, Peter's and Jacob's mobile devices respectively. The requirement that Jack and Peter should be able to access Jacob's location information at anytime and share it with anyone they trust (e.g. an emergency service or a physician) can be specified by the privacy type `priv`$[\{f, g\}, \{f, g\}, *]$. Suppose Jack allows Peter and Jacob to access his location information but not to share it with anyone else; this requirement can be specified as `priv`$[\{g, h\}, \emptyset, \emptyset]$. Privacy types are used to specify the *privacy requirements* of ambients. Type checking guarantees that a well-typed system does not violate the privacy requirements of any ambient. The syntax of typed CCA is presented in the following section.

## 3. Syntax of typed CCA

This section presents the syntax and the informal semantics of typed CCA. Table 1 depicts the syntax of typed CCA, based on three syntactic categories: processes (denoted by $P$ or $Q$), capabilities (denoted by $M$) and context-expressions (denoted by $\kappa$). We assume a countably-infinite set of names, elements of which are written in lower-case letters, e.g. *n, g, x* and *y*. Each name has a type, either a message type $W$ (defined later in Section 5.1) or the group type `gr`.

*Processes*

The process $\mathbf{0}$, aka *inactivity process*, does nothing and terminates immediately. The process $P|Q$ denotes the concurrent execution of the processes $P$ and $Q$. The process $(\nu\, n{:}\ W)P$ creates a new name $n$ of type $W$ and the scope of that name is limited to the process $P$. The replication $!P$ denotes a process which can always create a new copy of $P$, i.e. $!P$ is equivalent to $P|!P$. Replication, first introduced in the $\pi$-Calculus (Milner, 1999), can be used to implement both iteration and recursion. The process $n[P]$ denotes an ambient named $n$ whose behaviours are described by the process $P$. The pair of square brackets '[' and ']'outlines the boundary of that ambient. An ambient is represented graphically as in Section 2.

A context expression specifies a property upon the state of the environment. A *context-guarded prefix* $\kappa?M.P$ is a process that waits until the environment satisfies the context expression $\kappa$, then performs the capability $M$ and continues like the process $P$. The dot symbol '.' denotes the sequential composition of processes. We let $M.P$ denote the process `true`$?M.P$, where `true` is a context expression satisfied by all context. A process abstraction $x \triangleright (y_1, \ldots, y_\ell).P$ denotes the linking of the name $x$ to the process $P$ where $y_1, \ldots, y_\ell$ are the *formal parameters*. This linking is local to the ambient where the process abstraction is defined. So a name $x$ can be linked to a specific process in one ambient and to a different process in another ambient. This locality property is interesting for context-awareness, especially when combined with ambient mobility. Hence an ambient calling a process abstraction may behave differently depending on its current location. A *selection* process `if` $\kappa_1?M_1.P_1$ ... $\kappa_\ell?M_\ell.P_\ell$ `fi` waits until at least one of the context-expressions $(\kappa_i)_{1 \le i \le \ell}$ holds; then proceeds non-deterministically like one of the processes $\kappa_j?M_j.P_j$ for which $\kappa_j$ holds.

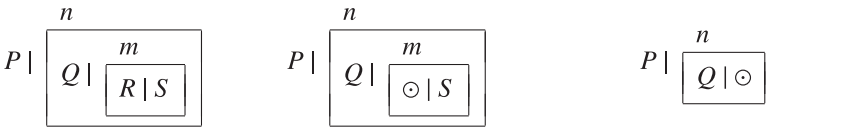
*Capabilities*

A call to a process abstraction named $x$ is done by a capability of the form $\alpha\ x(z_1, \ldots, z_\ell)$ where $\alpha$ specifies the location where the process abstraction is defined and $z_1, \ldots, z_\ell$ are *actual parameters*. The location $\alpha$ can be '$\uparrow$ ' for any parent, '$n\uparrow$ ' for a specific parent $n$, '$\downarrow$' for any child, '$n\downarrow$' for a specific child $n$, ': : ' for any sibling, '$n$: : ' for a specific sibling $n$, or $\epsilon$ (empty string) for the calling ambient itself. A process call $\alpha\ x(z_1, \ldots, z_\ell)$ behaves like the process linked to $x$ at location $\alpha$, in which each actual parameter $z_i$, $i = 1, \ldots, \ell$ is substituted for each occurrence of the corresponding formal parameter. A process call can only take place if the corresponding process abstraction is *defined* at the specified location.

Ambients exchange messages using the output capability $\alpha$ `send`$(\{l_1 = z_1, \ldots, l_\ell = z_\ell\})$ to send a record containing the names $z_1, \ldots, z_\ell$ to a location $\alpha$, and the input capability $\alpha$ `recv`$(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\})$ to receive a record containing names of specified types from a location $\alpha$, where $l_1, \ldots, l_\ell$ are the fields of this record type (see Table 6). The mobility capabilities `in` and `out` are defined as follows. An ambient that performs the capability `in` $n$ moves into the sibling ambient $n$. The capability `out` moves the ambient that performs it out of that ambient's parent. The capability `del` $n$ deletes an ambient of the form $n[\mathbf{0}]$ situated at the same level as that capability, i.e. the process `del` $n.P \mid n[\mathbf{0}]$ reduces to $P$. The capability `del` acts as a garbage collector that deletes ambients which have completed their computations.

**Example 3.1.**

- The process $n[\mathtt{in}\ m.\mathtt{out}.\mathbf{0}] \mid m[\mathtt{in}\ n.\mathtt{out}.\mathbf{0}]$ describes the behaviours of two sibling ambients $n$ and $m$ concurrently willing to move in and out of one another.
- The ambient $n[(\diamond\mathtt{at}(m))? :: \mathtt{send}(\{\ell = msg\}).\mathbf{0}]$ releases the message '*msg*' only when at location $m$; where the context expression $\diamond\mathtt{at}(m)$ holds if $n$ is a child ambient of the ambi-




(a) System model (b) Context of process $R$ (c) Context of ambient $m$

**Fig. 5.** Graphical illustration of the context of a process.

**Table 2**
Syntax of contexts.

| | | |
|---|---|---|
| $\mathcal{C}$ | ::= | $\mathbf{0} \mid \odot \mid n[\mathcal{C}] \mid \mathcal{C}\mid P \mid (\nu\ n : W)\ \mathcal{C} \mid (\nu\ g : \mathtt{gr})\ \mathcal{C}$ |

**Table 3**
Satisfaction relation for context expressions.

| | | | |
|---|---|---|---|
| $\mathcal{C}$ | $\models$ | `true` | |
| $\mathcal{C}$ | $\models$ | $n = n$ | |
| $\mathcal{C}$ | $\models$ | $\bullet$ | iff $\mathcal{C} = \odot$ |
| $\mathcal{C}$ | $\models$ | $\neg\kappa$ | iff $\mathcal{C} \not\models \kappa$ |
| $\mathcal{C}$ | $\models$ | $\kappa_1 \mid \kappa_2$ | iff exist $\mathcal{C}_1, \mathcal{C}_2$ such that $\mathcal{C} = \mathcal{C}_1 \mid \mathcal{C}_2$ and $\mathcal{C}_1 \models \kappa_1$ and $\mathcal{C}_2 \models \kappa_2$ |
| $\mathcal{C}$ | $\models$ | $\kappa_1 \wedge \kappa_2$ | iff $\mathcal{C} \models \kappa_1$ and $\mathcal{C} \models \kappa_2$ |
| $\mathcal{C}$ | $\models$ | $n[\kappa]$ | iff exists $\mathcal{C}'$ such that $\mathcal{C} = n[\mathcal{C}']$ and $\mathcal{C}' \models \kappa$ |
| $\mathcal{C}$ | $\models$ | $\oplus\kappa$ | iff exist $\mathcal{C}', n$ such that $\mathcal{C} = n[\mathcal{C}']$ and $\mathcal{C}' \models \kappa$ |
| $\mathcal{C}$ | $\models$ | $\Diamond\kappa$ | iff $\mathcal{C} \models \kappa$ or exist $\mathcal{C}', n$ such that $\mathcal{C} = n[\mathcal{C}']$ and $\mathcal{C}' \models \Diamond\kappa$ |

ent $m$. The formal definition of the predicate '`at`' is given in Example 3.2.

*Context model*

In `CCA`, a context is modelled as a process with a hole in it. The hole (denoted by $\odot$) in a context represents the position of the process that context is the context of. For example, suppose a system is modelled by the process $P \mid n[Q \mid m[R \mid S]]$. So, the context of the process $R$ in that system is $P \mid n[Q \mid m[\odot \mid S]]$, and that of the ambient named $m$ is $P \mid n[Q \mid \odot]$ as depicted graphically in Fig. 5.

Thus the contexts of typed `CCA` processes are described by the grammar in Table 2. A property of a context can be described by a formula called a *context expression* (CE in short).

*Context expressions*

The CE `true` always holds. A CE $n = m$ holds if the names $n$ and $m$ are lexically identical. The CE $\bullet$ holds solely for the hole context, i.e. the position of the process evaluating that context expression. Propositional operators such as negation ($\neg$) and conjunction ($\wedge$) expand their usual semantics to context expressions. A CE $\kappa_1 \mid \kappa_2$ holds for a context if that context is a parallel composition of two contexts such that $\kappa_1$ holds for one and $\kappa_2$ holds for the other. A CE $n[\kappa]$ holds for a context if that context is an ambient named $n$ such that $\kappa$ holds inside that ambient. A CE $\oplus\kappa$ holds for a context if that context has a child context for which $\kappa$ holds. A CE $\Diamond\kappa$ holds for a context if there exists somewhere in that context a sub-context for which $\kappa$ holds. The operator $\Diamond$ is called *somewhere modality*, while $\oplus$ is aka *spatial next modality*. The formal semantics of CEs with respect to the context model of Table 2 is given in Table 3, where the notation '$\mathcal{C} \models \kappa$' means that the context $\mathcal{C}$ satisfies the context expression $\kappa$. We also write '$\models\kappa$' to mean that a context expression $\kappa$ is *valid*, i.e. is satisfied by all contexts.

**Example 3.2.** We now give some examples of predicates that can be used to specify common context properties such as the location of the user, with whom the user is and what resources are nearby. In these sample predicates we take the view that a process is evaluated by the immediate ambient $\lambda$ say that contains it. We also use the notation '$\widehat{=}$' to mean '*by definition*'.

- $\mathtt{has}(n) \ \widehat{=}\ \oplus(\bullet \mid n[\mathtt{true}] \mid \mathtt{true})$ : holds if $\lambda$ is top ambient and contains an ambient named $n$
- $\mathtt{at}(n) \ \widehat{=}\ n[\oplus(\bullet \mid \mathtt{true})] \mid \mathtt{true}$ : holds if $\lambda$ is located at a top ambient named $n$
- $\mathtt{with}(n) \ \widehat{=}\ n[\mathtt{true}] \mid \oplus(\bullet \mid \mathtt{true})$ : holds if $\lambda$ is (co-located) with an ambient named $n$ at a top ambient.

**Example 3.3.** The predicates of Example 3.2 can be generalised to binary predicates as follows:

- $\mathtt{at2}(n, m) \ \widehat{=}\ n[m[\mathtt{true}] \mid \mathtt{true}] \mid \mathtt{true}$ : holds if ambient $m$ is located at a top ambient named $n$
- $\mathtt{with2}(n, m) \ \widehat{=}\ n[\mathtt{true}] \mid m[\mathtt{true}] \mid \mathtt{true}$ : holds if ambient $m$ is (co-located) with a top ambient named $n$
- $\Diamond\mathtt{with2}(n, m)$ : holds if ambient $m$ is (co-located) with an ambient named $n$ somewhere in the ambient hierarchy.

## 4. Reduction semantics of typed CCA

The operational semantics of typed `CCA` is defined using a structural congruence '$\equiv$' and a reduction relation '$\rightarrow$'. The structural congruence is the smallest congruence relation on processes that satisfies the axioms in Table 4. These axioms allow the manipulation of the structure of processes. It follows from these axioms that the structural congruence is a commutative monoid for ($\mathbf{0}$, $\mid$). These axioms are inherited from `CCA` (Siewe et al., 2011), differ just with the type annotations. For instance, the axiom (S22) says that a capability guarded with `true` is the same as that capability without guards. The next two axioms (S23) and (S24) define the equivalence of process abstractions and context-guarded prefix, respectively.

The reduction relation of processes is defined in Table 5, where $P\{z_1/y_1, \ldots, z_\ell/y_\ell\}$ denotes the process obtained by replacing each free occurrence of the name $y_i$ in $P$ by $z_i$, $i = 1, \ldots, \ell$. We sometimes denote by $\sigma$ a substitution $\{z_1/y_1, \ldots, z_\ell/y_\ell\}$. The first set of rules (R1 to R7) gives the semantics of a process call. It states that a process call takes place only if a corresponding process abstraction is defined at the specified location. The next set of rules, (R8) to (R20), are related to message passing communication between processes and across ambient boundaries. In these rules $\mathtt{fn}(X)$ stands for the set of all the names that occur free in a term $X$ and $\mathtt{fg}(X)$ the set of all the groups that occur free in $X$; these two functions are defined in Table 7 and Table 8, respectively. The mobility rules (R21) and (R22) are inherited from `MA` (Cardelli and Gordon, 2000). The rule (R23) states that an ambient that has completed its computation can be explicitly deleted from the system using the capability `del`. The rules (R24) and (R25) propagate reduction across scopes, while the rule (R26) states that the reduction relation is closed under structural equivalence. The rule (R27) asserts that a context-guarded capability reduces in a context if that context satisfies the guard of that capability. In this rule, $\sigma$ denotes a substitution of names. Finally, the semantics of a selection process is formalised by the rule (R28); one of the enabled branches is chosen non-deterministically for execution.

**Table 4**
Structural congruence for processes.

**(S1)** $P \equiv P$ **(S2)** $P \equiv Q \Rightarrow Q \equiv P$ **(S3)** $P \equiv Q, Q \equiv R \Rightarrow P \equiv R$
**(S4)** $P \equiv Q \Rightarrow (\nu n{:}\ W)P \equiv (\nu n{:}\ W)Q$ **(S5)** $P \equiv Q \Rightarrow P|R \equiv Q|R$
**(S6)** $P \equiv Q \Rightarrow\ !P \equiv\ !Q$ **(S7)** $P \equiv Q \Rightarrow n[P] \equiv n[Q]$ **(S8)** $P|Q \equiv Q|P$
**(S9)** $P|(Q|R) \equiv (P|Q)|R$ **(S10)** $(\nu n{:}\ W_1)(\nu m{:}\ W_2)P \equiv (\nu m{:}\ W_2)(\nu n{:}\ W_1)P$
**(S11)** $(\nu n{:}\ W)P|Q \equiv (\nu n{:}\ W)(P|Q)$ if $n \notin \mathbf{fn}(Q)$
**(S12)** $(\nu n{:}\ W)m[P] \equiv m[(\nu n{:}\ W)P]$ if $n \neq m$
**(S13)** $(\nu g_1 : \mathtt{gr})\ (\nu g_2 : \mathtt{gr})\ P \equiv (\nu g_2 : \mathtt{gr})\ (\nu g_1 : \mathtt{gr})\ P$
**(S14)** $(\nu g : \mathtt{gr})\ (\nu n : W)\ P \equiv (\nu n : W)\ (\nu g : \mathtt{gr})\ P$ if $g \notin \mathbf{fg}(W)$
**(S15)** $(\nu g : \mathtt{gr})\ P \mid Q \equiv (\nu g : \mathtt{gr})\ (P \mid Q)$ if $g \notin \mathbf{fg}(Q)$
**(S16)** $(\nu g : \mathtt{gr})\ m[P] \equiv m[(\nu g : \mathtt{gr})\ P]$ **(S17)** $!P \equiv P|!P$ **(S18)** $P|\mathbf{0} \equiv P$ **(S19)** $!\mathbf{0} \equiv \mathbf{0}$
**(S20)** $(\nu n{:}\ W)\mathbf{0} \equiv \mathbf{0}$ **(S21)** $(\nu g : \mathtt{gr})\ \mathbf{0} \equiv \mathbf{0}$ **(S22)** $\mathtt{true}?M.P \equiv M.P$
**(S23)** $P \equiv Q \Rightarrow x \triangleright (y_1, \ldots, y_\ell).P \equiv x \triangleright (y_1, \ldots, y_\ell).Q$
**(S24)** $P \equiv Q, (\vdash \kappa \Leftrightarrow \kappa') \Rightarrow \kappa?M.P \equiv \kappa'?M.Q$

**Table 5**
Reduction relation for processes.

**(R1)** $x \triangleright (y_1, \ldots, y_\ell).P \mid x(z_1, \ldots, z_\ell) \rightarrow x \triangleright (y_1, \ldots, y_\ell).P \mid P\{z_1/y_1, \ldots, z_\ell/y_\ell\}$
**(R2)** $n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid m[:: x(z_1, \ldots, z_\ell) \mid R] \rightarrow n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid m[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]$
**(R3)** $n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid m[n :: x(z_1, \ldots, z_\ell) \mid R] \rightarrow n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid m[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]$
**(R4)** $x \triangleright (y_1, \ldots, y_\ell).P \mid m[\uparrow x(z_1, \ldots, z_\ell) \mid R] \rightarrow x \triangleright (y_1, \ldots, y_\ell).P \mid m[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]$
**(R5)** $n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P \mid m[n \uparrow x(z_1, \ldots, z_\ell) \mid R]] \rightarrow n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P \mid m[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]]$
**(R6)** $n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid \downarrow x(z_1, \ldots, z_\ell) \rightarrow n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid P\{z_1/y_1, \ldots, z_\ell/y_\ell\}$
**(R7)** $n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid n \downarrow x(z_1, \ldots, z_\ell) \rightarrow n[Q \mid x \triangleright (y_1, \ldots, y_\ell).P] \mid P\{z_1/y_1, \ldots, z_\ell/y_\ell\}$
**(R8)** $\mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q \rightarrow P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q$
**(R9)** $n[:: \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P|Q] \mid m[:: \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).R|S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\}|Q] \mid m[R|S]$
**(R10)** $n[:: \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P|Q] \mid m[n :: \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).R|S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q] \mid m[R \mid S]$
**(R11)** $n[m :: \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P|Q] \mid m[:: \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).R|S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q] \mid m[R \mid S]$
**(R12)** $n[m :: \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P|Q] \mid m[n :: \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).R|S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q] \mid m[R \mid S]$
**(R13)** $\downarrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid m[\uparrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q \mid R] \rightarrow P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid m[Q \mid R]$
**(R14)** $m \downarrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid m[\uparrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q \mid R] \rightarrow P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid m[Q \mid R]$
**(R15)** $\downarrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P \mid m[\uparrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).Q \mid R] \rightarrow P \mid m[Q\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]$
**(R16)** $m \downarrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P \mid m[\uparrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).Q \mid R] \rightarrow P \mid m[Q\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R]$
**(R17)** $n[\downarrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid m[n \uparrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q|R||S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid m[Q \mid R] \mid S]$
**(R18)** $n[m \downarrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid m[n \uparrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q|R||S] \rightarrow n[P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid m[Q \mid R] \mid S]$
**(R19)** $n[\downarrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P \mid m[n \uparrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).Q|R||S] \rightarrow n[P \mid m[Q\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R] \mid S]$
**(R20)** $n[m \downarrow \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P \mid m[n \uparrow \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).Q|R||S] \rightarrow n[P \mid m[Q\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid R] \mid S]$
**(R21)** $n[\mathtt{in}\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$ **(R22)** $m[n[\mathtt{out}.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$
**(R23)** $\mathtt{del}\ n.P \mid n[\mathbf{0}] \rightarrow P$ **(R24)** $P \rightarrow P' \Rightarrow (\nu n{:}\ W)P \rightarrow (\nu n{:}\ W)P'$
**(R25)** $P \rightarrow P' \Rightarrow (\nu g : \mathtt{gr})\ P \rightarrow (\nu g : \mathtt{gr})\ P'$ **(R26)** $P \equiv Q, Q \rightarrow Q', Q' \equiv P' \Rightarrow P \rightarrow P'$
**(R27)** $\mathcal{C}(M.P) \rightarrow \mathcal{C}'(P\sigma) \Rightarrow \mathcal{C}(\kappa?M.P) \rightarrow \mathcal{C}'(P\sigma)$ if $\mathcal{C} \models \kappa$
**(R28)** $\mathcal{C}(\kappa_i?M_i.P_i) \rightarrow \mathcal{C}'(P_i\sigma) \Rightarrow \mathcal{C}(\mathtt{if}\ \kappa_1?M_1.P_1\ \ldots\ \kappa_\ell?M_\ell.P_\ell\ \mathtt{fi}) \rightarrow \mathcal{C}'(P_i\sigma)$ for some $i$, $1 \leq i \leq \ell$

**Table 6**
Types.

| | | | |
|---|---|---|---|
| $W$ | ::= | **Message types** | |
| | | $\mathtt{amb}(g)[X, T, U]$ | ambient name |
| | | $\mathtt{abs}(W_1 \times \cdots \times W_\ell)[X, T, U]$ | process name |
| | | $Z$ | type variable |
| | | $\mu Z.W$ | recursive type |
| $T$ | ::= | **Exchange types** | |
| | | $\mathtt{shh}$ | no exchange |
| | | $\{l_1{:}\ W_1, \cdots, l_\ell{:}\ W_\ell\}$ | record exchange |
| $X$ | ::= | **Mobility types** | |
| | | $\curvearrowright$ | mobile |
| | | $\not\curvearrowright$ | immobile |
| $U$ | ::= | **Privacy types** | |
| | | $\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$ | |

## 5. A privacy type system for CCA

This section presents a novel type system for protecting privacy in CCA processes. Firstly the syntax of the type annotations is presented, then the definition of a notion of subtyping, and finally the typing rules that governs the proposed type system.

### 5.1. Types

In CCA, an ambient can exchange messages, can be mobile and can be aware of the presence of other ambients. These interactions among ambients, if not properly regulated, may lead to unwanted disclosure of private information, whether directly or indirectly. The types in Table 6 are devised to regulate communica-

**Table 7**
Free names in processes.

| | | |
|---|---|---|
| $\mathtt{fn}(\mathbf{0}) = \mathtt{fn}(\mathtt{out})$ | = | $\emptyset$ |
| $\mathtt{fn}(P \mid Q)$ | = | $\mathtt{fn}(P) \cup \mathtt{fn}(Q)$ |
| $\mathtt{fn}((\nu n : W)\ P)$ | = | $\mathtt{fn}(P) \backslash \{n\}$ |
| $\mathtt{fn}((\nu g : \mathtt{gr})\ P)$ | = | $\mathtt{fn}(P)$ |
| $\mathtt{fn}(n[P])$ | = | $\mathtt{fn}(P) \cup \{n\}$ |
| $\mathtt{fn}(!P)$ | = | $\mathtt{fn}(P)$ |
| $\mathtt{fn}(\kappa?P)$ | = | $\mathtt{fn}(\kappa) \cup \mathtt{fn}(P)$ |
| $\mathtt{fn}(x \triangleright (y_1, \ldots, y_\ell).P)$ | = | $\{x\}$ |
| $\mathtt{fn}(\mathtt{if}\ \kappa_1?M_1.P_1\ \ldots\ \kappa_\ell?M_\ell.P_\ell\ \mathtt{fi})$ | = | $\bigcup_{i=1}^{\ell} \mathtt{fn}(\kappa_i?M_i.P_i)$ |
| $\mathtt{fn}(\mathtt{in}\ n)$ | = | $\{n\}$ |
| $\mathtt{fn}(\alpha\ x(z_1, \ldots, z_\ell).P)$ | = | $\mathtt{fn}(\alpha) \cup \{x\} \cup \{z_1, \ldots, z_\ell\} \cup \mathtt{fn}(P)$ |
| $\mathtt{fn}(\alpha\ \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P)$ | = | $\mathtt{fn}(\alpha) \cup (\mathtt{fn}(P) \setminus \{y_1, \ldots, y_\ell\})$ |
| $\mathtt{fn}(\alpha\ \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P)$ | = | $\mathtt{fn}(\alpha) \cup \{z_1, \ldots, z_\ell\} \cup \mathtt{fn}(P)$ |
| $\mathtt{fn}(\uparrow) = \mathtt{fn}(\downarrow) = \mathtt{fn}(::) = \mathtt{fn}(\epsilon)$ | = | $\emptyset$ |
| $\mathtt{fn}(n \uparrow) = \mathtt{fn}(n \downarrow) = \mathtt{fn}(n ::)$ | = | $\{n\}$ |
| $\mathtt{fn}(\mathtt{true}) = \mathtt{fn}(\bullet)$ | = | $\emptyset$ |
| $\mathtt{fn}(n = m)$ | = | $\{n\} \cup \{m\}$ |
| $\mathtt{fn}(\neg\kappa) = \mathtt{fn}(\oplus\kappa) = \mathtt{fn}(\Diamond\kappa)$ | = | $\mathtt{fn}(\kappa)$ |
| $\mathtt{fn}(\kappa_1 \mid \kappa_2)$ | = | $\mathtt{fn}(\kappa_1) \cup \mathtt{fn}(\kappa_2)$ |
| $\mathtt{fn}(n[\kappa])$ | = | $\mathtt{fn}(\kappa) \cup \{n\}$ |
| $\mathtt{fn}(\kappa_1 \wedge \kappa_2)$ | = | $\mathtt{fn}(\kappa_1) \cup \mathtt{fn}(\kappa_2)$ |

**Table 8**
Free groups in processes.

| | | |
|---|---|---|
| $\mathtt{fg}(\mathbf{0}) = \mathtt{fg}(\mathtt{in}\ n) = \mathtt{fg}(\mathtt{out}) = \mathtt{fg}(\alpha) = \mathtt{fg}(Z)$ | = | $\emptyset$ |
| $\mathtt{fg}(P \mid Q)$ | = | $\mathtt{fg}(P) \cup \mathtt{fg}(Q)$ |
| $\mathtt{fg}((\nu n : W)\ P)$ | = | $\mathtt{fg}(P) \cup \mathtt{fg}(W)$ |
| $\mathtt{fg}((\nu g : \mathtt{gr})\ P)$ | = | $\mathtt{fg}(P) \backslash \{g\}$ |
| $\mathtt{fg}(n[P])$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(!P)$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(\kappa?P)$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(x \triangleright (y_1, \ldots, y_\ell).P)$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(\mathtt{if}\ \kappa_1?M_1.P_1\ \ldots\ \kappa_\ell?M_\ell.P_\ell\ \mathtt{fi})$ | = | $\bigcup_{i=1}^{\ell} \mathtt{fg}(\kappa_i?M_i.P_i)$ |
| $\mathtt{fg}(\alpha\ x(z_1, \ldots, z_\ell).P)$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(\alpha\ \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P)$ | = | $\mathtt{fg}(P) \cup \mathtt{fg}(\{l_1 : W_1, \ldots, l_\ell : W_\ell\})$ |
| $\mathtt{fg}(\alpha\ \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P)$ | = | $\mathtt{fg}(P)$ |
| $\mathtt{fg}(\mathtt{amb}(g)[X, T, U])$ | = | $\{g\} \cup \mathtt{fg}(T) \cup \mathtt{fg}(U)$ |
| $\mathtt{fg}(\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X, T, U])$ | = | $\mathtt{fg}(W_1) \cup \ldots \cup \mathtt{fg}(W_\ell) \cup \mathtt{fg}(T) \cup \mathtt{fg}(U)$ |
| $\mathtt{fg}(\mu Z.W)$ | = | $\mathtt{fg}(W)$ |
| $\mathtt{fg}(\{l_1 : W_1, \ldots, l_\ell : W_\ell\})$ | = | $\mathtt{fg}(W_1) \cup \ldots \cup \mathtt{fg}(W_\ell)$ |
| $\mathtt{fg}(\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}])$ | = | $\mathtt{fg}(\mathcal{R}) \cup \mathtt{fg}(\mathcal{O}) \cup \mathtt{fg}(\mathcal{I})$ |
| $\mathtt{fg}(\{g_1, \ldots, g_\ell\})$ | = | $\{g_1, \ldots, g_\ell\}$ |
| $\mathtt{fg}(*) = \mathtt{fg}(\mathtt{shh}) = \mathtt{fg}(\mathbf{1})$ | = | $\emptyset$ |
| $\mathtt{fg}(\mathtt{proc}[X, T, U])$ | = | $\mathtt{fg}(T) \cup \mathtt{fg}(U)$ |

tion, mobility and context-awareness in a manner that protects the privacy of individual ambients. There are four main categories of types: message types (denoted by $W$), exchange types ($T$), mobility types ($X$) and privacy types ($U$).

A message type $\mathtt{amb}(g)[X, T, U]$ is the type of an ambient of group $g$, mobility type $X$, exchange type $T$ and privacy type $U$. Likewise, a message type $\mathtt{abs}(W_1 \times \cdots \times W_\ell)[X, T, U]$ is the type of a process abstraction whose $i^{th}$ parameter is of the type $W_i$, $i = 1, \ldots, \ell$; and the body of this process abstraction is a process of mobility type $X$, exchange type $T$ and privacy type $U$. To enable an ambient to exchange names of that ambient type, we use a *recursive type* of the form $\mu Z.W$, where $Z$ is a type variable possibly occurring free in $W$. Note that all recursive types that can be defined in our type system are guarded, either by the keyword amb or abs. Because a recursive type is a finite representation of a possibly infinite type, it is important to introduce a notion of type equality. We write $W \sim W'$ to mean that two message types $W$ and $W'$ are equal. The relation $\sim$ is a congruence relation that satisfies the following axiom:

$$\frac{}{\mu Z.W \sim W\{\mu Z.W/Z\}} \tag{2}$$

It follows from this axiom that $\mu Z.W \sim W$ if $Z$ does not occur free in $W$; e.g. $\mu Z.\mathtt{amb}(g)[\curvearrowright, \mathtt{shh}, \mathtt{priv}[*, *, *]] \sim \mathtt{amb}(g)[\curvearrowright, \mathtt{shh}, \mathtt{priv}[*, *, *]]$. So, we will use $\mu Z.\mathtt{amb}(g)[X, T, U]$ as the general form of the ambient types; and $\mu Z.\mathtt{abs}(W_1 \times \cdots \times W_\ell)[X, T, U]$ as the general form of the process abstraction types.

The exchange types specify the communication interface of a process: shh if the process has no communication capability; and $\{l_1 : W_1, \ldots, l_\ell : W_\ell\}$ if the process has the capability of communicating a value of this record type, where $l_i$, $1 \le i \le \ell$ are the fields of the record type and are all different. By convention, the type of an empty record (i.e. $\ell = 0$) is denoted by $\mathbf{1}$. A subtyping relation '$\le$' is defined over exchange types as in Definition 5.1.

**Definition 5.1** (Subtyping)**.** The subtyping relation '$\le$' over exchange types is a pre-order relation defined as follows, for $n$ and $m$ non-negative integers:

$$\{l_1 : W_1, \ldots, l_{n+m} : W_{n+m}\} \le \{l_1 : W_1, \ldots, l_n : W_n\} \le \mathbf{1} \le \mathtt{shh}.$$

Similarly to the type system for mobile ambients (Cardelli et al., 1999), the mobility types state whether a process is mobile ($\curvearrowright$) or immobile ($\not\curvearrowright$).

The privacy types control the information flow among processes to ensure that only authorised ambients can access or share a specific piece of context information. A privacy type has the form $\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$, where $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{I}$ are sets of groups such that:

- $\mathcal{R}$: only ambients of these groups are allowed to *read* the context of any ambient of this privacy type. We say that an ambient $n$ reads the context of another ambient $m$ if the name $m$ occurs freely in a context-expression $\kappa$ or in a location term $\alpha$ (see grammar in Table 1) in the body of the ambient $n$. The notation $\mathcal{R}$ comes from the action *read*.
- $\mathcal{O}$: only ambients of these groups are allowed to *output to* (or share with) other ambients the context of any ambient of this privacy type. The notation $\mathcal{O}$ comes from the action *output*.
- $\mathcal{I}$: only ambients of these groups are allowed to *input* (or receive) the context of any ambient of this privacy type. So we use the notation $\mathcal{I}$ for the action *input*.

The intuition behind this privacy type is that an ambient must be able to decide who can read its context, who can share its context with a third-party and who that third-party might be. In this way, the context information is protected from unintended disclosure. By convention, we let $*$ denote the set of all groups. So, $\texttt{priv}[\emptyset,\emptyset,\emptyset]$ is the most restricted privacy type (no one can read or share the context information of an ambient of this privacy type) while $\texttt{priv}[*,*,*]$ is the least restricted one (the context of an ambient of this privacy type can be read by everyone and shared with anyone). Consequently, if $A$ is a set of groups then $*\cup A = *$ and $*\cap A = A$.

**Example 5.1** (Privacy Types)**.** Let $g_1$, $g_2$, $g_3$ and $g_4$ be four distinct groups. We have the following privacy types:

1. $\texttt{priv}[\{g_1\},\emptyset,\emptyset]$: ambients of group $g_1$ are allowed to read the context of ambients of this privacy type, but are not allowed to share them with any other ambient.
2. $\texttt{priv}[\{g_1\},\{g_1\},\{g_2\}]$: ambients of group $g_1$ are allowed to read the context of ambients of this privacy type, and to share them solely with ambients of group $g_2$.
3. $\texttt{priv}[\{g_1,g_2\},\{g_1\},\{g_3\}]$: ambients of group $g_1$ or $g_2$ are allowed to read the context of ambients of this privacy type, but solely ambients of group $g_1$ are allowed to share this context with ambients of group $g_3$. However, ambients of group $g_3$ are not allowed to share this context further with any other ambient.
4. $\texttt{priv}[\{g_1,g_2\},\{g_1,g_3\},\{g_3,g_4\}]$: similar to (3), but ambients of group $g_3$ are now allowed to shared the context further with ambients of group $g_4$ and among themselves.
5. $\texttt{priv}[\{g_1\},\{g_1\},*]$: ambients of group $g_1$ are allowed to read the context of ambients of this privacy type, and to share them with any other ambient. The group $g_1$ can be thought of as that of a *trusted server* which is allowed to acquire context and share them with the ambients that it trusts, without passing to these ambients the right to share this context further.

**Example 5.2** (Exchange and mobility types)**.** In the following types, $g$ and $h$ are groups:

- $\texttt{amb}(g)[\frown,\texttt{shh},\texttt{priv}[\emptyset,\emptyset,\emptyset]]$: ambients of this type are of group $g$, are mobile, quiet (no exchange), and secretive (do not share their context).
- $\texttt{amb}(g)[\not\frown,\mathbf{1},\texttt{priv}[*,*,*]]$: ambients of this type are of group $g$, immobile, can exchange signals (empty record), and are happy to share their context with anyone.
- $\mu Z.\texttt{amb}(g)[\frown,\{l_1:Z,l_2:\texttt{amb}(h)[\frown,\texttt{shh},\texttt{priv}[\emptyset,\emptyset,\emptyset]]\},\texttt{priv}[*,*,*]]$: ambients of this type are of group $g$, mobile, can exchange values of a record type whose one of the fields is recursively of this type, and are happy to share their context with anyone.

**Definition 5.2** (Privacy type consistency relation)**.** A privacy type $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]$ is consistent with a privacy type $\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']$ and we write $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]\sqsubseteq\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']$ if $\mathcal{R}\subseteq\mathcal{R}'$ and $\mathcal{O}\subseteq\mathcal{O}'$ and $\mathcal{I}\subseteq\mathcal{I}'$.

It follows from Definition 5.2 that if $U\sqsubseteq U'$ then the privacy type $U$ cannot grant any group a right that is not granted to that group by $U'$. However, a group may be granted more rights in $U'$ than in $U$. Note that this consistency relation is a partial order relation over privacy types as stated in Theorem 5.1.

**Theorem 5.1** (Privacy type consistency relation)**.** *The privacy type consistency relation $\sqsubseteq$ is a partial order relation over privacy types.*

**Proof.** The proof is straightforward from Definition 5.2.

- **Reflexivity** Since $\mathcal{R}\subseteq\mathcal{R}$ and $\mathcal{O}\subseteq\mathcal{O}$ and $\mathcal{I}\subseteq\mathcal{I}$, from Definition 5.2 we conclude that $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]\sqsubseteq\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]$.
- **Antisymmetry** Suppose that $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]\sqsubseteq\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']$ and $\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']\sqsubseteq\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]$. By Definition 5.2 we have $\mathcal{R}\subseteq\mathcal{R}'$ and $\mathcal{O}\subseteq\mathcal{O}'$ and $\mathcal{I}\subseteq\mathcal{I}'$ and $\mathcal{R}'\subseteq\mathcal{R}$ and $\mathcal{O}'\subseteq\mathcal{O}$ and $\mathcal{I}'\subseteq\mathcal{I}$; which imply that $\mathcal{R}=\mathcal{R}$ and $\mathcal{O}=\mathcal{O}$ and $\mathcal{I}=\mathcal{I}$.
- **Transitivity** Suppose that $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]\sqsubseteq\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']$ and $\texttt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']\sqsubseteq\texttt{priv}[\mathcal{R}'',\mathcal{O}'',\mathcal{I}'']$. By Definition 5.2 we have $\mathcal{R}\subseteq\mathcal{R}'$ and $\mathcal{O}\subseteq\mathcal{O}'$ and $\mathcal{I}\subseteq\mathcal{I}'$ and $\mathcal{R}'\subseteq\mathcal{R}''$ and $\mathcal{O}'\subseteq\mathcal{O}''$ and $\mathcal{I}'\subseteq\mathcal{I}''$; which imply that $\mathcal{R}\subseteq\mathcal{R}''$ and $\mathcal{O}\subseteq\mathcal{O}''$ and $\mathcal{I}\subseteq\mathcal{I}''$.

□

**Example 5.3** (Privacy type consistency relation)**.**

- $\texttt{priv}[\emptyset,\emptyset,\emptyset]$ is the least privacy type with respect to the consistency relation $\sqsubseteq$, i.e. $\texttt{priv}[\emptyset,\emptyset,\emptyset]\sqsubseteq U$, for all privacy type $U$.
- $\texttt{priv}[*,*,*]$ is the greatest privacy type with respect to the consistency relation $\sqsubseteq$, i.e. $U\sqsubseteq\texttt{priv}[*,*,*]$, for all privacy type $U$.
- $\texttt{priv}[\{g\},\{g,h\},\{h\}]\sqsubseteq\texttt{priv}[\{g\},\{g,h\},\{h,t\}]$
- $\texttt{priv}[\{g\},\{g,h\},\{h\}]\not\sqsubseteq\texttt{priv}[\{g\},\{g,h\},\{t\}]$ and $\texttt{priv}[\{g\},\{g,h\},\{t\}]\not\sqsubseteq\texttt{priv}[\{g\},\{g,h\},\{h\}]$, .i.e. these two privacy types are not comparable with respect to $\sqsubseteq$ (partial order relation).
- $\texttt{priv}[*,\{g,h\},\{h\}]\not\sqsubseteq\texttt{priv}[\{g\},\{g,h\},\{h,t\}]$

For the sake of simplicity, given a privacy type $\texttt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]$, we will refer to the set $\mathcal{R}$ (respectively $\mathcal{O}$, $\mathcal{I}$) as the $\mathcal{R}$-component (respectively $\mathcal{O}$-component, $\mathcal{I}$-component) of that privacy type.

### 5.2. Typing rules

We formalise the type system using *typing rules*. The terms that can be typed using these rules are the *well-typed* terms. If $n$ is a name and $A$ a type, we denote by $n$: $A$ the *assignment* of the type $A$ to the name $n$; the name $n$ is called the name of that assignment, while $A$ is called the type of that assignment. A *type environment* (aka *type assumption*) is a finite set of assignments of types to names, where the names in the assignments are all different. We let $\Gamma$ range over type environments, and $\texttt{dom}(\Gamma)$ denote the set of the names of the assignments in $\Gamma$.

#### 5.2.1. Type judgements

A type judgement has the form $\Gamma\vdash G$: $A$, $\Gamma\vdash A$ or $\Gamma\vdash\diamond$, where $A$ is a type and $G$ is a process, a context expression or a name as shown in Table 9. A type judgement $\Gamma\vdash G$: $A$ asserts that $G$ has the type $A$ under the type assumption $\Gamma$; $\Gamma\vdash A$ means that $A$ is a good type under the type assumption $\Gamma$; while $\Gamma\vdash\diamond$ means that the type assumption $\Gamma$ is well-formed (or good) in the sense that the names in assignments are all different. Given a type system, a *valid* type judgement is one that can be proved from the axioms and inference rules of the type system. We now present the axioms and the inference rules of the proposed type system.

#### 5.2.2. Type environment rules

A good type environment is built using the inference rules in Table 10. Basically these rules say that an empty environment is a good type environment (the rule E1), as so is a good type environment to which an assignment of a good type to a new name or new group is added (the rules E2 and E3). The rules E4 and E5 state how a valid type judgement for a message or a group can be

**Table 9**
Type judgements.

| | |
|---|---|
| $\Gamma \vdash \diamond$ | good environment |
| $\Gamma \vdash W$ | good message type $W$ |
| $\Gamma \vdash T$ | good exchange type $T$ |
| $\Gamma \vdash U$ | good privacy type $U$ |
| $\Gamma \vdash n$: $W$ | good message $n$ of type $W$ |
| $\Gamma \vdash g : \mathtt{gr}$ | good group $g$ |
| $\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}]$ | good context expression $\kappa$ of disclosure set $\mathcal{R}$ (see Definition 5.3) |
| $\Gamma \vdash M : \mathtt{proc}[X, T, U]$ | good capability with mobility type $X$, exchange type $T$ and privacy type $U$ |
| $\Gamma \vdash P : \mathtt{proc}[X, T, U]$ | good process $P$ with mobility type $X$, exchange type $T$ and privacy type $U$ |

**Table 10**
Environment rules.

$$\textbf{(E1)}\ \frac{}{\emptyset \vdash \diamond} \qquad \textbf{(E2)}\ \frac{\Gamma \vdash W \quad n \notin \mathtt{dom}(\Gamma)}{\Gamma, n : W \vdash \diamond} \qquad \textbf{(E3)}\ \frac{\Gamma \vdash \diamond \quad g \notin \mathtt{dom}(\Gamma)}{\Gamma, g : \mathtt{gr} \vdash \diamond}$$

$$\textbf{(E4)}\ \frac{\Gamma', n : W, \Gamma'' \vdash \diamond}{\Gamma', n : W, \Gamma'' \vdash n : W} \qquad \textbf{(E5)}\ \frac{\Gamma', g : \mathtt{gr}, \Gamma'' \vdash \diamond}{\Gamma', g : \mathtt{gr}, \Gamma'' \vdash g : \mathtt{gr}} \qquad \textbf{(E6)}\ \frac{\Gamma \vdash n : W \quad W \sim W'}{\Gamma \vdash n : W'}$$

**Table 11**
Good types rules.

$$\textbf{(A1)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathtt{priv}[\emptyset, \emptyset, \emptyset]} \qquad \textbf{(A2)}\ \frac{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}] \quad \Gamma \vdash g : \mathtt{gr}}{\Gamma \vdash \mathtt{priv}[\mathcal{R} \cup \{g\}, \mathcal{O}, \mathcal{I}]} \qquad \textbf{(A3)}\ \frac{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}] \quad \Gamma \vdash g : \mathtt{gr}}{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O} \cup \{g\}, \mathcal{I}]}$$

$$\textbf{(A4)}\ \frac{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}] \quad \Gamma \vdash g : \mathtt{gr}}{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I} \cup \{g\}]} \qquad \textbf{(A5)}\ \frac{\Gamma \vdash \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}] \quad \mathcal{R}' \in \{\mathcal{R}, *\} \quad \mathcal{O}' \in \{\mathcal{O}, *\} \quad \mathcal{I}' \in \{\mathcal{I}, *\}}{\Gamma \vdash \mathtt{priv}[\mathcal{R}', \mathcal{O}', \mathcal{I}']}$$

$$\textbf{(A6)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathtt{shh}} \qquad \textbf{(A7)}\ \frac{\Gamma \vdash W_1 \quad \ldots \quad \Gamma \vdash W_\ell \quad i \neq j \Rightarrow l_i \neq l_j \quad 1 \leq i, j \leq \ell}{\Gamma \vdash \{l_1 : W_1, \ldots, l_\ell : W_\ell\}}$$

$$\textbf{(A8)}\ \frac{\Gamma \vdash g : \mathtt{gr} \quad \Gamma \vdash T \quad \Gamma \vdash U}{\Gamma \vdash \mu Z.\mathtt{amb}(g)[X, T, U]} \qquad \textbf{(A9)}\ \frac{\Gamma \vdash W_1 \ \ldots \ \Gamma \vdash W_\ell \quad \Gamma \vdash T \quad \Gamma \vdash U}{\Gamma \vdash \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X, T, U]}$$

$$\textbf{(A10)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash Z} \qquad \textbf{(A11)}\ \frac{\Gamma \vdash W \quad W \sim W'}{\Gamma \vdash W'}$$

inferred from a good type environment. Finally, the rule E6 says that a well-typed name of type $W$ under a type assumption $\Gamma$ can also be assigned any type equal to $W$ under $\Gamma$.

*5.2.3. Good types rules*

A good type is constructed using the inference rules in Table 11. The rules A1 to A5 define a good privacy type. Indeed, the $\mathcal{R}$- ($\mathcal{O}$- and $\mathcal{I}$-, respectively) component of a good privacy type under a given type assumption $\Gamma$ is either $\emptyset$, $*$, or a subset of groups that belong to $\mathtt{dom}(\Gamma)$. The rule A6 says that `shh` is a good exchange type, as is any record type whose labels are all different and of good message types. Good message types are defined by the rules A8 and A9. The rule A10 says that a type variable is a good type and the rule A11 states that any type equal to a good message type is also a good message type.

**Definition 5.3** (Disclosure set)**.** Let $\Gamma$ be a type assumption, and $\kappa$ a context expression. The *disclosure set* of $\kappa$ under $\Gamma$ is:

- the set $*$ of all groups, if no ambient name occurs in $\kappa$; or
- the intersection of the $\mathcal{R}$-components of the privacy types of all the ambient names that occur in the context expression $\kappa$, under the type assumption $\Gamma$.

**Example 5.4** (Disclosure set)**.** Suppose that $\Gamma \vdash n : \mathtt{amb}(g)[\curvearrowright, \mathbf{1}, \mathtt{priv}[\{h, t\}, \{v\}, \{u\}]]$ and $\Gamma \vdash m : \mathtt{amb}(h)[\not\curvearrowright, \mathbf{1}, \mathtt{priv}[\{g, t\}, \{v\}, \{u\}]]$ are valid type judgements for some type assumption $\Gamma$; ambients $n$ and $m$; and groups $g$, $h$, $t$, $u$, and $v$.

- The disclosure set of the context expression '$\oplus(\bullet \mid \mathtt{true})$' is $*$.
- The disclosure set of the context expression '$n[\mathtt{true}]$' is $\{h, t\}$.
- The disclosure set of the context expression '$n[\mathtt{true}] \mid m[\mathtt{true}] \mid \mathtt{true}$' is $\{t\}$.

*5.2.4. Context expression typing rules*

The typing rules for context expressions are given in Table 12. A type judgement of the form $\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}]$ means that ambients of groups in the set $\mathcal{R}$ are allowed to read the context of any ambient that occurs in the context expression $\kappa$, under the type assumption $\Gamma$. That is, $\mathcal{R}$ is the disclosure set of $\kappa$ under $\Gamma$ (see Definition 5.3). It follows that the privacy type of the context expression `true` is $\mathtt{cont}[*]$ (the rule K1); rightly so because `true` discloses the name of no ambient. For the same reason, the hole context has the same privacy type (the rule K2). Same for the matching of two names of identical type that are not ambient names (the rule K3). However, if these are ambient names, then the disclosure set for the matching coincides with the $\mathcal{R}$-component of the privacy type of these ambients (the rule K4). The rules K5, K7 and K9 say that $\neg\kappa$, $\oplus\kappa$ and $\diamond\kappa$ take the type of $\kappa$, respectively. The disclosure set for $\kappa_1|\kappa_2$ and $\kappa_1 \wedge \kappa_2$ is the intersection of the disclosure set for $\kappa_1$ and the disclosure set for $\kappa_2$ (rules K6 and K8). If $n$ is an ambient and $\kappa$ a context expression, then the disclosure set for $n[\kappa]$ is the intersection of the $\mathcal{R}$-component of the privacy type of $n$ and the disclosure set for $\kappa$ (the rule K10). The rule K11 says that equivalent context expressions have the same type under a given type assumption.

**Example 5.5.** Suppose that $\Gamma \vdash n : \mathtt{amb}(g)[\curvearrowright, \mathbf{1}, \mathtt{priv}[\{h, t\}, \{v\}, \{u\}]]$ and $\Gamma \vdash m : \mathtt{amb}(h)[\not\curvearrowright, \mathbf{1}, \mathtt{priv}[\{g, t\}, \{v\}, \{u\}]]$ are valid type judgements for some type assumption $\Gamma$; ambients $n$ and $m$; and groups $g$, $h$, $t$, $u$, and $v$. The following type judgements are valid.

- $\Gamma \vdash \oplus(\bullet \mid \mathtt{true}) : \mathtt{cont}[*]$
- $\Gamma \vdash n[\mathtt{true}] : \mathtt{cont}[\{h, t\}]$
- $\Gamma \vdash \oplus\ (\bullet \mid n[\mathtt{true}] \mid \mathtt{true}) : \mathtt{cont}[\{h, t\}]$
- $\Gamma \vdash n[\oplus(\bullet \mid \mathtt{true})] \mid \mathtt{true} : \mathtt{cont}[\{h, t\}]$
- $\Gamma \vdash n[\mathtt{true}] \mid m[\mathtt{true}] \mid \mathtt{true} : \mathtt{cont}[\{t\}]$

*5.2.5. Capability typing rules*

In CCA, context information of ambients may be leaked through a context expression, an input capability, or an output capability.

**Definition 5.4** (Exposed ambient)**.** An ambient name $n$ is *exposed* in a process $P$ if that name occurs in a context expression, an input

**Table 12**
Context expression typing rules.

$$\textbf{(K1)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathtt{true} : \mathtt{cont}[*]} \qquad \textbf{(K2)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \bullet : \mathtt{cont}[*]} \qquad \textbf{(K3)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X,T,U] \quad \Gamma \vdash m : \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X,T,U]}{\Gamma \vdash n = m : \mathtt{cont}[*]}$$

$$\textbf{(K4)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \quad \Gamma \vdash m : \mu Z.\mathtt{amb}(g)[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]}{\Gamma \vdash n = m : \mathtt{cont}[\mathcal{R}]} \qquad \textbf{(K5)}\ \frac{\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}]}{\Gamma \vdash \oplus\kappa : \mathtt{cont}[\mathcal{R}]} \qquad \textbf{(K6)}\ \frac{\Gamma \vdash \kappa_1 : \mathtt{cont}[\mathcal{R}_1] \quad \Gamma \vdash \kappa_2 : \mathtt{cont}[\mathcal{R}_2]}{\Gamma \vdash \kappa_1|\kappa_2 : \mathtt{cont}[\mathcal{R}_1 \cap \mathcal{R}_2]}$$

$$\textbf{(K7)}\ \frac{\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}]}{\Gamma \vdash \neg\kappa : \mathtt{cont}[\mathcal{R}]} \qquad \textbf{(K8)}\ \frac{\Gamma \vdash \kappa_1 : \mathtt{cont}[\mathcal{R}_1] \quad \Gamma \vdash \kappa_2 : \mathtt{cont}[\mathcal{R}_2]}{\Gamma \vdash \kappa_1 \wedge \kappa_2 : \mathtt{cont}[\mathcal{R}_1 \cap \mathcal{R}_2]} \qquad \textbf{(K9)}\ \frac{\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}]}{\Gamma \vdash \diamond\kappa : \mathtt{cont}[\mathcal{R}]}$$

$$\textbf{(K10)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X,T,\mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]] \quad \Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}_2]}{\Gamma \vdash n[\kappa] : \mathtt{cont}[\mathcal{R}_1 \cap \mathcal{R}_2]} \qquad \textbf{(K11)}\ \frac{\Gamma \vdash \kappa_1 : \mathtt{cont}[\mathcal{R}] \quad \models \kappa_1 \Leftrightarrow \kappa_2}{\Gamma \vdash \kappa_2 : \mathtt{cont}[\mathcal{R}]}$$

**Table 13**
Capability typing rules.

$$\textbf{(C1)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X',T,U'] \quad \Gamma \vdash P : \mathtt{proc}[X,T,U]}{\Gamma \vdash \mathtt{in}\ n.P : \mathtt{proc}[\curvearrowright,T,U]} \qquad \textbf{(C2)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X,T,U]}{\Gamma \vdash \mathtt{out}.P : \mathtt{proc}[\curvearrowright,T,U]}$$

$$\textbf{(C3)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X',T',U'] \quad \Gamma \vdash P : \mathtt{proc}[X,T,U]}{\Gamma \vdash \mathtt{del}\ n.P : \mathtt{proc}[X,T,U]}$$

$$\textbf{(C4)}\ \frac{\begin{array}{c}\Gamma, y_1 : W_1, \ldots, y_\ell : W_\ell \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \\ W_{i_j} = \mu Z.\mathtt{amb}(g_{i_j})[X_{i_j},T_{i_j},\mathtt{priv}[\mathcal{R}_{i_j},\mathcal{O}_{i_j},\mathcal{I}_{i_j}]] \\ 1 \le j \le r \qquad i_j \in \{1,\ldots,\ell\} \qquad j \ne k \Rightarrow i_j \ne i_k \qquad r \le \ell \\ \Gamma \vdash \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \qquad T \le \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \qquad \mathcal{I}' = (\overset{r}{\underset{j=1}{\Cap}} \mathcal{I}_{i_j}) \cap \mathcal{I}\end{array}}{\Gamma \vdash \theta\ \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}']]}$$

$$\textbf{(C5)}\ \frac{\begin{array}{c}\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X',T,\mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]] \qquad \mathcal{R}' = \mathcal{R}_1 \cap \mathcal{R} \\ \Gamma \vdash \theta\ \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]\end{array}}{\Gamma \vdash n\,\theta\,\mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}',\mathcal{O},\mathcal{I}]]}$$

$$\textbf{(C6)}\ \frac{\begin{array}{c}\Gamma \vdash \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \qquad \Gamma \vdash z_1 : W_1 \quad \ldots \quad \Gamma \vdash z_\ell : W_\ell \\ W_{i_j} = \mu Z.\mathtt{amb}(g_{i_j})[X_{i_j},T_{i_j},\mathtt{priv}[\mathcal{R}_{i_j},\mathcal{O}_{i_j},\mathcal{I}_{i_j}]] \\ 1 \le j \le r \qquad i_j \in \{1,\ldots,\ell\} \qquad j \ne k \Rightarrow i_j \ne i_k \qquad r \le \ell \\ \Gamma \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]] \qquad T \le \{l_1 : W_1, \cdots, l_\ell : W_\ell\} \qquad \mathcal{O}' = (\overset{r}{\underset{j=1}{\Cap}} \mathcal{O}_{i_j}) \cap \mathcal{O}\end{array}}{\Gamma \vdash \theta\ \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O}',\mathcal{I}]]}$$

$$\textbf{(C7)}\ \frac{\begin{array}{c}\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X',T,\mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]] \quad \mathcal{R}' = \mathcal{R}_1 \cap \mathcal{R} \\ \Gamma \vdash \theta\ \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]\end{array}}{\Gamma \vdash n\ \theta\ \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}',\mathcal{O},\mathcal{I}]]}$$

$$\textbf{(C8)}\ \frac{\Gamma \vdash x : \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X,T,U] \quad \Gamma \vdash z_1 : W_1 \quad \ldots \quad \Gamma \vdash z_\ell : W_\ell}{\Gamma \vdash \theta\ x(z_1,\ldots,z_\ell) : \mathtt{proc}[X,T,U]} \qquad \textbf{(C9)}\ \frac{\Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X',T,\mathtt{priv}[\mathcal{R}',\mathcal{O}',\mathcal{I}']] \quad \Gamma \vdash \theta\ x(z_1,\ldots,z_\ell) : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]}{\Gamma \vdash n\ \theta\ x(z_1,\ldots,z_\ell) : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}' \cap \mathcal{R},\mathcal{O},\mathcal{I}]]}$$

capability, or an output capability, outside the bodies of all ambients in $P$.

**Example 5.6** (Exposed ambient)**.** Consider the process

$$n :: \mathtt{send}().\mathtt{recv}(\{l_1 = (u : W)\}).\mathbf{0} \mid t[m :: \mathtt{recv}().\mathbf{0}],$$

where $n$, $u$, $t$ and $m$ are ambient names. Then $n$ and $u$ are exposed names in this process, while $m$ and $t$ are not.

The typing rules for capabilities are given in Table 13, where a type judgement of the form $\Gamma \vdash M.P : \mathtt{proc}[X,T,U]$ asserts that $M.P$ is a process of mobility type $X$, exchange type $T$ and privacy type $U$. The mobility type $X$ indicates whether the process $M.P$ contains any mobility capability such as `in` or `out`; while the exchange type $T$ specifies the type of messages this process can exchange. The privacy type $U$ of a well-typed process $M.P$ is any privacy type which is consistent with (in the sense of Definition 5.2) the privacy of the continuation process $P$ and consistent with the privacy type of each ambient exposed in the capability $M$. The rule C1 in Table 13 says that if $n$ is an ambient of exchange type $T$, and $P$ a process of mobility type $X$ and exchange type $T$; then the process '`in` $n.P$' has the mobility type $\curvearrowright$, the exchange $T$ and the same privacy type as $P$. It is important that the ambient $n$ and the process '`in` $n.P$' have the same exchange type so that an ambient executing that process to move inside $n$ be able to communicate safely with $n$. Similarly, the rule C2 asserts that any process of the form `out`$.P$ is always of mobility type $\curvearrowright$, while the rule C3 states that the process '`del` $n.P$' takes the same type as $P$ provided $n$ is an ambient name.

In the rules C4 to C9, $\theta$ belongs to the set $\{\uparrow, \downarrow, :, :, \epsilon\}$. We also use the following notation where $B_i$, $1 \le i \le r$ and $0 \le r$ are sets of groups:

$$\overset{r}{\underset{i=1}{\Cap}} B_i = \begin{cases} * & \text{if } r = 0 \\ \overset{r}{\underset{i=1}{\cap}} B_i & \text{otherwise.} \end{cases}$$

The rule C4 says that the type of the argument of an input capability $M$ is a record type; and the exchange type of the continuation process $P$ must be a subtype of that record type. If some of the fields of that record type are ambient names, then the $\mathcal{I}$-component of the privacy type of $M.P$ must be equal to the intersection of that of the privacy type of the continuation process $P$ and the $\mathcal{I}$-components of the privacy types of these fields. Recall that the $\mathcal{I}$-component of a process privacy type is the set of groups and the contexts of the ambient names exposed in that process can be shared only with ambients of those groups. The intersection is justified by the fact that an ambient that receives another ambient's name must belong to a group the latter would like to share its context with. This will become clearer with the rule P5 of Table 14 explained in Section 5.2.6. When the sender $n$ is specified (i.e. $M$ is prefixed as $n\ \theta$ `recv`) like in the rule C5, then the

**Table 14**
Process typing rules.

$$\textbf{(P1)}\ \frac{\Gamma \vdash T}{\Gamma \vdash \mathbf{0} : \mathtt{proc}[X, T, \mathtt{priv}[*, *, *]]} \qquad \textbf{(P4)}\ \frac{\Gamma, g : \mathtt{gr} \vdash P : \mathtt{proc}[X, T, U] \quad g \notin \mathtt{fg}(\mathtt{proc}[X, T, U])}{\Gamma \vdash (\nu\ g : \mathtt{gr})P :\ \mathtt{proc}[X, T, U]}$$

$$\textbf{(P2)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X, T, U]}{\Gamma \vdash\ !P :\ \mathtt{proc}[X, T, U]} \qquad \textbf{(P5)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]] \quad \Gamma \vdash n : \mu Z.\mathtt{amb}(g)[X, T, U] \quad g \in \mathcal{R} \cap \mathcal{O} \cap \mathcal{I}}{\Gamma \vdash n[P] :\ \mathtt{proc}[X', T, \mathtt{priv}[*, *, *]]}$$

$$\textbf{(P3)}\ \frac{\Gamma, n : W \vdash P : \mathtt{proc}[X, T, U]}{\Gamma \vdash (\nu\ n : W)P :\ \mathtt{proc}[X, T, U]} \qquad \textbf{(P6)}\ \frac{\Gamma \vdash x : \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X, T, U] \quad \Gamma \vdash T' \quad \Gamma, y_1 : W_1, \ldots, y_\ell : W_\ell \vdash P : \mathtt{proc}[X, T, U]}{\Gamma \vdash x \triangleright (y_1, \ldots, y_\ell).P : \mathtt{proc}[X', T', \mathtt{priv}[*, *, *]]}$$

$$\textbf{(P7)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]] \quad \Gamma \vdash Q : \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}', \mathcal{O}', \mathcal{I}']]}{\Gamma \vdash P|Q :\ \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R} \cap \mathcal{R}', \mathcal{O} \cap \mathcal{O}', \mathcal{I} \cap \mathcal{I}']]}$$

$$\textbf{(P8)}\ \frac{\Gamma \vdash M.P : \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}_1, \mathcal{O}, \mathcal{I}]] \quad \kappa : \mathtt{cont}[\mathcal{R}_2]}{\Gamma \vdash \kappa?M.P :\ \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}_1 \cap \mathcal{R}_2, \mathcal{O}, \mathcal{I}]]} \qquad \textbf{(P9)}\ \frac{\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}] \quad \Gamma \vdash M.P : \mathtt{proc}[X, T, \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]] \quad \Gamma \vdash \mathcal{C}(M.P) : \mathtt{proc}[X', T, U']}{\Gamma \vdash \mathcal{C}(\kappa?M.P) :\ \mathtt{proc}[X', T, U']}$$

$$\textbf{(P10)}\ \frac{\Gamma \vdash \mathcal{C}(\kappa_i?M_i.P_i) : \mathtt{proc}[X, T, U], 1 \le i \le \ell}{\Gamma \vdash \mathcal{C}(\mathtt{if}\ \kappa_1?M_1.P_1\ \ldots\ \kappa_\ell?M_\ell.P_\ell\ \mathtt{fi}) : \mathtt{proc}[X, T, U]} \qquad \textbf{(P11)}\ \frac{\Gamma \vdash P : \mathtt{proc}[X, T, U] \quad U' \sqsubseteq U \quad T' \le T}{\Gamma \vdash P : \mathtt{proc}[X, T', U']}$$

$\mathcal{R}$-component of the privacy type of the process *M.P* is the intersection of the $\mathcal{R}$-component of the privacy type of the sender *n* and that of the privacy type of the continuation process *P*. This is because such a capability reads the context (e.g. the location $\theta$) of the sender; hence must be allowed by the sender to do so.

The rule C6 says that the type of the argument of an output capability *M* is a record type; and the exchange type of the continuation process *P* must be a subtype of that record type. If some of the fields of that record type are ambient names, then the $\mathcal{O}$-component of the privacy type of *M.P* must be equal to the intersection of that of the privacy type of the continuation process *P* and the $\mathcal{O}$-components of the privacy types of these fields. The $\mathcal{O}$-component of a process privacy type is the set of the groups of the ambients that are allowed to share the contexts of all the ambients exposed in that process. The intersection is justified by the fact that an ambient that sends out another ambient's name must be allowed by the latter to do so. When the receiver *n* is specified (i.e. *M* is prefixed as $n\ \theta$ `send`) like in the rule C7, then the $\mathcal{R}$-component of the privacy type of the process *M.P* is the intersection of the $\mathcal{R}$-component of the privacy type of the sender *n* and that of the privacy type of the continuation process *P*. This is because such a capability reads the context (e.g. the location $\theta$) of the receiver; hence must be allowed by the receiver to do so.

The rule C8 asserts that if *x* is a process abstraction name of a given type, then a process call to *x* such that the types of the actual parameters match those of the formal parameters is a well-typed capability of the same mobility, exchange and privacy types as the name *x*. If in addition the ambient *n* where the process abstraction *x* is defined is specified (i.e. *M* is prefixed as $n\theta x$) like in the rule C9, then the $\mathcal{R}$-component of the privacy type of the process abstraction call *M* is the intersection of the $\mathcal{R}$-component of the privacy type of the ambient *n* and that of the privacy type of the process abstraction *x*.

**Example 5.7.** Let $W_1 = \mathtt{amb}(g)[\curvearrowright, \mathbf{1}, \mathtt{priv}[\{h, t\}, \{v\}, \{u\}]]$ and $W_2 = \mathtt{amb}(h)[\not\curvearrowright, \mathbf{1}, \mathtt{priv}[\{g, t\}, \{v\}, \{u\}]]$ be two message types. Suppose that $\Gamma \vdash n: W_1$ and $\Gamma \vdash m: W_2$ are valid type judgements for some type assumption $\Gamma$; ambients *n* and *m*; and groups *g, h, t, u*, and *v*. The following type judgements are also valid.

- $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[*, *, \{u\}]]$
- $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1), l_2 = (y_2 : W_2)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[*, *, \{u\}]]$
- $\Gamma \vdash m :: \mathtt{recv}(\{l_1 = (y_1 : W_1)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[\{g, t\}, *, \{u\}]]$
- $\Gamma \vdash \mathtt{send}(\{l_1 = n\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[*, \{v\}, *]]$
- $\Gamma \vdash \mathtt{send}(\{l_1 = n, l_2 = m\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[*, \{v\}, *]]$
- $\Gamma \vdash m :: \mathtt{send}(\{l_1 = n\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[\{g, t\}, \{v\}, *]]$
- $\Gamma \vdash \mathtt{in}\ m.\mathbf{0} : \mathtt{proc}[\curvearrowright, \mathtt{shh}, \mathtt{priv}[*, *, *]]$
- $\Gamma \vdash \mathtt{out}.\mathbf{0} : \mathtt{proc}[\curvearrowright, \mathtt{shh}, \mathtt{priv}[*, *, *]]$

### 5.2.6. Process typing rules

The typing rules of processes are depicted in Table 14. The inactivity process $\mathbf{0}$ can take any mobility type and any exchange type; but its privacy type is set to $\mathtt{priv}[*, *, *]$ (the rule P1). Although this privacy type allows the process $\mathbf{0}$ to read the information of any ambient and share it with any other ambient, this will never happen because this process is no able to perform any capabilities. However, such a privacy type does not affect the privacy type of any process that contains the process $\mathbf{0}$. The processes $!P$ and $(\nu\ n: W)P$ take the type of the process *P* (rules P2 and P3, respectively). Same for the process $(\nu\ g : \mathtt{gr})P$, provided the group *g* does not occur free in the type of *P* (the rule P4). The rule P5 is very important and states whether an ambient behaves in a way that does not violate the privacy of other ambients. This rule says that if *n* is an ambient of group *g* and *P* a well-typed process with the same mobility type and the same exchange type as *n* such that *g* belongs to the $\mathcal{R}$-, $\mathcal{O}$- and $\mathcal{I}$-components of the privacy type of *P*, then *n*[*P*] is a well-typed process with the same exchange type as *n*. Indeed, if the ambient *n*'s group does not belong to one of these sets, then the process *n*[*P*] may violate the privacy of some ambient at runtime as illustrated in Example 5.8.

**Example 5.8.** Let $W_1 = \mathtt{amb}(g)[\curvearrowright, \{l_1 : W_2\}, \mathtt{priv}[\{h, t\}, \{v\}, \{u\}]]$ and $W_2 = \mathtt{amb}(h)[\not\curvearrowright, \mathbf{1}, \mathtt{priv}[\{t\}, \{g, v\}, \{g, u\}]]$ be two message types. Suppose that $\Gamma \vdash n: W_1$ and $\Gamma \vdash m: W_2$ are valid type judgements for some type assumption $\Gamma$; ambients *n* and *m*; and groups *g, h, t, u*, and *v*. Then the type judgement in Eq. (3) is valid.

$$\Gamma \vdash\ \ \Diamond(m[\mathtt{true}] \mid \mathtt{true})? :: \mathtt{send}().\mathbf{0} \mid\ \uparrow \mathtt{recv}(\{l_1 = (y_1 : W_2)\}) \cdot \mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_2\}, \mathtt{priv}[\{t\}, *, \{g, u\}]] \tag{3}$$

$$n[\Diamond(m[\mathtt{true}] \mid \mathtt{true})? :: \mathtt{send}().\mathbf{0} \mid\ \uparrow \mathtt{recv}(\{l_1 = (y_1 : W_2)\}).\mathbf{0}] \tag{4}$$

Now consider the process in Eq. (4) which is an ambient named *n* whose body is exactly the process in the type judgement depicted in Eq. (3). Note that the ambient *n* can sense the presence of the ambient *m* through the context expression $\Diamond(m[\mathtt{true}] \mid \mathtt{true})$ which holds if there is an ambient named *m* somewhere in the

system. Therefore the ambient $n$ so specified violates the privacy requirement (see $W_2$) of the ambient $m$ which does not allow ambients of group $g$ to read its context information; only ambients of type $t$ are allowed to do so. Hence the process in Eq. (4) cannot be typed in the proposed type system.

The rule P6 says that if $x$ is a process abstraction name of a given type then the type of the body of that process abstraction must match that of $x$ in the scope of the formal parameters to $x$. A process abstraction definition that meets this requirement is a well-typed process of exchange type any good exchange type and of privacy type $\mathtt{priv}[*,*,*]$. If $P$ and $Q$ are well-typed processes with identical mobility type and exchange type, then the process $P|Q$ is well-typed with the same mobility and exchange type as $P$ and $Q$, but whose privacy type components are the intersection of corresponding components in the privacy types of $P$ and $Q$ (the rule P7).

If $\kappa$ is a well-typed context-expression and $M.P$ a well-typed process, then the *context-guarded prefix* $\kappa?M.P$ is a well-typed process with the same mobility and exchange types as $M.P$. The $\mathcal{R}$-component of the privacy type of $\kappa?M.P$ is the intersection of the disclosed set of $k$ and the $\mathcal{R}$-component of the privacy type of $M.P$; however, its $\mathcal{O}$- and $\mathcal{I}$-components are identical to those of the privacy type of $M.P$ (the rule P8). The rule P9 says that if a context-expression $\kappa$ and a process $M.P$ are well-typed and the disclose set of $\kappa$ coincides with the $\mathcal{R}$-component of the privacy type of $M.P$ and the process $\mathcal{C}(M.P)$ is also well-typed, then the process $\mathcal{C}(\kappa?M.P)$ is of the same type as $\mathcal{C}(M.P)$. A selection process is well-typed if all its branches are well-typed and have the same type (the rule P10). The rule P11 states that a process can inherit any privacy type that is *consistent with* (in the sense of Definition 5.2) its privacy type; and any exchange type that is a *subtype* of its exchange type.

**Example 5.9.** Let $W_1 = \mathtt{amb}(g)[\curvearrowright, \mathbf{1}, \mathtt{priv}[\{h,t\},\{v\},\{u\}]]$ and $W_2 = \mathtt{amb}(h)[\not\curvearrowright, \mathbf{1}, \mathtt{priv}[\{g,t\},\{v\},\{u\}]]$ be two message types. Suppose that $\Gamma \vdash n$: $W_1$ and $\Gamma \vdash m$: $W_2$ are valid type judgements for some type assumption $\Gamma$; ambients $n$ and $m$; and groups $g$, $h$, $t$, $u$, and $v$. The following type judgements are also valid.

- $\Gamma \vdash \mathtt{send}(\{l_1 = n\}).\mathbf{0} \mid \mathtt{recv}(\{l_1 = (y_1 : W_1)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[*,\{v\},\{u\}]]$
- $\Gamma \vdash \mathtt{send}(\{l_1 = n\}).\mathtt{recv}(\{l_1 = (y_1 : W_1), l_2 = (y_2 : W_2)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[*,\{v\},\{u\}]]$
- $\Gamma \vdash m :: \mathtt{send}(\{l_1 = n\}).\mathbf{0} \mid \mathtt{recv}(\{l_1 = (y_1 : W_1)\}).\mathbf{0} : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[\{g,t\},\{v\},\{u\}]]$
- $\Gamma \vdash \mathtt{in}\ m.\mathtt{send}(\{l_1 = n\}).\mathbf{0} : \mathtt{proc}[\curvearrowright, \{l_1 : W_1\}, \mathtt{priv}[*,\{v\},*]]$
- $\Gamma \vdash n[\mathtt{out}.\mathbf{0}] : \mathtt{proc}[\curvearrowright, \mathbf{1}, \mathtt{priv}[*,*,*]]$

## 6. Subject reduction property

We now have to demonstrate that the proposed type system is consistent with the reduction semantics of CCA. This property is commonly known as the *subject reduction* property of a type system (Sangiorgi and Walker, 2001). It states that a well-typed process can only reduce to well-typed processes; therefore process reduction preserves the privacy of the involved ambients.

Theorem 6.1 says that the well-typedness is preserved by reduction. To prove this theorem, Proposition 6.1 is needed and says that the structural congruence relation also preserves well-typedness.

**Proposition 6.1** (Subject Congruence). *If $P \equiv Q$ then*

(a1) *if $\Gamma \vdash P$: $A$ then there exist $g_1, \ldots, g_\ell$ such that $g_1 : \mathtt{gr}, \ldots, g_\ell : \mathtt{gr}, \Gamma \vdash Q : A$.*
(a2) *if $\Gamma \vdash Q$: $A$ then there exist $g_1, \ldots, g_\ell$ such that $g_1 : \mathtt{gr}, \ldots, g_\ell : \mathtt{gr}, \Gamma \vdash P : A$.*

**Proof.** The proof is done by induction on the derivation of $P \equiv Q$. Here are couple of examples; the full proof is detailed in Appendix A.1 in the supplementary material.

- **Proof of the rule S2.** Suppose $P \equiv Q$. Then $Q \equiv P$. For part (a1), assume that $\Gamma \vdash P$: $A$. By induction hypothesis (a2), if $Q \equiv P$ and $\Gamma \vdash P$: $A$ then there exist $g_1, \ldots, g_\ell$ such that $g_1 : \mathtt{gr}, \ldots, g_\ell : \mathtt{gr}, \Gamma \vdash Q : A$.
- **Proof of the rule S12.** For part (a1), suppose $\Gamma \vdash (\nu n : W)m[P] : \mathtt{proc}[X,T,U]$, for $n \neq m$. By rule P3 it implies $\Gamma, n : W \vdash m[P] : \mathtt{proc}[X,T,U]$. By rule P5 we have $\Gamma, n : W \vdash m : \mu Z.\mathtt{amb}(g)[X',T,U']$ and $\Gamma, n : W \vdash P : \mathtt{proc}[X',T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$ and $g \in \mathcal{R} \cap \mathcal{O} \cap \mathcal{I}$. By rule P3, $\Gamma \vdash (\nu n : W)P : \mathtt{proc}[X',T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$. Since $n \neq m$, we also have $\Gamma \vdash m : \mu Z.\mathtt{amb}(g)[X',T,U']$. And by rule P5, $\Gamma \vdash m[(\nu n : W)P] : \mathtt{proc}[X,T,U]$.
  For part (a2), suppose $\Gamma \vdash m[(\nu n : W)P] : \mathtt{proc}[X,T,U]$, for $n \neq m$. By rule P5, this must have been derived from $\Gamma \vdash m : \mu Z.\mathtt{amb}(g)[X',T,U']$ and $\Gamma \vdash (\nu n : W)P : \mathtt{proc}[X',T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$ and $g \in \mathcal{R} \cap \mathcal{O} \cap \mathcal{I}$. By rule P3 it follows that $\Gamma, n : W \vdash P : \mathtt{proc}[X',T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$. By rule E2 it follows that $n \notin \mathtt{dom}(\Gamma)$ and therefore, we have $\Gamma, n : W \vdash m : \mu Z.\mathtt{amb}(g)[X',T,U']$. By rule P5 we have $\Gamma, n : W \vdash m[P] : \mathtt{proc}[X,T,U]$. By rule P3, we conclude that $\Gamma \vdash (\nu n : W)m[P] : \mathtt{proc}[X,T,U]$.
- **Proof of the rule S24.** Suppose $P \equiv Q$ and $\vDash \kappa \Leftrightarrow \kappa'$. Then $M.P \equiv M.Q$, for some capability $M$. For part (a1), assume $\Gamma \vdash \kappa?M.P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$. This must have been derived by rule P8 from $\Gamma \vdash \kappa : \mathtt{cont}[\mathcal{R}_1]$ and $\Gamma \vdash M.P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}_2,\mathcal{O},\mathcal{I}]]$ such that $\mathcal{R} = \mathcal{R}_1 \cap \mathcal{R}_2$. It follows from (K11) that $\Gamma \vdash \kappa' : \mathtt{cont}[\mathcal{R}_1]$. By induction hypothesis (a1), there exist $g_1, \ldots, g_n$ such that $g_1 : \mathtt{gr}, \ldots, g_n : \mathtt{gr}, \Gamma \vdash M.Q : \mathtt{proc}[X,T,U_2]$. This implies that "$g_1 : \mathtt{gr}, \ldots, g_n : \mathtt{gr}, \Gamma$" is a good type invironment and therefore $\{g_1, \ldots, g_n\} \cap \mathtt{dom}(\Gamma) = \emptyset$ (using the rule E3 in Table 10). We can then write $g_1 : \mathtt{gr}, \ldots, g_n : \mathtt{gr}, \Gamma \vdash \kappa' : \mathtt{cont}[\mathcal{R}_1]$. And by rule P8, we conclude that $g_1 : \mathtt{gr}, \ldots, g_n : \mathtt{gr}, \Gamma \vdash \kappa'?M.Q : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R},\mathcal{O},\mathcal{I}]]$. The proof of part (a2) is symmetric.

□

**Theorem 6.1** (Subject Reduction). *If $\Gamma \vdash P$: $A$ and $P \to Q$ then there exist $g_1, \ldots, g_\ell$ such that $g_1 : \mathtt{gr}, \ldots, g_\ell : \mathtt{gr}, \Gamma \vdash Q : A$.*

**Proof.** The proof is done by induction on the derivation of $P \to Q$ as illustrated below (see full proof in Appendix A.2 in the supplementary material. We use the following notational convention: if $U_1 = \mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]$ and $U_2 = \mathtt{priv}[\mathcal{R}_2,\mathcal{O}_2,\mathcal{I}_2]$ then we write $U_1 \cap U_2 = \mathtt{priv}[\mathcal{R}_1 \cap \mathcal{R}_2, \mathcal{O}_1 \cap \mathcal{O}_2, \mathcal{I}_1 \cap \mathcal{I}_2]$.

- **Proof of the rule R1.** Suppose $\Gamma \vdash x \triangleright (y_1, \ldots, y_\ell).P \mid x(z_1, \ldots, z_\ell) : \mathtt{proc}[X,T,U]$. This must have been derived by rule P7 from $\Gamma \vdash x \triangleright (y_1, \ldots, y_\ell).P : \mathtt{proc}[X,T,U_1]$ and $\Gamma \vdash x(z_1, \ldots, z_\ell) : \mathtt{proc}[X,T,U_2]$ such that $U = U_1 \cap U_2$. The former is derived by rule P6, so $U_1 = \mathtt{priv}[*,*,*]$ and $U = U_1 \cap U_2 = U_2$. The latter is derived by rule C8 from $\Gamma \vdash x : \mu Z.\mathtt{abs}(W_1 \times \ldots \times W_\ell)[X,T,U]$ and $\Gamma \vdash z_1 : W_1, \ldots, \Gamma \vdash z_\ell : W_\ell$. Using the rule P6 again, we have $\Gamma, y_1 : W_1, \ldots, y_\ell : W_\ell \vdash P : \mathtt{proc}[X,T,U]$. Hence, we get $\Gamma \vdash P\{z_1/y_1, \ldots, z_\ell/y_\ell\} : \mathtt{proc}[X,T,U]$. Finally by rule P7 we have $\Gamma \vdash x \triangleright (y_1, \ldots, y_\ell).P \mid P\{z_1/y_1, \ldots, z_\ell/y_\ell\} : \mathtt{proc}[X,T,U]$.
- **Proof of the rule R8.** Suppose that $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \mid \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q : \mathtt{proc}[X,T,U]$. By rule P7, $\Gamma \vdash \mathtt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P : \mathtt{proc}[X,T,U_1]$ and $\Gamma \vdash \mathtt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).Q : \mathtt{proc}[X,T,U_2]$ and $U = U_1 \cap U_2$. By rule C4, we must have $T = \{l_1 : W_1, \ldots, l_\ell : W_\ell\}$ and $\Gamma, \{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\} \vdash P : \mathtt{proc}[X,T,\mathtt{priv}[\mathcal{R}_1,\mathcal{O}_1,\mathcal{I}_1]]$ and

**Table 15**
Syntax of CCA with the error process.

| $P$ | ::= | | **Process** |
|---|---|---|---|
| | | ... | the other constructs, as in Table 1 |
| | | wrong | the wrong process |

$U_1 = \texttt{priv}[\mathcal{R}_1, \mathcal{O}_1, \mathcal{I}_1 \cap \mathcal{I}']$, where $\mathcal{I}'$ is the intersection of the $\mathcal{I}$-components of the privacy type of all the $y_i$ that are ambient names; if this intersection is empty then $\mathcal{I}'$ is taken to be '$*$'. By rule C6, $\Gamma \vdash z_1$: $W_1$, ..., $\Gamma \vdash z_\ell$: $W_\ell$ and $\Gamma \vdash Q : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}_2, \mathcal{O}_2, \mathcal{I}_2]]$ and $U_2 = \texttt{priv}[\mathcal{R}_2, \mathcal{O}_2 \cap \mathcal{O}', \mathcal{I}_2]$, where $\mathcal{O}'$ is the intersection of the $\mathcal{O}$-components of the privacy type of all the $z_i$ that are ambient names; if this intersection is empty then $\mathcal{O}'$ is taken to be '$*$'. Hence, we get $\Gamma \vdash P\{z_1/y_1, \ldots, z_\ell/y_\ell\} : \texttt{proc}[X, T, \texttt{priv}[\mathcal{R}_1, \mathcal{O}_1, \mathcal{I}_1]]$. By rule P11 twice, we have $\Gamma \vdash P\{z_1/y_1, \ldots, z_\ell/y_\ell\} : \texttt{proc}[X, T, U_1]$ and $\Gamma \vdash Q : \texttt{proc}[X, T, U_2]$. Finally by rule P7, we have $\Gamma \vdash P\{z_1/y_1, \ldots, z_\ell/y_\ell\} \mid Q : \texttt{proc}[X, T, U]$.

- **Proof of the rule R27.** Suppose that $\mathcal{C}(M.P) \to \mathcal{C}'(P\sigma)$ for some contexts $\mathcal{C}$ and $\mathcal{C}'$. Assume $\Gamma \vdash \mathcal{C}(\kappa?M.P) : \texttt{proc}[X, T, U]$ for some context expression $\kappa$ such that $\mathcal{C} \models \kappa$. By rule P9, we must have $\Gamma \vdash \mathcal{C}(M.P) : \texttt{proc}[X, T, U]$. By induction hypothesis, $\mathcal{C}(M.P) \to \mathcal{C}'(P\sigma)$ implies there exist $g_1, \ldots, g_\ell$ such that $g_1 : \texttt{gr}, \ldots, g_\ell : \texttt{gr}, \Gamma \vdash \mathcal{C}'(P\sigma) : \texttt{proc}[X, T, U]$.

□

A consequence of the subject reduction property is *type soundness*, asserting that under reduction well-typed processes do not give rise to run-time errors, as explained in the following section.

## 7. Type soundness property

In order to define the type soundness property we will extend the syntax of CCA as in Table 15, where the term wrong represents a process in which a run-time error occurs. The structural congruence relation of CCA is extended in Table 16 to define the semantics of processes that can cause run-time errors during reduction. The rule S25 says that any attempt to delete (using the capability del) an object which is not an ambient instance will cause a run-time errors. Same for a process that attempts to move into an object which is not an ambient instance (rule S26), or to exchange a message with such an object (rules S27 and S28), or to refer to such an object as the location of a process abstraction (rule S30) . A run-time error also occurs if an ambient instance is assigned to a variable which is not of an ambient type, or an ambient violates the privacy of another ambient (rule S29). If a name is wrongly used as an abstraction in a process then this process will cause an error during reduction (rules S30, S31 and S32). The proofs of Theorem 7.1 and Corollary 7.1 guarantee that the proposed type system is sound; i.e. no ambient can violate the privacy of another ambient and no run-time error may occur, during the reduction of a well-typed process.

**Theorem 7.1.** *If $\Gamma \vdash P$: $A$ then $P$ does not have* wrong *as a subterm.*

**Proof.** The proof is straightforward by induction on the depth $\gamma$ of the derivation of the type judgement $\Gamma \vdash P$: $A$.

**Base case** ($\gamma = 0$) Theorem 7.1 holds for the process **0**, since this process does not have wrong as a subterm. Suppose $\Gamma \vdash \theta\ x(y_1, \ldots, y_\ell) : \texttt{proc}[X, T, U]$ is a valid. This must have been established using the typing rule C8 (in Table 13). Therefore $x$ is a process abstraction name and hence the process $\theta\ x(y_1, \ldots, y_\ell)$ does not reduce to the process wrong (see rule S31 in Table 16).

**Induction step** Suppose that the lemma holds for all the type judgements of derivation depth less than $\gamma$. We must then prove that the lemma also holds for the type judgements of derivation depth $\gamma$. Let $\Gamma \vdash Q : \texttt{proc}[X, T, U]$ be a type judgement of derivation depth $\gamma$.

- If the process $Q$ is of one of the following forms: in $n.P$, del $n.P$, and $n\,\theta\,P$; then the type judgement $\Gamma \vdash Q : \texttt{proc}[X, T, U]$ must have been derived using the typing rules C1, C3, and C5 (or C7 or C9), repectively. Therefore the process $P$ is well-typed and $n$ is an ambient name. It follows by induction hypothesis that the process $P$ does not have wrong as a subterm. Given that $n$ is an ambient name, we conclude that $Q$ does not have wrong as a subterm (see rules S25-S28 and S30 in Table 16).
- If the process $Q$ is of the form $n[P]$, then the type judgement $\Gamma \vdash Q : \texttt{proc}[X, T, U]$ must have been derived using the typing rule P5, with $U = \texttt{priv}[*, *, *]$. It follows that $\Gamma \vdash P : \texttt{proc}[X', T, \texttt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]]$ and $\Gamma \vdash n : \mu Z.\texttt{amb}(g)[X', T, U']$ and $g \in \mathcal{R} \cap \mathcal{O} \cap \mathcal{I}$. Therefore $n$ is an ambient name and its group $g$ belongs to the $\mathcal{R}$-, $\mathcal{O}$- and $\mathcal{I}$-components of the privacy types of all the ambient names exposed in $P$. By induction hypothesis, it follows that the process $P$ does not have wrong as a subterm. We conclude that $Q$ also does not have wrong as a subterm (see rule S29 in Table 16).
- If the process $Q$ is of the form $x \triangleright (y_1, \ldots, y_\ell).P$, then the type judgement $\Gamma \vdash Q : \texttt{proc}[X, T, U]$ must have been derived using the typing rule P6, with $U = \texttt{priv}[*, *, *]$. This implies that $\Gamma \vdash x : \mu Z.\texttt{abs}(W_1 \times \ldots \times W_\ell)[X, T, U]$ and $\Gamma, y_1 : W_1, \ldots, y_\ell : W_\ell \vdash P : \texttt{proc}[X, T, U]$ are valid type judgements. Therefore, $x$ is an abstraction name. By induction hypothesis, it follows that the process $P$ does not have wrong as a subterm. We conclude that $Q$ also does not have wrong as a subterm (see rule S32 in Table 16).
- For any other form of $Q$, the proof is straightforward using the induction hypothesis and the typing rules in Table 13 and Table 14. □

**Table 16**
Structural congruence relation for the error process.

| | |
|---|---|
| ... | the other congruence rules, as in Table 4. (Let $\theta \in \{\uparrow, \downarrow, ::\}$). |
| **(S25)** $\texttt{del}\ n.P \equiv \texttt{wrong}$ | if $n$ is not an ambient name. |
| **(S26)** $\texttt{in}\ n.P \equiv \texttt{wrong}$ | if $n$ is not an ambient name. |
| **(S27)** $n\ \theta\ \texttt{recv}(\{l_1 = (y_1 : W_1), \ldots, l_\ell = (y_\ell : W_\ell)\}).P \equiv \texttt{wrong}$ | if $n$ is not an ambient name. |
| **(S28)** $n\ \theta\ \texttt{send}(\{l_1 = z_1, \ldots, l_\ell = z_\ell\}).P \equiv \texttt{wrong}$ | if $n$ is not an ambient name. |
| **(S29)** $n[P] \equiv \texttt{wrong}$ | if $n$ is not an ambient name, or the group of the ambient name $n$ does not belong to the $\mathcal{R}$-, $\mathcal{O}$- or $\mathcal{I}$-component of the privacy type of some ambient name exposed in $P$. |
| **(S30)** $n\ \theta\ x(y_1, \ldots, y_\ell) \equiv \texttt{wrong}$ | if $n$ is not an ambient name or $x$ is not a process abstraction name. |
| **(S31)** $\theta\ x(y_1, \ldots, y_\ell) \equiv \texttt{wrong}$ | if $x$ is not a process abstraction name. |
| **(S32)** $x \triangleright (y_1, \ldots, y_\ell).P \equiv \texttt{wrong}$ | if $x$ is not a process abstraction name. |

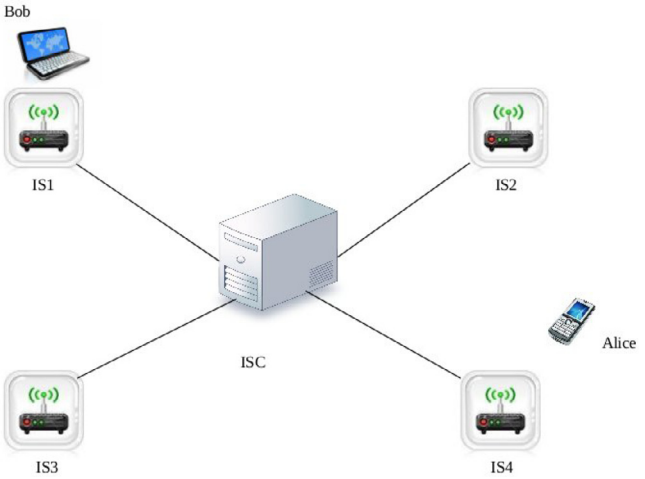



**Fig. 6.** An infostation-based communication system (IMC).

**Corollary 7.1** (Type Soundness). *If* $\Gamma \vdash P$: *A and* $P \to Q$ *then* $Q$ *does not have* `wrong` *as a subterm.*

**Proof.** This follows from Theorem 6.1 and Theorem 7.1, viz:

1. Suppose $\Gamma \vdash P$: *A*.
2. Suppose $P \to Q$.
3. It follows from 1. and Theorem 7.1 that $P$ does not have `wrong` as a subterm.
4. Theorem 6.1 and 1. and 2. imply that there exist $g_1, \ldots, g_\ell$ such that $g_1 : \mathtt{gr}, \ldots, g_\ell : \mathtt{gr}, \Gamma \vdash Q : A$.
5. Therefore, it implies from Theorem 7.1 that $Q$ does not have `wrong` as a subterm. □

We show in the following section how the proposed type system can be used in practice to protect the privacy of the users in a mobile communication system.

## 8. A privacy preserving infostation-based mobile communication system

The infostations paradigm is an infrastructural system concept first proposed by (Frenkiel and Imielinski, 1996) to provide *many-time, many-where* wireless data services. We consider a simple context-aware Infostation-based Mobile Communication (IMC) system which allows users to exchanges *anonymous text messages* in a university campus using mobile devices such as laptops, phones or tablets. The architecture of the system is depicted in Fig. 6 and is made up of one central infostation centre (ISC) which acts as a server and many wireless access points, called *infostations* (ISs), deployed at key locations around the campus.

An infostation is aware of any mobile device within its range and is able to interact with the device. A user within the range of an infostation can send or receive text messages using a mobile device. A text messaging request specifies the text message to be delivered and the identity of the recipient. Once the infostation receives a text messaging request, it checks if the recipient is within range in which case it forwards the text message to the recipient. If the recipient is not within the range of the infostation, the infostation forwards the request to the infostation centre.

The infostation centre is aware of all the infostations connected to it and can communicate with them. When the infostation centre receives a text messaging request from an infostation, it looks for the infostation where the recipient is located and forwards the request to this infostation which then forwards the text message to the recipient mobile device. The privacy requirement of the text messaging service is that the identity and the location of the sender must not be disclosed to the recipient.

### 8.1. Modelling the IMC system in typed CCA

We model each component of this system as an ambient. For the sake of simplicity, we name the user mobile device after the user; e.g. *Bob* and *Alice*. Also, we say that a mobile device is in the range of an infostation if the ambient modelling that mobile device is a child ambient of that infostation. In this case the location of the user is taken to be that infostation. It is assumed that the ranges of the infostations do not overlap. So the overall IMC system depicted in Fig. 6 can be modelled as a composition of concurrent ambients like in Eq. (5):

$$Sys = ISC[P_{isc}] \mid IS1[P_{is} \mid Bob[P_{usr}]] \mid IS2[P_{is}] \mid IS3[P_{is}] \mid IS4[P_{is} \mid Alice[P_{usr}]], \tag{5}$$

where the process $P_x$ describes the behaviour of the ambient that contains it. Note that all the infostations exhibit the same behaviour described by the process $P_{is}$ (see Eq. (12)), while the user mobile devices provide the same services described by the process $P_{usr}$ (see Eq. (9)). We now have to assign a type to each of these ambients. To do so, we define the following groups:

- $g_{txt}$: text messages belong to this group
- $g_{usr}$: users' devices belong to this group
- $g_{is}$: infostations belong to this group
- $g_{isc}$: infostation centres belong to this group.

A text message can be sent and received by any of the devices involved in the system. So the type of a text message can be defined as

$$W_1 = \mathtt{amb}(g_{txt})[\not\curvearrowright, \mathtt{shh}, \mathtt{priv}[*, *, *]],$$

i.e. the type of an ambient that is immobile, does not communicate, and is happy to share its context with anyone. A user mobile device can send and receive text messages, but cannot learn the sender's identity nor the sender's location; the location of a user being the infostation in range with that user mobile device. To prevent a user mobile device from accessing the identity and the location of a sender, the $\mathcal{R}$-component and the $\mathcal{I}$-component of a user mobile device privacy type must not contain the group $g_{usr}$. Thus the type of a user mobile device is the recursive type

$$W_2 = \mu Z.\mathtt{amb}(g_{usr})[\curvearrowright, \{l_1 : W_1, l_2 : Z\}, \mathtt{priv}[\{g_{is}, g_{isc}\}, \{g_{is}, g_{usr}, g_{isc}\}, \{g_{is}, g_{isc}\}]],$$

where the field $l_1$ represents the text message being exchanged and $l_2$ denotes the identity of the recipient mobile device. The privacy type of the type $W_2$ states that, ambients of the group $g_{usr}$ cannot receive nor read a name of the type $W_2$, but can output them (in the sense defined in Section 5.1, page 10).

Unlike user mobile devices, infostations are immobile. However, they have the same exchange type as user mobile devices in order to be able to communicate with them. The infostation centre is aware of their context and can share the context of each of them with the others. So the type of an infostation is

$$W_3 = \mathtt{amb}(g_{is})[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[\{g_{isc}\}, \{g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]].$$

The infostation centre is also immobile, of the same exchange type and allows solely infostations to access its context. Its type is hence defined as

$$W_4 = \mathtt{amb}(g_{isc})[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[\{g_{is}\}, \{g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]].$$

We now specify the behaviour of each of these devices. When in the range of an infostation, a user mobile device is able to:

- create and send a text messaging request containing a message (denoted here by $m$) and the identity of the recipient e.g. *Alice*, i.e.

$$(\nu\ m : W_1)\ \ \uparrow \texttt{send}(\{l_1 = m, l_2 = Alice\}).\mathbf{0} \tag{6}$$

- receive a text message, i.e.

$$!\uparrow \texttt{recv}(\{l_1 = (msg : W_1)\}).\mathbf{0} \tag{7}$$

- move with its user around the university campus, i.e.

$$\texttt{out.in}\ IS1.\mathbf{0} \mid \texttt{out.in}\ IS2.\mathbf{0} \mid \texttt{out.in}\ IS3.\mathbf{0} \mid \texttt{out.in}\ IS4.\mathbf{0} \tag{8}$$

So the behaviour of a user mobile device is the parallel composition of Eq. (6), Eq. (7) and Eq. (8), i.e.

$$P_{usr} = \text{Eq. (6)} \mid \text{Eq. (7)} \mid \text{Eq. (8)}. \tag{9}$$

Once an infostation receives a text messaging request from a user mobile device, it sends the message to the recipient if the latter is in its range; otherwise it forwards the request to the infostation centre. This is specified as follows, where $\texttt{has}(n)$ means that the ambient $n$ is located at the current ambient and is formally defined in Example 3.2 (on page 8):

$$\left(\begin{array}{l} !\downarrow \texttt{recv}(\{l_1 = (msg : W_1), l_2 = (to : W_2)\}).\texttt{if} \\ \quad (\texttt{has}(to))?to\downarrow \texttt{send}(\{l_1 = msg\}).\mathbf{0} \\ \quad (\neg\texttt{has}(to))?ISC :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \texttt{fi} \end{array}\right) \tag{10}$$

Conversely, an infostation can also receive a request from the infostation centre to forward a text message to a user mobile device which is in its range. This is specified as follows:

$$\left(\begin{array}{l} !ISC :: \texttt{recv}(\{l_1 = (msg : W_1), l_2 = (to : W_2)\}).\texttt{if} \\ \quad (\texttt{has}(to))?to\downarrow \texttt{send}(\{l_1 = msg\}).\mathbf{0} \\ \quad (\neg\texttt{has}(to))?ISC :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \texttt{fi} \end{array}\right) \tag{11}$$

The overall behaviour of an infostation is then modelled by the process in Eq. (12).

$$P_{is} = \text{Eq. (10)} \mid \text{Eq. (11)}. \tag{12}$$

As for the infostation centre, it receives a text messaging request from one of the infostations and forwards the request to the infostation in range with the recipient of that text messaging request. If no infostation is in range with the recipient, the request is saved in the infostation centre and will be released when the recipient eventually joins the network. This is specified in Eq. (13), where $\texttt{at2}(n, m)$ means that the ambient $m$ is located at the ambient $n$ and is formally defined in Example 3.3 (on page 9).

$$P_{isc} = \left(\begin{array}{l} !:: \texttt{recv}(\{l_1 = (msg : W_1), l_2 = (to : W_2)\}).\texttt{if} \\ \quad (\texttt{at2}(IS1, to))?IS1 :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \quad (\texttt{at2}(IS2, to))?IS2 :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \quad (\texttt{at2}(IS3, to))?IS3 :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \quad (\texttt{at2}(IS4, to))?IS4 :: \texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} \\ \texttt{fi} \end{array}\right) \tag{13}$$

### 8.2. Type-checking of the IMC system

The following theorem asserts that the IMC system's model *Sys* of Eq. (5) is well-typed. This ensures that the privacy (as specified by the types) of no ambient can be violated during the execution of the system. In particular, the recipient of a message is unable to determine the identity of the sender, nor the sender's location.

**Theorem 8.1.** *Let* $\Gamma = \{Bob : W_2, Alice : W_2, IS1 : W_3, IS2 : W_3, IS3 : W_3, IS4 : W_3, ISC : W_4\}$. *Then*

$$\Gamma \vdash Sys : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]].$$

**Proof.**

1. Using the rules P1, C6, P3 and P2, we have
$\Gamma \vdash \text{Eq. (6)} : \texttt{proc}[\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, \{g_{usr}, g_{is}, g_{isc}\}, *]]$.
2. Similarly, from the rules P1, C4, P2 and P11, we have
$\Gamma \vdash \text{Eq. (7)} : \texttt{proc}[\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$.
3. Using the rules P1, C1, C2 and P7, we have
$\Gamma \vdash \text{Eq. (8)} : \texttt{proc}[\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$.
4. From 1., 2. and 3. and the rule P7, we have
$\Gamma \vdash P_{usr} : \texttt{proc}[\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, \{g_{usr}, g_{is}, g_{isc}\}, *]]$.
5. Using the rule P5, we have
$\Gamma \vdash Bob[P_{usr}] : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$, and
$\Gamma \vdash Alice[P_{usr}] : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$.
6. Using the rules P1, C6 and C7, we have
$\Gamma, msg : W_1, to : W_2 \vdash to\downarrow \texttt{send}(\{l_1 = msg\}).\mathbf{0} :$
$\texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{is}, g_{isc}\}, *, *]]$.
7. Recall that the context-expression $\texttt{has}(to)$ is defined in Example 3.2 as $\texttt{has}(to) = \diamond(\bullet \mid to[\texttt{true}] \mid \texttt{true})$. Therefore, it follows from the rules K1, K2, K6, K9 and K10 that
$\Gamma, msg : W_1, to : W_2 \vdash \texttt{has}(to) : \texttt{cont}[\{g_{is}, g_{isc}\}]$.
8. From 6. and 7. and the rule P8, we have
$\Gamma, msg : W_1, to : W_2 \vdash (\texttt{has}(to))?to\downarrow \texttt{send}(\{l_1 = msg\}).\mathbf{0} :$
$\texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{is}, g_{isc}\}, *, *]]$.
9. Similarly to 6.-8., we have
$\Gamma, msg : W_1, to : W_2 \vdash (\neg\texttt{has}(to))?ISC ::$
$\texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} :$
$\texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{is}\}, \{g_{usr}, g_{is}, g_{isc}\}, *]]$.
10. From 8., 9. and the rules P10, C4, P2 and P11; we have
$\Gamma \vdash \text{Eq. (10)} : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\},$
$\texttt{priv}[\{g_{is}\}, \{g_{usr}, g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]]$.
11. From 8., 9. and the rules P10, C5, P2 and P11; we have
$\Gamma \vdash \text{Eq. (11)} : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{is}\},$
$\{g_{usr}, g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]]$.
12. From 10., 11. and the rule P7, we have
$\Gamma \vdash P_{is} : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{is}\},$
$\{g_{usr}, g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]]$.
13. From 5., 12. and the rules P7 and P5,
$\Gamma \vdash IS1[P_{is} | Bob[P_{usr}]] : \texttt{proc}[\not\curvearrowright,$
$\{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$, and
$\Gamma \vdash IS2[P_{is}] : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$, and
$\Gamma \vdash IS3[P_{is}] : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[*, *, *]]$, and
$\Gamma \vdash IS4[P_{is} \mid Alice[P_{usr}]] : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1,$
$l_2 : W_2\}, \texttt{priv}[*, *, *]]$.
14. Similarly to 9., and for $k = 1, 2, 3, 4$, we have
$\Gamma, msg : W_1, to : W_2 \vdash (\texttt{at2}(IS_k, to))?IS_k ::$
$\texttt{send}(\{l_1 = msg, l_2 = to\}).\mathbf{0} :$
$\texttt{proc}[\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{isc}\}, \{g_{usr}, g_{is}, g_{isc}\}, *]]$.
15. From 14. and the rules P10, C4 and P2, we have
$\Gamma \vdash P_{isc} : \texttt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \texttt{priv}[\{g_{isc}\},$
$\{g_{usr}, g_{is}, g_{isc}\}, \{g_{is}, g_{isc}\}]]$.

```
 1. CCA Parser Version 4.04:  Reading from file privacy3.cca . . .
 2. CCA Parser Version 4.04:  CCA program parsed successfully.

 3. Execution mode: random


 4. --->  {ambient "Bob" moves out of ambient "IS1"}
 5. --->  {ambient "Alice" moves out of ambient "IS4"}
 6. --->  {ambient "Bob" moves into ambient "IS2"}
 7. --->  {Child to parent: Bob ===(hello,Alice)===> IS2}
 8. --->  {ambient "Bob" moves out of ambient "IS2"}
 9. --->  {Sibling to sibling: IS2 ===(hello,Alice)===> ISC}
10. --->  {ambient "Bob" moves into ambient "IS4"}
11. --->  {ambient "Bob" moves out of ambient "IS4"}
12. --->  {ambient "Bob" moves into ambient "IS1"}
13. --->  {ambient "Bob" moves out of ambient "IS1"}
14. --->  {ambient "Bob" moves into ambient "IS3"}
15. --->  {ambient "Alice" moves into ambient "IS3"}
16. --->  {ambient "Alice" moves out of ambient "IS3"}
17. --->  {ambient "Alice" moves into ambient "IS1"}
18. --->  {Sibling to sibling: ISC ===(hello,Alice)===> IS1}
19. --->  {Parent to child: IS1 ===(hello)===> Alice}
20. --->  {ambient "Alice" moves out of ambient "IS1"}
21. --->  {ambient "Alice" moves into ambient "IS2"}
22. --->  {ambient "Alice" moves out of ambient "IS2"}
23. --->  {ambient "Alice" moves into ambient "IS4"}
```

**Fig. 7.** A simulation output of the IMC system.

16. From 15. and the rule P5,

$$\Gamma \vdash ISC[P_{isc}] : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[*, *, *]].$$

17. Finally, from 13., 16. and the rule P7, we conclude that

$$\Gamma \vdash Sys : \mathtt{proc}[\not\curvearrowright, \{l_1 : W_1, l_2 : W_2\}, \mathtt{priv}[*, *, *]].$$

□

### 8.3. Simulation of the IMC system

Theorem 8.1 guarantees that the IMC system's model in Eq. (5) fulfils the privacy requirements specified by the types. However this does not guarantee that the system reliably enables two users to actually exchange messages. To check this, the IMC system's model is randomly simulated in `ccaPL` (i.e. the `CCA` interpreter) on a HP ProBook 6470b laptop computer with 8GB of RAM and a quad-core Intel processor (@2.60GHZ × 4). `ccaPL` implements the reduction semantics of `CCA` described in Section 4. The model of execution is interleaving; and at each execution step one process is chosen non-deterministically from the pool of enabled processes.

The simulation results of more than 100 runs show that any message sent by a user is eventually received by the recipient. The typical simulation output is depicted in Fig. 7 for a scenario involving two users Alice and Bob wandering around a university campus covered by the IMC system. Initially, Bob is in the range of the infostation IS1 and Alice is located in the zone covered by the infoastation IS4 (see Eq. (5)). In the execution output, the symbol "`--->`" represents the reduction relation defined in Section 4; and the explanation of each reduction is given between a pair of curly brackets. For example the reduction in line 4 lets the ambient Bob move out of the ambient IS1; while the reduction in line 7 says that a child ambient Bob sends the message `(hello,Alice)` to its parent ambient IS2. The remaining reductions can be explained in a similar manner. From line 4 to line 7, Bob moves from IS1 to IS4 while Alice moves out of IS4 (and into IS3 later in line 15); using the mobility capabilities specified in Eq. (8). At IS2 (line 7) Bob sends the message `(hello,Alice)` to its current parent ambient (see Eq. (6)) which happens to be IS2; then moves from IS2 to IS3 via IS4 and IS1 respectively (lines 8, 10–14). During that time, Alice moves from IS4 to IS1 through IS3 (line 5, lines 15–17), forcing IS2 to forward the message to ISC (line 9), which then forwards the message to IS1 where Alice (the recipient) is located (line 18). The message `(hello)` finally reaches Alice in IS1 (line 19). The message's routes from the sender to the recipient may differ between runs due to the non-determinism of the random simulation.

The simulator `ccaPL` also displays the execution output graphically in the form of a diagram, which enables the user to visualise the concurrent behaviours of the system. Two types of diagrams can be produced: (i) a communication diagram depicting message passing between ambients; and (ii) a behaviour diagram showing both the movements of ambients and the communications between them. The diagrams corresponding to the execution output in Fig. 7 are depicted in Fig. 8 for the communication diagram and Fig. 9 for the behaviour diagram. The top row in both diagrams lists the names of the ambients involved in the system being executed. The vertical dashed line represents the timeline for each ambients (time increases from top to bottom); and a solid directed line from the timeline of an ambient A say to that of an ambient B indicates a message passing from the ambient A to the ambient B at a specific time point, with the content of the message carried as a label to that line. Moreover, a behaviour diagram provides additional information about the mobility of ambients. A box labelled as "X `-->` Y" on the timeline of an ambient A indicates that the ambient A has moved from location X to location Y. Unlike the textual execution output like in Fig. 7, a behaviour diagram shows clearly the parallelism among of the involved ambients; e.g. in Fig. 9 the ambients Bob and Alice move freely while the communications between the infostations are taking place.

## 9. Related work

A type system is a powerful technique to check statically that a system cannot violate certain properties at run-time. For instance, (Milner, 1999; Sangiorgi and Walker, 2001) proposed type systems for ensuring that communications over channels are safe in the pi-calculus and related languages. Type systems for bigraphs were studied in (Elsborg et al., 2008) as an modular approach to devise type systems before transferring them to the family of calculi that

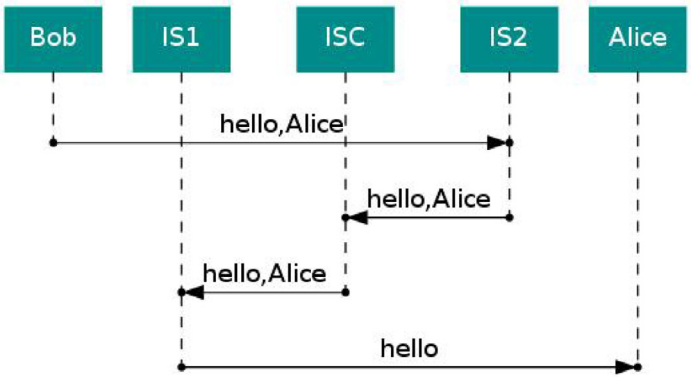



**Fig. 8.** A communication diagram of the IMC system.

the bigraphs model such as the pi-calculus. The mobility types for mobile ambients were introduced in (Cardelli et al., 1999), and extended with the concept of group in (Cardelli et al., 2000). The concept of group was also used in (Coppo et al., 2003) with a type system to control the mobility of ambients. Unlike these works, in addition to explicit flow of information through communication primitives our type system is able to control effectively the implicit flow of context information generated by context-awareness, e.g. through context-expressions.

In (Hennessy and Riely, 2002), a type system is proposed for resource access control in a distributed pi-calculus. That type system is based on a notion of location type, which describes the set of resources available to an agent at a location; where a resource is essentially a communication channel. Similar work was done in (Nicola et al., 1998) to control access to tuples spaces in a Linda (Carriero et al., 1995) like language. Likewise, (Chaudhuri and Abadi, 2006) proposed secrecy types to control access to a file system in a variant of the pi-calculus; a secrecy type being assigned a set of the clients that any information of that type can reach. The latter is the closest to our work and their notion of secrecy type can be expressed in our privacy type as $\mathtt{priv}[G, G, G]$ where $G$ is the set of the clients that any information of that type can reach.

Attribute-based access control (ABAC) and variants of role-based access control (RBAC) models and languages have been proposed in (OASIS, 2010; Crampton and Morisset, 2012; Covington et al., 2001; 2002) for the specification of access control in open systems. (Choi et al., 2008) proposed an adaptive access control scheme which extend RBAC to enable dynamic user and permission assignment based on changes in context. They proposed algorithms for adaptive role assignment, delegation and revocation. Usage control (UCON) model (Park and Sandhu, 2004) enhances the traditional access control models to enable mutability of subject and object attributes, and continuity of control on usage of resources. It is shown in (Park and Sandhu, 2004) that previous access control models such as RBAC and Digital Rights Management (DRM) can be represented in UCON. Further extension of UCON (Almutairi and Siewe, 2011; 2013) enables adaptation when access is denied or stopped due to changes in the environmental context. In these works security is addressed separately, generally as an add-on after the system is developed. These works use context information to control access to data and services, but do not control the flow of this context information within the system. Our type system controls both the access (*input* or *output*) to data and services, and the flow of context information generated (*read*) through sensing.

Recently, researchers have been interested in developing a fine-grained access control model for the Android platform (Bai et al., 2010; Conti et al., 2012) based on the UCON model. This is motivated by the fact that current Android permission model does not allow user to revoke nor change the permissions of an application once the application is installed; users can neither specify context-dependent policies to grant permissions to applications only in specific situations. (Bai et al., 2010) proposed a context-aware usage control (ConUCON) for Android that enables the specification of spatial and temporal context information in policies to control how permissions are granted to applications based on time and location. They then extend the existing security mechanism of Android to support ConUCON and enable revocation and modification of an application's permissions at run-time. They later applied ConUCON to enhance security of services in Web of Things (WoT) Bai et al.. Likewise, (Conti et al., 2012) considered a subset of the UCON model and modified the Android implementation to enable the enforcement of context-dependent policies. Their system, CRêPE, enables the users to specify own policies to regulate the behaviour of the applications installed on their smart devices. These systems used context information to control access to behaviours of applications but do not control how context information flows within the system, which is the main focus of our approach.

Bucur and Nielsen (2008) proposed a Mobile Ambients based process calculus to describe context-aware computing in an infrastructure-based ubicomp system. In their calculus they model contextual information as macro definitions residing at ambients like in (Zimmer, 2005), but enable these macros to be defined and called across multiple ambients's boundaries. They used macros to model context provision and discovery. In our case macros are called process abstractions and access to them is limited to the vicinity of the ambient where they are defined, namely the siblings, children and the parent of that ambient. This design choice was taken to enable local context provision through process abstractions while enabling system wise context discovery through context-expressions using spatial modalities such as somewhere $\Diamond$. Context-expressions as defined in Table 1 are much more expressive than macros. They proposed a type system that assigns each ambient an entry policy and a security policy. The entry policy of an agent $A$ is a set of agent names that are allowed to enter the agent $A$. As for the security policy, it regulates the provision and discovery of contextual information over "a distributed flows of actions". In our work, context provision through output and input capability are controlled by the $\mathcal{O}$- and $\mathcal{I}$-components of ambients' privacy types; while context discovery through context-expressions is controlled by the $\mathcal{R}$-components of ambients' privacy types. In order to enforce their type system, they have combined both static and dynamic type checking. The use of groups and not ambient names in our type system makes it possible to check it statically; which is much more efficient.

Kouzapas and Philippou (2015) devised a formal framework for studying privacy based on the $\pi$-calculus. They use the concept of group to control leakage of information: a name cannot exist

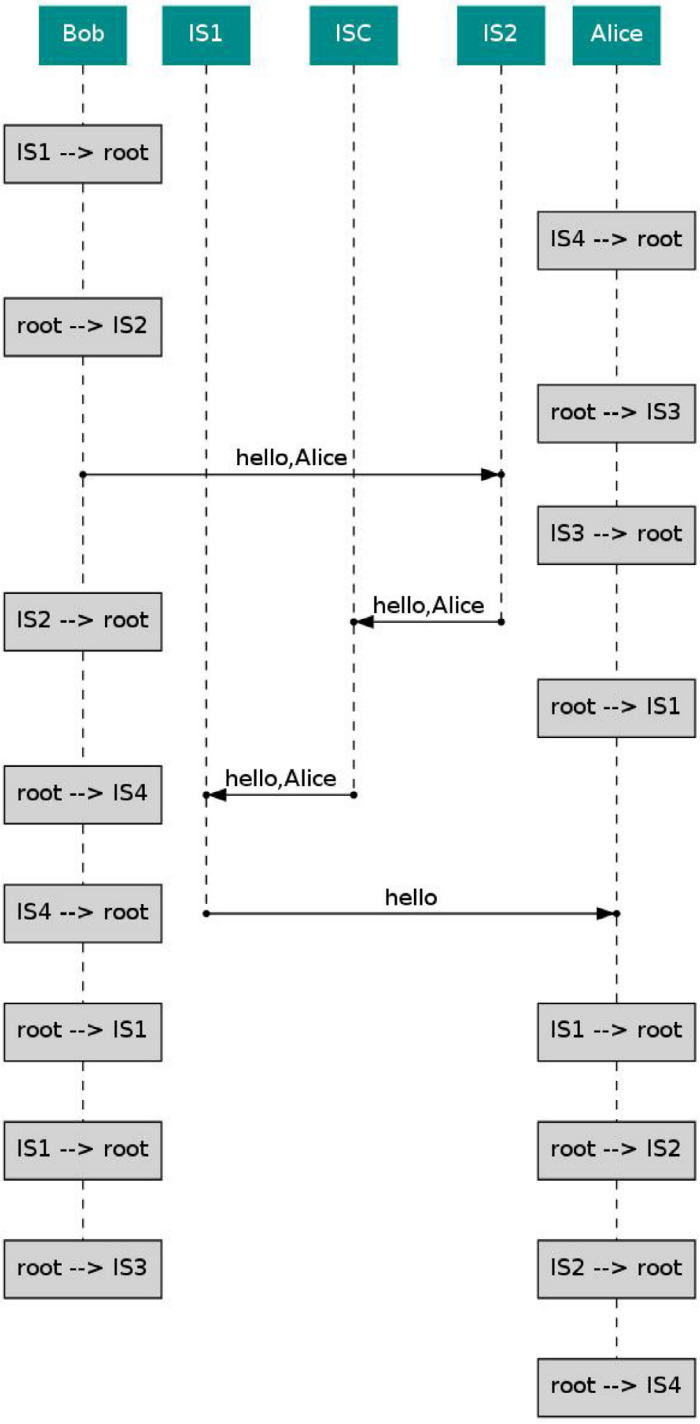



**Fig. 9.** A behaviour diagram of the IMC system.

outside the scope of its group. This is also valid for our work as an ambient's group is embedded in the type of that ambient and therefore that ambient can only exist in the scope of its group, e.g. in the process $\nu\ g : \mathtt{gr}\ \nu\ n : \mathtt{amb}(g)[\curvearrowright, \mathbf{1}, \mathtt{priv}[*,*,*]]\ P$. A type assignment to a channel $c$ has the form $c$: $G[T]^{p\lambda}$, where $G$ is the group of $c$ belongs to, $T$ is the type of values that can be exchanged on the channel, $p \in \{_, \mathtt{r}, \mathtt{w}, \mathtt{rw}\}$ indicates how the channel is used (input/output), and $\lambda$ the number of times it may be disseminated (linearity). In our type system, there is no distinction between input and output exchanges; thus $\mathtt{rw}$ is assumed for $p$. As for type linearity, this can be specified using the privacy types. For example, linearity 0 corresponds to $\mathtt{priv}[*, \emptyset, \emptyset]$ i.e. a name of this privacy type cannot be disseminated; linearity 1 corresponds to $\mathtt{priv}[*, \{g\}, \{g\}]$ i.e. a name of this privacy type can be disseminated to a single group $g$; and linearity $\infty$ corresponds to $\mathtt{priv}[*,*,*]$ i.e. no dissemination limit. However, there is no feature in their language to represent context information, which is a vital aspect of ubicomp systems.

Pasqualin et al. defined type system for CCA with focus on the communication between processes; extending existing works on the type systems for mobile computing (Cardelli et al., 1999). Their type system does not control the flow of context information and the type of a context-expression is simply the Boolean type, i.e. a well-typed context-expression has the form $\kappa$: *Bool*. More-

over, no mechanism for protecting privacy is presented in their work. In our approach, types are used to collect context information contained in a context-expression and subsequently check that this context information is disclosed accordingly to the privacy requirements of involved ambients; privacy requirements are specified using privacy types. Thus a well-typed context-expression has the form $\kappa : \mathtt{cont}[\mathcal{R}]$, where $\mathcal{R}$ is the disclosure set of $k$ (see Definition 5.3).

## 10. Conclusion

This paper proposed a novel type system for protecting privacy in ubicomp systems. The type system is based on CCA and comprises mobility types, exchange types and privacy types. While the mobility types control the movement of ambients within a system, exchange types specify the types of the values exchanged by each ambient. For example, an ambient of mobility type $\not\curvearrowright$ (i.e. immobile) must not have any mobility capability. Similarly, an ambient of exchange type $T$ must not input nor output values of type other than $T$ or a subtype of $T$. While mobility types and exchange types are widely studied in concurrent and mobile computing (Cardelli et al., 1999; Merro and Sassone, 2002; Pierce and Sangiorgi, 1996; Milner, 1999), privacy types for ubicomp are still under development.

As much as mobility and concurrency, context-awareness and calm adaptation make it difficult to protect user privacy in ubicomp. The privacy type proposed in this paper uses the concept of group to tag the different relevant components of a system that are able to sense or exchange context information. Each of these components is modelled as an ambient of a given type $\mathtt{amb}(g)[X, T, \mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]]$ that indicates that ambient's group $g$, its mobility type $X$, its exchange type $T$ and its privacy type $\mathtt{priv}[\mathcal{R}, \mathcal{O}, \mathcal{I}]$, where $\mathcal{R}$, $\mathcal{O}$ and $\mathcal{I}$ are sets of groups such that:

- $\mathcal{R}$ is the set of the groups of the ambients that are allowed to read or sense the context of an ambient of this privacy type;
- $\mathcal{O}$ is the set of the groups of the ambients that are allowed to output any context information of an ambient of this privacy type;
- $\mathcal{I}$ is the set of the groups of the ambients that are allowed to input any context information of an ambient of this privacy type.

Thus an ambient specifies in its privacy type who can read its context, who can share its context with a third party, and who that third party might be. In this way the flow of context information can be controlled statically.

The typing rules of the type system were defined; only terms that can be typed using these rules are well-typed. The subject reduction property of the proposed type system was formally established with respect to the reduction semantics of typed CCA. This guarantees that well-typedness is preserved by reduction. The soundness property of the type system was also established formally to guarantee that well-typed processes are free from runtime errors and cannot accidentally disclose private information. A case study of an infostation-based mobile communication system is used to illustrate how the proposed type system can be used in practice to protect the privacy of the users. This system is fully specified in typed CCA and its well-typedness formally established.

In future work, we will investigate a dynamic type system for CCA and bisimulation relations to reason about the behaviours of the Internet of Things (IoT) systems. In particular, it should be possible for an ambient to be assigned more than one group to cater for the versatility of interactions in IoT systems.

## Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.jss.2016.07.037

## References


http://www.emarketer.com/corporate/coverage#/results/1284 (accessed: 09-08-2015). 2015.

Al-Doori, M., Siewe, F., Al-Bayatti, A.. Routing management for dtn networks in vanet. Int. J. Mach, Lear. Comput. 1(5).

Al-Sammarraie, M., Siewe, F., Zedan, H., 2010. Formalising policies of a mlearning system using cca. In: 4th ACM Workshop on Context-awareness for Self-managing Systems. ACM, Copehagen, Denmark, pp. 11–19.

Almutairi, A., Siewe, F., 2011. Ca-ucon: A context-aware usage control model. In: Proceedings of the 5th ACM International Workshop on Context-Awareness for Self-Managing Systems, CASEMANS '11. ACM, New York, NY, USA, pp. 38–43.

Almutairi, A., Siewe, F., 2013. Formal specification of ca-ucon model using cca. In: Science and Information Conference, pp. 369–375.

Bai, G., Gu, L., Feng, T., Guo, Y., Chen, X., 2010. Context-aware usage control for android. In: Jajodia, S., Zhou, J. (Eds.), Security and Privacy in Communication Networks, Vol. 50 of Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, pp. 326–343.

Bai, G., Yan, L., Gu, L., Guo, Y., Chen, X.,. Context-aware usage control for web of things. In: Security and Communication Networks.

Bucur, D., Nielsen, M., 2008. Secure Data Flow in a Calculus for Context Awareness. In: Concurrency, Graphs and Models, Vol. 5065 of Lecture Notes in Computer Science. Springer, pp. 439–456.

Cardelli, L., Ghelli, G., Gordon, A., 1999. Mobility types for mobile ambients. In: 26th International Colloquium on Automata, Languages and Programming (ICALP'99). LNCS, Springer, pp. 230–239.

Cardelli, L., Ghelli, G., Gordon, A., 2000. Ambient groups and mobility types. Springer, pp. 333–347.

Cardelli, L., Gordon, A.D., 2000. Mobile ambients. Theoretical Computer Science 240, 177–213.

Carriero, N., Gelernter, D., Zuck, L., 1995. Bauhaus linda. In: Object-Based Models and Languages for Concurrent Systems, Vol. 924 of LNCS. Springer-Verlag, Berlin, pp. 66–76.

Chaudhuri, A., Abadi, M., 2006. Secrecy by typing and file-access control. In: 19th IEEE Computer Security Foundations Worshop (CSFW'06). IEEE Computer Society, pp. 112–123.

Choi, J.H., Khang, S.D.H., Jang, H., Eom, Y.I., 2008. Adaptive access control scheme utilizing context awareness in pervasive computing environments. In: IEEE International Performance, Computing and Communications Conference. IPCCC, pp. 491–498. IEEE

Conti, M., Crispo, B., Fernandes, E., Zhauniarovich, Y., 2012. CRêPE: A system for enforcing fine-grained context-related policies on android. IEEE Trans. Inf. Forensics Secur. 7 (5), 1426–1438.

Coppo, M., Dezani-Ciancaglini, M., Giovannetti, E., Salvo, I., 2003. $M^3$: Mobility types for mobile processes in mobile ambients. In: Computing: The Australasian Theory Symposium (CATS'03), Vol. 78 of Electronic Notes in Theoretical Computer Science, pp. 144–177.

Covington, M.J., Fogla, P., Zhan, Z., Ahamad, M., 2002. A context-aware security architecture for emerging applications. In: Proceedings of the 18th Annual Conference on Computer Security Applications, pp. 249–258.

Covington, M.J., Long, W., Srinivasan, S., Dev, A.K., Ahamad, M., Abowd, G.D., 2001. Securing context-aware applications using environment roles. In: Proceedings of the Sixth ACM Symposium on Access Control Models and Technologies, SACMAT '01. ACM, New York, NY, USA, pp. 10–20.

Crampton, J., Morisset, C., 2012. Ptacl: A language for attribute-based access control in open systems. In: Proceedings of the First International Conference on Principles of Security and Trust, POST'12. Springer-Verlag, Berlin, Heidelberg, pp. 390–409.

Elsborg, E., Hildebrandt, T., Sangiorgi, D., 2008. Type systems for bigraphs. Tech. rep. tr-2008-110. The IT University of Copenhagen.

Frenkiel, R., Imielinski, T., 1996. Infostations: The joy of many-time, many-where communications. Tech. rep.. WINLAB.

Hennessy, M., Riely, J., 2002. Resource access control in systems of mobile agents. Inf. Comput. 173, 82–120.

Hong, J.I., Ng, J.D., Lederer, S., Landsay, J.A., 2004. Privacy risk models for designing privacy-sensitive ubiquitous computing systems. In: In Designing Interactive Systems (DIS2004). ACM Press, pp. 91–100.

Kouzapas, D., Philippou, A., 2015. Type checking privacy policies in the $\pi$-calculus. In: Formal Techniques for Distributed Objects, Components, and Systems, Vol. 9039 of Lecture Notes in Computer Science, pp. 181–195.

Merro, M., Sassone, V., 2002. Typing and subtyping mobility in boxed ambients. In: Concurrency Theory (CONCUR), Vol. 2421 of LNCS. Springer-Verlag, pp. 304–320.

Milner, R., 1999. Communication and Mobile Systems: The $\pi$-Calculus. Cambridge University Press.

Nicola, R.D., Ferrari, G., Pugliese, R., 1998. Klaim: A kernel language for agents interaction and mobility. IEEE Trans. Software Eng. 24 (5), 315–330.

OASIS, 2010. Extensible access control markup language (xacml) version 3.0.

Park, J., Sandhu, R., 2004. The $\mathrm{UCON}_{\mathrm{ABC}}$ Usage Control Model. ACM Trans. on Inf. and Syst. Security 7 (1), 128–174.

Pasqualin, D. P., Vizzotto, J. K., Piveta, E. K.,. Typed context awareness ambient calculus for pervasive applications. Formal Aspects of Computing 27 (5–6).
Pierce, B., Sangiorgi, D., 1996. Typing and subtyping for mobile processes. In: Mathematical structures in computer science, pp. 376–385.
Sangiorgi, D., Walker, D., 2001. The $\pi$-calculus: A Theory of Mobile Processes. Cambridge University Press.
Schilit, B., Hong, J., Gruteser, M., 2003. Wireless location privacy protection. Computer 36 (12), 135–137.
Siewe, F., Zedan, H., Cau, A., 2011. The calculus of context-aware ambients. J. Comput. Sys. Sci. 77 (4), 597–620.
Weiser, M., 1991. The computer for the 21st century. Scientific American 265 (3), 94–104.
Zimmer, P., 2005. A Calculus for Context-awareness. Tech. rep.. BRICS.