Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 180 (2007) 3–15

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Types for Access Control in a Calculus of Mobile Resources

Hans Hüttel[1]

Department of Computer Science, Aalborg University
Fredrik Bajersvej 7E, 9220 Aalborg Ø, Denmark

Morten Kühnrich[2]

Department of Computer Science, University of Copenhagen
Universitetsparken 1, 2100 København, Denmark

Abstract

This paper presents a type system for the calculus of Mobile Resources (MR) proposed by Godskesen et al. The type system is able to prevent undesirable border-crossing behaviour such as Trojan horses. This is achieved by combining the notion of group with a notion of security policy. Well-typed processes satisfy a safety property which is preserved under reduction. An algorithm is presented which computes the minimal security policy making a process well typed.

Keywords: Process calculus, mobility, types, access control, type inference.

1 Introduction

Mobility of software across computer networks is now a widespread phenomenon. Examples range from Java applets moving from servers to clients to the physical movement of a computer between wireless networks. As a result, ensuring the security of such migration has become an important issues.

A number of process calculi based on the π-calculus have been developed to address this and related issues. Examples include the calculus of Mobile Ambients [2], the Seal calculus [8], the Kell calculus [7] and the calculus of mobile resources [3].

In this setting, there already exist a number of type systems that describe information flow properties. Cardelli et al. introduced a type system for access control

[1] E-mail: hans@cs.aau.dk

[2] E-mail: mokyhn@diku.dk

doi:10.1016/j.entcs.2005.05.044

within Mobile Ambients [1]. Later, Merro and Sassone presented a type system [5] for border-crossing within the calculus of Boxed Ambients.

In this paper we present a first such type system for the calculus of Mobile Resources proposed by Godskesen et al. [3]. In contrast to e.g. the original formulation of Mobile Ambients, migration is *objective* – a process must be instructed to migrate by some other process. Moreover, the calculus is *linear*: once occupied, no process can enter a location until it becomes vacant. Finally, unlike e.g. Mobile Ambients, the MR calculus allows migration between arbitrary locations as long as their relative adresses *(paths)* are known.

Like [1] and [5], our type system is based on the notion of group. Moreover, we introduce a notion of security policy. Because migration may occur between any two locations, the type system needs to keep track of the nesting structure of a process. Still, the resulting type system is in our opinion less complex type than those of [1] and [5].

Our type system guarantees safety: Well-typed processes satisfy a specific safety property and, thanks to a subject reduction theorem, preserve safety under reduction. Moreover (and unlike [5]) we handle type inference, in that we describe how one many compute the minimal security policy which makes a process well typed.

2 Syntax and semantics of the MR calculus

2.1 Syntax

Our syntax is that of [3] extended with type annotations. See Table 1.

We assume countable sets of *names* $\mathcal{N}$ and *co-names* $\overline{\mathcal{N}}$. We let $m, n, \ldots$ range over $\mathcal{N}$, let $\tilde{m}, \tilde{n}, \ldots \subseteq \mathcal{N}$ and let α range over $\mathcal{N} \cup \overline{\mathcal{N}}$. $\delta, \delta', \ldots$ range over arbitrary sequences from $\mathcal{N}^+$, and $\gamma, \gamma', \ldots$ range over $\mathcal{N}^*$.

In MR locations are modelled as named *slots* which may be empty (written $\tilde{n}\,\lfloor \cdot \rfloor_m$) or contain processes. Slots may be named by any non-empty set of names $\tilde{n}$. If the name of a slot is unique, we omit brackets.

As slots can be nested, the exact location of a slot is described by its *path*. The MR calculus describes mobility by a move construct $\delta \rhd \overline{\delta'}$ that moves the contents of the slot with path δ to a slot with path δ', provided that an empty slot with this path exists. Slots $\tilde{n}\,\lfloor p \rfloor_m$ can be removed by the prefix $\natural\, m$.

The remaining process constructs are standard. They include prefixing with synchronization actions, parallel composition and replication. Note that a synchronization prefix may be a path, allowing communication between slots at different levels of nesting. Restriction makes the name n local to p. Restriction annotates the restricted name n by an $s \in \mathcal{S}$, a so-called *security policy type*, and $b \in B$, which is a group. These will be described in section 3.

Restriction is the sole binder of MR; free names $(\mathrm{fn}(p))$ and bound names $(\mathrm{bn}(p))$ of a process p are defined as expected. The substitution of a free name n for a free name m in p is written $p[m := n]$ and defined as expected. We write $p \equiv_\alpha p'$ if p' and p are equal up to renaming of bound names.

λ	$::=$	$\gamma\alpha$	direction path
	$\mid$	$\delta \rhd \overline{\delta'}$	move resource from the slot at δ to the slot at δ'
	$\mid$	$\natural n$	delete slot with name n
p	$::=$	$\mathbf{0}$	nil process
	$\mid$	$\lambda.p$	prefixed process
	$\mid$	$p_1 \parallel p_2$	parallel composition
	$\mid$	$!p$	replication
	$\mid$	$(n : s, b)p$	restriction
	$\mid$	$\tilde{n} \lfloor \cdot \rfloor_m$	empty slot
	$\mid$	$\tilde{n} \lfloor p \rfloor_m$	slot containing process

Table 1
The syntax of mobile resources

The set of slot names $\mathrm{sn}(p)$ of a process p is defined as the set of names used as names of slots in p.

2.2 Semantics

Our reduction semantics follows that of [3]; reductions correspond to communications and moves. Reductions occur within *evaluation contexts*.

Definition 2.1 Evaluation contexts $\mathcal{E}$ have the syntax

$$\mathcal{E} ::= (-) \mid \tilde{n} \lfloor \mathcal{E} \rfloor_m \mid (n : s, b)\mathcal{E} \mid \mathcal{E} \parallel p$$

$\mathcal{E}(p)$ means that p is inserted at the hole in $(-)$ in $\mathcal{E}$.

Definition 2.2 The relation of structural equivalence $\equiv$ on $\mathcal{P}$ is the least binary reflexive, symmetric and transitive relation satisfying the rules below.

(E1) $p \parallel \mathbf{0} \equiv p$

(E2) $p \parallel q \equiv q \parallel p$

(E3) $(n : s, b)p \parallel q \equiv (n : s, b)(p \parallel q)$, if $n \notin \mathrm{fn}(q)$

(E4) $(p \parallel p') \parallel p'' \equiv p \parallel (p' \parallel p'')$

(E5) $!p \equiv p \parallel !p$

(E6) $(n : s, b)\mathbf{0} \equiv \mathbf{0}$

(E7) $(n : s, b)\tilde{n} \lfloor p \rfloor_m \equiv \tilde{n} \lfloor (n : s, b)p \rfloor_m$, if $n \notin \tilde{n} \cup \{m\}$

(E8) $\mathcal{E}(p) \equiv \mathcal{E}(q)$, if $p \equiv q$.

(E$_\alpha$) $p \equiv q$, if $p \equiv_\alpha q$

Path contexts denote the positions where moves and synchronizations may occur.

Definition 2.3 The $\mathcal{N}^*$-indexed family of path contexts $\mathcal{C}_\gamma$ is defined by

$$\mathcal{C}_\varepsilon ::= (-) \qquad \mathcal{C}_{n\gamma} ::= \tilde{n} \lfloor \mathcal{C}_\gamma \parallel p \rfloor_m ,$$

for arbitrary m and p and $\tilde{n}$ with $n \in \tilde{n}$.

Resource contexts single out the sources and targets of moves.

Definition 2.4 The resource contexts $\mathcal{D}$ are defined as

$$\mathcal{D}_{\gamma n} ::= \mathcal{C}_\gamma(\tilde{n} \lfloor (-) \rfloor_m), \quad n \in \tilde{n}$$

We can now define the reduction semantics of the MR calculus.

Definition 2.5 Let $\longrightarrow$ be the least relation on $\mathcal{P}$ satisfying the following rules:

(R1) $\gamma\alpha.p \parallel \mathcal{C}_\gamma(\overline{\alpha}.q) \longrightarrow p \parallel \mathcal{C}_\gamma(q)$

(R2) $\gamma\delta_1 \rhd \overline{\gamma\delta_2}.p \parallel \mathcal{C}_\gamma\Big(\mathcal{D}_{\delta_1}(q) \parallel \mathcal{D}_{\delta_2}(\cdot)\Big) \longrightarrow p \parallel \mathcal{C}_\gamma\Big(\mathcal{D}_{\delta_1}(\cdot) \parallel \mathcal{D}_{\delta_2}(q)\Big)$

(R3) $\natural\, m.p \parallel \tilde{n} \lfloor r \rfloor_m \longrightarrow p, \quad r = \cdot \vee r = q$

(R4) $\mathcal{E}(p) \longrightarrow \mathcal{E}(q)$, if $p \longrightarrow q$

(R5) $p' \longrightarrow q'$, if $p' \equiv p$ and $p \longrightarrow q$ and $q \equiv q'$.

The rule (R1) describes synchronization: A process $\gamma\alpha.p$ can interact along a path γ with a process $\overline{\alpha}.q$. Rule (R2) describes migration. The (possibly empty) path γ describes the path common to the source slot and the target slot. Rule (R3) defines the semantics of slot deletion. Rules (R4) and (R5) describe the role of evaluation contexts and structural equivalence.

3 A type system for MR

Our type system assigns information of the form $A\&e$ to a process p. Here, A describes the potential interface of p and e the *effect* (cf. [4]), an abstract description of the potential moves of p.

3.1 Groups and security policies

Every slot is assigned a *group* name and a *security policy.* A security policy is a quadruple Sec[TG, TS, TR, TM], whose components are sets of types that describe the types of *trusted guests* (TG), *trusted senders* (TS), *trusted receivers* (TR) and *possible move actions* (TM).

If the set TR is known to be $\{T_1, \ldots, T_i\}$, we write TR$[T_1, \ldots, T_i]$. A slot whose security policy has this component allows moves to slots whose types are in $\{T_1, \ldots, T_i\}$. Similarly, a slot with component TS$[T_1, \ldots, T_i]$ allows moves from slots of types $\{T_1, \ldots, T_i\}$, and a slot with component TG$[T_1, \ldots, T_i]$ may contain slots of types $\{T_1, \ldots, T_i\}$. The wildcard type is $\star$; TR$[\star]$ means that the contents may be moved to slots of any type, $\textsc{ts}[\star]$ that content may be sent from slots of any type and $\textsc{tg}[\star]$ that slots of any type may occupy the slot.

If a security policy Σ has $\textsc{tr}$ as its set of trusted receivers, we write $\textsc{tr} \in \Sigma$. The same notation applies for $\textsc{ts}$, $\textsc{tg}$ and $\textsc{tm}$.

Definition 3.1 We let B range over a finite set of *group names* and let Σ range over the set of *security policy types* $\mathcal{S}$ whose syntax is

$T ::= B \mid \Sigma$	groups and security policies
$\Sigma ::= \textsc{Sec}[R\ S\ G\ M]$	security policy
$R ::= \textsc{tr}[\star] \mid \textsc{tr}[T_1, \ldots, T_i]$	trusted receivers
$S ::= \textsc{ts}[\star] \mid \textsc{ts}[T_1, \ldots, T_i]$	trusted senders
$G ::= \textsc{tg}[\star] \mid \textsc{tg}[T_1, \ldots, T_i]$	trusted guests
$M ::= \textsc{tm}[\star \rhd \star] \mid \textsc{tm}[W_1, \ldots, W_i]$	trusted moves
$W ::= \star \rhd B \mid B \rhd \star \mid B \rhd B$	basic moves

Note that in any of the components, a security policy can be specified directly instead of providing the name of a group. Also note that we (unlike [5]) can describe *full immobility*, as TR may be empty.

Example 1 Consider a slot n with security policy $\textsc{Sec}[\textsc{tr}[\text{CIA}, \text{FBI}]\ \ \textsc{ts}[\text{Capitol}]\ \ \textsc{tg}[\text{Politicians}]\ \ , \textsc{tm}[\text{Politicians} \rhd \text{Politicians}]]$. Here, moves from n to CIA and FBI are allowed (trusted receivers). Also moves from the Capitol to n are allowed (trusted senders). Politicians are welcome to guest n, and moves from any politician to another politician are allowed.

3.2 Effects

Effects describe potential migrations up to the groups involved.

Definition 3.2 The set of effects $\mathcal{E}$ of a program are subsets of the set $\{x \rhd y \mid x, y \in B\}$.

Definition 3.3 The set of abstracted effects $\mathcal{E}^*$ is $\{x \rhd y \mid x, y \in B \cup \mathcal{S} \cup \{\star\}\}$.

3.3 Judgements and rules

The *typing judgements* used in the following are of the form $\Delta, \Gamma \vdash p \Rightarrow A \ \&\ e$, where $A \subseteq \mathcal{N}$, $e \in \mathcal{E}$ and Δ and Γ are the security type environment and the group environment:

Definition 3.4 The security policy type environment Δ is a mapping $\Delta : \mathcal{N} \to \mathcal{S}$. The group environment Γ is a mapping $\Gamma : \mathcal{N} \to B$.

We only consider environments that are *well-formed*.

$$(\text{T-NIL})\frac{}{\Delta,\Gamma\vdash \mathbf{0}\Rightarrow\emptyset\ \&\ \emptyset}\qquad(\text{T-SYNC})\frac{\Delta,\Gamma\vdash p\Rightarrow A\ \&\ e}{\Delta,\Gamma\vdash \gamma\alpha.p\Rightarrow A\ \&\ e}$$

$$(\text{T-PAR})\frac{\Delta,\Gamma\vdash p_1\Rightarrow A_1\ \&\ e_1\qquad \Delta,\Gamma\vdash p_2\Rightarrow A_2\ \&\ e_2}{\Delta,\Gamma\vdash p_1\parallel p_2\Rightarrow A_1\cup A_2\ \&\ e_1\cup e_2}$$

$$(\text{T-BANG})\frac{\Delta,\Gamma\vdash p\Rightarrow A\ \&\ e}{\Delta,\Gamma\vdash !p\Rightarrow A\ \&\ e}\qquad(\text{T-REMOVE})\frac{\Delta,\Gamma\vdash p\Rightarrow A\ \&\ e}{\Delta,\Gamma\vdash \natural n.p\Rightarrow A\ \&\ e}$$

$$(\text{T-RES})\frac{\Delta[n\mapsto s],\Gamma[n\mapsto b]\vdash p\Rightarrow A\ \&\ e}{\Delta,\Gamma\vdash (n:s,b)p\Rightarrow A\setminus\{n\}\ \&\ e}$$

Table 2
Typing rules for standard processes

Definition 3.5 The *well-formedness predicate* for arbitrary environment pairs (Γ,Δ) and processes p, $wf(p,\Delta,\Gamma)$, is defined inductively by

$$\begin{aligned}
\mathrm{wf}(\mathbf{0},\Delta,\Gamma) &= \mathrm{true}\\
\mathrm{wf}(\gamma\alpha.p,\Delta,\Gamma) &= \mathrm{wf}(p,\Delta,\Gamma)\\
\mathrm{wf}(\delta\rhd\overline{\delta'}.p,\Delta,\Gamma) &= \mathrm{wf}(p,\Delta,\Gamma)\\
\mathrm{wf}(p_1\parallel p_2,\Delta,\Gamma) &= \mathrm{wf}(p_1,\Delta,\Gamma)\wedge\mathrm{wf}(p_2,\Delta,\Gamma)\\
\mathrm{wf}(!p,\Delta,\Gamma) &= \mathrm{wf}(p,\Delta,\Gamma)\\
\mathrm{wf}((n:s,b)p,\Delta,\Gamma) &= \mathrm{wf}(p,\Delta[n\mapsto s],\Gamma[n\mapsto b])\\
\mathrm{wf}(\tilde{n}\lfloor\cdot\rfloor_l,\Delta,\Gamma) &= \forall m,n\in\tilde{n}:\Delta(m)=\Delta(n)\wedge\Gamma(m)=\Gamma(n)\\
\mathrm{wf}(\tilde{n}\lfloor p\rfloor_l,\Delta,\Gamma) &= \mathrm{wf}(\tilde{n}\lfloor\cdot\rfloor_l,\Delta,\Gamma)\wedge\mathrm{wf}(p,\Delta,\Gamma)
\end{aligned}$$

The typing rules are found in Tables 2 and 3. The rules in 3 describe the typing of migration and employ some auxiliary predicates that we shall now define.

In the T-SLOT rule the *guesthood relation* $\sqsubseteq_{\Delta,\Gamma}$ checks that names of slots in p are allowed to guest the slot with name $\tilde{n}$.

Definition 3.6 Define the relation $\sqsubseteq_{\Delta,\Gamma}\subseteq\mathcal{P}(\mathcal{N})\times\mathcal{S}$ by $A\sqsubseteq_{\Delta,\Gamma}\Sigma$ iff $\star\in\mathrm{TG}$ or $\forall n\in A:\ \Gamma(n)\in\mathrm{TG}\ \vee\ \Delta(n)\in\mathrm{TG}$, where $\mathrm{TG}\in\Sigma$.

The *effect relation* $\sqsubseteq$ also used in T-SLOT checks that a given set of effects stays within the limits given by the security policy.

Definition 3.7 Define the relation $\sqsubseteq\subseteq\mathcal{E}^*\times\mathcal{S}$ by $e\sqsubseteq\Sigma$ iff $\star\rhd\star\in\mathrm{TM}$ or $\forall w\rhd w'\in e:\ w\rhd w'\in\mathrm{TM}\ \vee\star\rhd w'\in\mathrm{TM}\ \vee\ w\rhd\star\in\mathrm{TM}$, where $\mathrm{TM}\in\Sigma$.

In the T-MOVE rule, the consistent$_{\Delta,\Gamma}$ predicate checks if the paths δ and $\overline{\delta'}$ are *consistent*. By this we mean that any subslot along the path actually is allowed as a guest in its superslot according to the security policy.

Definition 3.8 Define the predicate $\mathrm{consistent}_{\Delta,\Gamma}(\delta)$ on $\mathcal{N}^+$ by

$$\mathrm{consistent}_{\Delta,\Gamma}(n_1\dots n_k)\ \text{iff}$$

$$(\text{T-Nil-Slot})\frac{}{\Delta, \Gamma \vdash \tilde{n} \lfloor \cdot \rfloor_m \Rightarrow \tilde{n} \;\&\; \emptyset}$$

$$(\text{T-Slot})\frac{\Delta, \Gamma \vdash p \Rightarrow A \;\&\; e}{\Delta, \Gamma \vdash \tilde{n} \lfloor p \rfloor_m \Rightarrow A \cup \tilde{n} \;\&\; e} \begin{pmatrix} n \in \tilde{n} & & \\ A & \sqsubseteq_{\Delta,\Gamma} & \Delta(n) \\ e & \sqsubseteq & \Delta(n) \end{pmatrix}$$

$$(\text{T-Move})\frac{\Delta, \Gamma \vdash p \Rightarrow A \;\&\; e}{\Delta, \Gamma \vdash \delta \triangleright \overline{\delta'}.p \Rightarrow A \;\&\; e \cup \{\Gamma(n) \triangleright \Gamma(n')\}} \begin{pmatrix} \delta = \gamma_1 n, \\ \delta' = \gamma_2 n' \\ n \bowtie_{\Delta,\Gamma} n' \\ \text{consistent}_{\Delta,\Gamma}(\delta) \\ \text{consistent}_{\Delta,\Gamma}(\delta') \end{pmatrix}$$

Table 3
Typing rules for MR processes

$\forall 1 \leq i < k : \text{TG} \in \Delta(n_i) \Rightarrow \star \in \text{TG} \vee \text{TG}' \subseteq \text{TG}$, where $\text{TG}' \in \Delta(n_{i+1})$ and $\forall 1 \leq i \leq k : \text{TM} \in \Delta(n_i) \Rightarrow \text{TM} \sqsubseteq \Delta(n_{i-1})$.

Finally, the relation $\bowtie_{\Delta,\Gamma}$ guarantees that the receiving slot allows more or just as many guests as the sending slot and that the slots grant each other access.

Definition 3.9 The relation $\bowtie_{\Delta,\Gamma} \subseteq \mathcal{N} \times \mathcal{N}$ is defined by $m \bowtie_{\Delta,\Gamma} n$ if

- (Guests) $\star \in \text{TG}'$ or $\text{TG} \subseteq \text{TG}'$, where $\text{TG} \in \Delta(m)$, $\text{TG}' \in \Delta(n)$.
- (Sender and receiver)
  (i) $\star \in \text{TR}$ or $\Gamma(n) \in \text{TR}$ or $\Delta(n) \in \text{TR}$, where $\text{TR} \in \Delta(m)$ and
  (ii) $\star \in \text{TS}$ or $\Gamma(m) \in \text{TS}$ or $\Delta(m) \in \text{TS}$, where $\text{TS} \in \Delta(n)$
- (Effects) If it holds that $\text{TM} \sqsubseteq \Delta(n)$ where $\text{TM} \in \Delta(m)$.

Note that the conditions on senders and receivers reflect the usual notions of co/contravariance found in e.g. [6].

Definition 3.10 Let p be a process and Δ, Γ be environments such that $\text{wf}(p, \Delta, \Gamma)$. We say that p is *well-typed* under Δ, Γ iff $\exists A, e : \Delta, \Gamma \vdash p \Rightarrow A \;\&\; e$.

3.4 Properties of the type system

The two central properties of our type system are that typability is preserved along reductions and that *well-typed processes are well-behaved.*

Theorem 3.11 (Subject reduction) *For any well-typed process $\Delta, \Gamma \vdash p \Rightarrow A \;\&\; e$ it holds that if $p \longrightarrow p'$ then $\Delta, \Gamma \vdash p' \Rightarrow A' \;\&\; e'$ is also well-typed and $A' \subseteq A$ and $e' \subseteq e$.*

Definition 3.12 The effect extraction function effects(Γ, p) is defined by

$$\begin{aligned}
\text{effects}(\Gamma, \mathbf{0}) &= \emptyset \\
\text{effects}(\Gamma, \gamma\alpha.p) &= \text{effects}(\Gamma, p) \\
\text{effects}(\Gamma, \delta \triangleright \overline{\delta'}.p) &= \text{effects}(\Gamma, p) \cup \{\Gamma(n) \triangleright \Gamma(n')\} \\
&\quad \text{where } \delta = \gamma_1 n \text{ and } \overline{\delta'} = \overline{\gamma_2 n'} \\
\text{effects}(\Gamma, \natural\, n.p) &= \text{effects}(\Gamma, p) \\
\text{effects}(\Gamma, p_1 \parallel p_2) &= \text{effects}(\Gamma, p_1) \cup \text{effects}(\Gamma, p_2) \\
\text{effects}(\Gamma, !p) &= \text{effects}(\Gamma, p) \\
\text{effects}(\Gamma, (n : s, b)p) &= \text{effects}(\Gamma, p) \\
\text{effects}(\Gamma, \tilde{n} \lfloor \cdot \rfloor_m) &= \emptyset \\
\text{effects}(\Gamma, \tilde{n} \lfloor p \rfloor_m) &= \text{effects}(\Gamma, p)
\end{aligned}$$

We call a process *safe* in (Γ, Δ) if every visible subslot is allowed by the guesthood relation (Definition 3.6) and all effects are allowed by the effect relation (Definition 3.7).

Let subp(n, p) denote the set of subprocesses of process p that are either slots named by n or subprocesses under such slots.

Definition 3.13 Given security environment Δ and group environment Γ, process p is *safe* if for all free slot names n in p we have that

$$\forall \tilde{n} \lfloor p' \rfloor_m \in \text{subp}(n, p) : \text{sn}(p') \setminus \text{bn}(p') \sqsubseteq_{\Delta,\Gamma} \Delta(n) \wedge \text{effects}(p') \sqsubseteq \Delta(n)$$

The following lemma states that the typing rules always extract the correct information about visible slots and potential moves.

Lemma 3.14 *For any process p it holds that if $\Delta, \Gamma \vdash p \Rightarrow A \,\&\, e$ then $A =$ sn$(p) \setminus$ bn(p) and $e =$ effects(p).*

Theorem 3.15 (Safety) *Any well-typed process $\Delta, \Gamma \vdash p \Rightarrow A \,\&\, e$ is safe.*

Corollary 3.16 *If p is well-typed under Δ, Γ and $p \rightarrow^* p'$, then p' is safe.*

Safe processes need not be well-typed. A simple counterexample is the process $\mathtt{a}\lfloor \mathbf{0} \rfloor \parallel \mathtt{b}\lfloor \cdot \rfloor \parallel \mathtt{a} \triangleright \overline{\mathtt{b}}.\mathbf{0}$ with typings $\Gamma = \{\mathtt{a} \mapsto a', \mathtt{b} \mapsto b'\}$ and $\Delta = \{\mathtt{a} \mapsto$ $Sec[\text{TR}[]\text{TS}[]\text{TG}[\star]\text{TM}[]], \mathtt{b} \mapsto Sec[\text{TR}[]\text{TS}[]\text{TG}[]\text{TM}[]]\}$. This process is safe with respect to Γ and Δ but *not* typable, as b does not trust any senders – in particular, a is not trusted. Thus the move $\mathtt{a} \triangleright \overline{\mathtt{b}}$ is not well-typed.

Example 2 Consider the processes

$$\begin{aligned}
\text{FireKit} &= \mathtt{Matchbox}\lfloor \mathtt{Match}\lfloor \cdot \rfloor \rfloor \parallel \mathtt{Acetone}\lfloor \cdot \rfloor \\
\text{MakeAFire} &= \text{FireKit} \parallel \mathtt{MatchBox} \triangleright \overline{\mathtt{Acetone}}.\mathbf{0} \\
\text{Camp} &= \mathtt{FireSite}\lfloor \text{MakeAFire} \rfloor \\
\text{GoToLibrary} &= \text{Camp} \parallel \mathtt{Library}\lfloor \cdot \rfloor \parallel \mathtt{FireSite} \triangleright \overline{\mathtt{Library}} \quad \parallel \mathtt{People}\lfloor \cdot \rfloor
\end{aligned}$$

with group environment $\Gamma = \{\texttt{Matchbox} \mapsto b, \texttt{Match} \mapsto m, \texttt{Acetone} \mapsto a,$ $\texttt{FireSite} \mapsto f, \texttt{Library} \mapsto l, \texttt{People} \mapsto p\}$ and security policy environment

$$\begin{aligned} \Delta = \{\ & \texttt{MatchBox} \mapsto \textsc{Sec}[\textsc{tr}[\star]\ \textsc{ts}[b]\ \textsc{tg}[m]\ \textsc{tm}[\ ]] \\ & \texttt{Match} \mapsto \textsc{Sec}[\textsc{tr}[\ ]\ \textsc{ts}[\ ]\ \textsc{tg}[\ ]\ \textsc{tm}[\ ]] \\ & \texttt{Acetone} \mapsto \textsc{Sec}[\textsc{tr}[\ ]\ \textsc{ts}[b]\ \textsc{tg}[m]\ \textsc{tm}[\ ]] \\ & \texttt{FireSite} \mapsto \textsc{Sec}[\textsc{tr}[l]\ \textsc{ts}[\ ]\ \textsc{tg}[b,m,a]\ \textsc{tm}[b \rhd a]] \\ & \texttt{Library} \mapsto \textsc{Sec}[\textsc{tr}[p,l]\ \textsc{ts}[p,l]\ \textsc{tg}[p]\ \textsc{tm}[\star]] \\ & \texttt{People} \mapsto \textsc{Sec}[\textsc{tr}[\ ]\ \textsc{ts}[\ ]\ \textsc{tg}[\ ]\ \textsc{tm}[\ ]] \\ \}& \end{aligned}$$

The process Camp is well-typed as fire starting is an allowed activity at the firesite. This activity cannot be moved to the library, since the library does not permit matches and acetone as guests. Thus the move $\texttt{FireSite} \rhd \overline{\texttt{Library}}$ is not well-typed by the side conditions of T-MOVE.

4 Minimal security policy environments

We now consider the problem of finding a *minimal security policy environment* for a given process p, with a given group environment Γ. The idea is to construct the minimal security policy environment Δ^* from p that makes p typable with respect to Δ^* and Γ.

We make some simplifying assumptions on processes. For any process p we shall assume that it has been alpha-converted to a form where $\text{fn}(p) \cap \text{bn}(p) = \emptyset$ and all bound names are distinct.

Security policies occurring in restrictions are transformed as follows:

- We replace stars $\star$ occurring in $\textsc{tr}, \textsc{ts}$ and $\textsc{tg}$ by $\text{ran}(\Gamma)$, the range of Γ.
- We replace all $\star \rhd w$ with $w_1 \rhd w, \dots, w_k \rhd w$ where $\{w_1, \dots, w_k\} = \text{ran}(\Gamma)$ and $w \in \text{ran}(\Gamma)$. All occurrences of $w \rhd \star$ are replaced with $w \rhd w_1, \dots, w \rhd w_k$. Finally $\star \rhd \star$ is replaced with $\{w \rhd w' \mid w, w' \in \text{ran}(\Gamma)\}$.

The above transform can be applied to the set $\mathcal{S}$. Call the resulting set $\mathcal{S}_{\setminus\star}$. We can then define an ordering on security policies in $\mathcal{S}_{\setminus\star}$ as follows:

Definition 4.1 Let $\Sigma, \Sigma' \in \mathcal{S}_{\setminus\star}$.

$$\Sigma \subseteq \Sigma' \text{ iff}$$
$$\textsc{tr} \subseteq \textsc{tr}' \wedge \textsc{ts} \subseteq \textsc{ts}' \wedge \textsc{tg} \subseteq \textsc{tg}' \wedge \textsc{tm} \subseteq \textsc{tm}'$$
$$\text{where } \textsc{tr}, \textsc{ts}, \textsc{tg}, \textsc{tm} \in \Sigma \text{ and } \textsc{tr}', \textsc{ts}', \textsc{tg}', \textsc{tm}' \in \Sigma'$$

Definition 4.2 Define the order $\leq$ on $\left\{\Delta \mid \Delta : \mathcal{N} \to \mathcal{S}_{\setminus\star}\right\}$ by

$$\Delta \leq \Delta' \text{ iff } \forall n \in \mathrm{dom}(\Delta) : \Delta(n) \subseteq \Delta'(n).$$

and $\mathrm{dom}(\Delta) = \mathrm{dom}(\Delta')$.

Lemma 4.3 $(\left\{\Delta \mid \Delta : \mathcal{N} \to \mathcal{S}_{\setminus\star}\right\}, \leq)$ *is a complete lattice.*

We now show how to extract a set of constaints from a process. In our presentation, $\langle A \mid B \rangle$ denotes a tuple consisting of sets A and B and $\oplus$ denotes componentwise union of such tuples.

The algorithm presented in Table 4 extracts a pair of constraint sets $\langle C_1 | C_2 \rangle$ from a given process p under a given group environment Γ. Here C_1 contains constraints on the form $D \subseteq (n, t)$ where D is a set of types, and n is a name and $t \in \{\textsc{tr}, \textsc{ts}, \textsc{tg}, \textsc{tm}\}$. For example, a constraint $D \subseteq (n, \textsc{tg})$ states that the set of types D is a subset of trusted guests in the security policy of n. The set C_2 contains constraints on the form $(m, t) \subseteq (n, t)$, $t \in \{\textsc{tg}, \textsc{tm}\}$. As an example $(m, \textsc{tm}) \subseteq (n, \textsc{tm})$ states that the trusted moves of m should also be trusted as valid moves by n.

Define N as $\mathrm{sn}(p)$ and let l be the number of free names in N. Given a pair of constraint sets $\langle C_1 | C_2 \rangle$ we can now compute a minimal security policy environment for p by finding the fixpoint of the composition of the two functions $f_1, f_2 : \mathcal{P}(N)^l \times \mathcal{P}(N)^l \to \mathcal{P}(N)^l \times \mathcal{P}(N)^l$.

The function f_1 finds the minimal requirements of a security policy as defined by the constraints C_1:

$$f_1(\boldsymbol{x}_G, \boldsymbol{x}_M) = (\boldsymbol{y}_G, \boldsymbol{y}_M)$$

where the components of $\boldsymbol{y}_G$ and $\boldsymbol{y}_M$ are given by $y_G^i = x_G^i \cup \bigcup_{(D \subseteq (i, \textsc{tg})) \in C_1} D$ and $y_M^i = x_M^i \cup \bigcup_{(D \subseteq (i, \textsc{tm})) \in C_1} D$

The function f_2 finds the requirements as defined by the constraints C_2:

$$f_2(\boldsymbol{x}_G, \boldsymbol{x}_M) = (\boldsymbol{y}_G, \boldsymbol{y}_M)$$

where $y_G^i = x_G^i \cup \bigcup_{((m, \textsc{tg}) \subseteq (i, \textsc{tg})) \in C_2} x_G^m$ and $y_M^i = x_M^i \cup \bigcup_{((m, \textsc{tm}) \subseteq (i, \textsc{tm})) \in C_2} x_M^m$

The function $h = f_2 \circ f_1$ is clearly monotone since f_1 and f_2 are. By Tarski's theorem, h has a least fixpoint which we call $\boldsymbol{x}^*$. This is also the least prefixpoint under pointwise set inclusion, so $\boldsymbol{x}^*$ is exactly a solution of the constraints in C_1 and C_2. $\boldsymbol{x}^*$ can be computed iteratively using Tarski's theorem as $\mathrm{ran}(h)$ is finite and as the set of group names is finite. The security policy environment can be found from $\boldsymbol{x}^*$ as

$$\Delta = [n \mapsto \Sigma_n]_{n \in N}$$

where

$$\Sigma_n = \textsc{Sec}[\textsc{tr}[\bigcup_{(D \subseteq (n, \textsc{tr})) \in C_1} D]\ \textsc{ts}[\bigcup_{(D \subseteq (n, \textsc{ts})) \in C_1} D]\ \textsc{tg}[\pi_n \pi_1 \boldsymbol{x}^*]\ \textsc{tm}[\pi_n \pi_2 \boldsymbol{x}^*]]$$

```
function $\chi(p, \Gamma)$ =
    case $p$ of
    $\mathbf{0}$              : $\langle \emptyset \mid \emptyset \rangle$
    $\gamma\alpha.p$           : $\chi(p, \Gamma)$
    $m_1 \ldots m_k \triangleright$
      $n_1 \ldots n_l.p$ : $\Big\langle \{\{\Gamma(n_l)\} \subseteq (m_k, \text{TR}),$
                      $\{\Gamma(m_k)\} \subseteq (n_l, \text{TS})\} \Big|$
                      $\{\{(m_k, \text{TG}) \subseteq (n_l, \text{TG}), (m_k, \text{TM}) \subseteq (n_l, \text{TM})\}\} \cup$
                      $\{(m_i, \text{TG}) \subseteq (m_{i-1}, \text{TG}), (m_i, \text{TM}) \subseteq (m_{i-1}, \text{TM})\}_{1<i\leq k} \cup$
                      $\{(n_i, \text{TG}) \subseteq (n_{i-1}, \text{TG}), (n_i, \text{TM}) \subseteq (n_{i-1}, \text{TM})\}_{1<i\leq l}\Big\rangle \bigoplus$
                        $\chi(p, \Gamma)$
    $\natural\, n.p$           : $\chi(p, \Gamma)$
    $p_1 \parallel p_2$        : $\chi(p_1, \Gamma) \oplus \chi(p_2, \Gamma)$
    $!p$               : $\chi(p, \Gamma)$
    $(n : s, b)(p)$    : $\langle \{\pi_1 s \subseteq (n, \text{TR}), \pi_2 s \subseteq (n, \text{TS}),$
                      $\pi_3 s \subseteq (n, \text{TG}), \pi_4 s \subseteq (n, \text{TM})\} \mid \emptyset \rangle \oplus$
                        $\chi(p, \Gamma[n \mapsto b])$
    $\tilde{n}\lfloor \cdot \rfloor$             : $\langle \emptyset \mid \emptyset \rangle$
    $\tilde{n}\lfloor p \rfloor$             : $\langle \{\Gamma(\text{sn}(P)) \subseteq (n, \text{TG}), \text{effects}(p) \subseteq (n, \text{TM})\}_{n \in \tilde{n}} \mid \emptyset \rangle \oplus$
                        $\chi(p, \Gamma)$
          end
end
```

Table 4
χ extracts constraints from a process for a given group environment Γ.

Definition 4.4 We write $(p, \Gamma) \rightsquigarrow \Delta$ if Δ is found as above from p and Γ.

Due to the presence of restriction, one may specify processes which do not have a minimal security policy environment. As an example consider

$$(\mathtt{a} : s, a')(\mathtt{b} : s, b')(\mathtt{a}\lfloor \mathbf{0} \rfloor \parallel \mathtt{b}\lfloor \cdot \rfloor \parallel \mathtt{a} \triangleright \overline{\mathtt{b}}.\mathbf{0})$$

where s is $\text{SEC}[\text{TR}[]\ \text{TS}[]\ \text{TG}[]\ \text{TM}[]]$. Since the slots a and b do not trust each another, the move operation is impossible – i.e. untypable. When this process is processed by the algorithm in Table 4 we among other constraints get the constraints

$\{\{b'\} \subseteq (\mathtt{a}, \textsc{tr}), \{a'\} \subseteq (\mathtt{b}, \textsc{ts})\}$. The solution to this system will contain security policies for a and b which have been enlarged so that $\mathtt{a}$ and $\mathtt{b}$ now trust each other. The security policies have changed compared to their original security policy s. This problem gives rise to the following definition.

Definition 4.5 A process p with group environment Γ is *inconsistent* if it contains a subprocess $(n : s, b)p'$ for which $\Delta(n) \neq s$, where $(p, \Gamma) \rightsquigarrow \Delta$.

Definition 4.6 Given $(p, \Gamma) \rightsquigarrow \Delta$ and assume that p is not inconsistent. Then the minimal security policy environment Δ^* for p and Γ is defined as

$$\Delta^* \overset{\text{def}}{=} \Delta_{|\text{sn}(P)\setminus\text{bn}(P)}.$$

Theorem 4.7 *Let Δ^* be the minimal security policy environment for the consistent process p with respect to the group environment Γ. It holds that*

(i) $\Delta^*, \Gamma \vdash p \Rightarrow A \ \&\ e$ *for some A and e.*

(ii) $\nexists \Delta' : \Delta' < \Delta^* \wedge \Delta', \Gamma \vdash p \Rightarrow A \ \&\ e$ *for some A and e.*

5 Conclusions and future work

In this paper we have described a type system for the Calculus of Mobile Resources [3]. The type system allows us to reason about border-crossing phenomena, something that is less straightforward in the setting of the MR calculus. Our type system preserves a safety property. To our knowledge, this is the first such type system proposed for the calculus. With our notion of security policy it becomes straightforward to describe not just border-crossing properties but also properties such as immobility (contrast this with e.g. [5]).

We also describe an algorithm for finding the minimal security policy for a processes. A complexity analysis of our algorithm is a topic for further work.

One may imagine stronger type systems. In Definition 3.9 we require that the security policy regarding guests and effects for the receiving slot should be as least as general as that of the sending slot. Consider $a \lfloor b \lfloor P \rfloor \rfloor \parallel c \lfloor \cdot \rfloor \parallel d.a \rhd \overline{c}.\mathbf{0}$. Here there is no co-action $\overline{d}$, so the move $a \rhd \overline{c}$ never occurs. When this process is typed, the slot c must allow the same guests and effects as a. This gives a generality of the security policy for c which is unnecessary. It would be desirable to be able to type processes with respect to how they change configurations dynamically. This would in turn require a completely different type system in which issues concerning causality would become central.

References

[1] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Ambient groups and mobility types. In *Proceedings of TCS 2000 (Sendai, Japan)*, LNCS 1872, pp. 333–347, August 2000.

[2] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.

[3] J. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. *Proceedings of CONCUR'02, LNCS 2421, pp. 272-287.*, 1644:p.230, 2002.

[4] John M. Lucassen and David K. Gifford. Polymorphic effect systems. In *POPL*, pages 47–57, 1988.

[5] M. Merro and V. Sassone. Typing and subtyping mobility in boxed ambients. *CONCUR'02*, LNCS 2421:304–320, 2002.

[6] B. Pierce and D. Sangiorgi. Typing and Subtyping for Mobile Processes *Mathematical Structures in Computer Science*, vol. 6, pp. 409–454, 1996.

[7] J. Stefani. A calculus of kells. In 2nd Workshop on Foundations of Global Computing. To appear, 2003.

[8] Jan Vitek and Giuseppe Castagna. Seal: A framework for secure mobile computations. *Lecture Notes in Computer Science*, 1686, 1999.