Type Inference for Mobile Ambients in Prolog

Elio Giovannetti [1]

Dipartimento di Informatica
Università di Torino
corso Svizzera 185, 10149 Torino, Italia
e-mail: elio@di.unito.it

Abstract

The type system for the ambient calculus M^3 [8] is presented in a new form that derives the type of a term with the minimal set of mobility assumptions, and is therefore more amenable than the original form to a translation into a type inference algorithm. From the new formulation a Prolog program is derived, which implements a type inference algorithm for M^3 analogous to the one previously specified through formal rules. The implementation exploits in the standard way the peculiarities of the logic programming paradigm, and is therefore, in a sense, more abstract than the original algorithm's specification itself.

1 Introduction

Mobility (of devices, software and even running programs) has become an important aspect of computation and communication. Several theoretical models have been proposed for describing this aspect, reasoning about it, and thus controlling the behaviour of systems where mobility is a relevant feature.

Most models are based on the notion of a named *location* [9], or on the one of a *mobile ambient*, first introduced in [7]. *Behavioural* type systems have been extensively proposed as a privileged means for controlling mobility, interferences, security, resource usage, thus extending the notion of type well beyond its original meaning in logics and computer science. In this new setting, a type is simply a behavioural property of a piece of program, concerning whatever aspect one is interested in; types may therefore have, in different systems and for different purposes, the most diverse and original forms.

In any ambient calculus a type system is needed at least to ensure the correctness of communication. The first system also controlling mobility and opening, though only in a binary way (mobile/immobile, etc.), was the one in [5]. The next step was

[1] Partially supported by MURST Cofin'01 NAPOLI Project and by EU within the FET - Global Computing initiative, project DART IST-2001-33477.

performed in [6]: the notion of an ambient *group* allowed the definition of systems able to statically track ambient movements more finely, while avoiding dependency of types from ambient names.

Type systems performing a non-trivial static flow analysis are the ones proposed in [3,4], where the statically provided approximation of the runtime behaviour is rather fine-grained. The calculus of *Boxed Ambients* [2] was the first ambient calculus that replaced ambient opening with inter-ambient communication, thus requiring that even its simplest type system deals with communication and mobility at the same time (an ambient cannot move where it should communicate with an ambient whose language it does not understand). In [10] *subtyping* was for the first time extensively used in a type system for ambients, to achieve greater expressive power and flexibility.

The system M^3, presented in [8], is a variant of the Calculus of Mobile Ambients [7] where the open capability is also dropped and a new form of process mobility is introduced, akin to the go primitive of the Distributed π-calculus [9]. As in the above cited proposals, the calculus is equipped with a type system which controls mobility and ensures that no runtime error due to communication of inappropriate values may occur.

Being – as is often the case – syntax-directed, the simple typing rules of M^3 are usable for type-checking, for they can be read as the specification of an algorithm that, receiving in input a typing judgement $\mathrm{E} \vdash_{\mathrm{M}^3} \mathit{trm}{:}\mathit{Type}$, checks whether the judgment is derivable in the system.

On the other hand, they cannot be used for type reconstruction, i.e., for generating a valid typing judgment $\mathrm{E} \vdash \mathit{trm}{:}\mathit{Type}$, if one exists, from a given *raw* term trm, with $|\mathit{trm}| = \mathsf{trm}$, if we call raw term a pseudo-term where all type information is absent, and if $|\mathit{trm}|$ is the raw term obtained from *trm* by erasing all type information.

This is mainly due to an implicit form of weakening present in the system, expressed by the simple set-theoretic conditions on ambient names and group types in the rule premisses. In [8] a type reconstruction algorithm is therefore separately specified by means of a set of formal rules; the type system of M^3 enjoys the property of principality in the sense of [12], and the algorithm computes the principal typing (for a raw term trm).

In this paper we follow a complementary approach: the typing rules are recast into a form that eliminates the implicit weakening (possibly isolating it in a final optional step, if typings other than principal are desired) and easily translates into a logic programming implementation of the type inference. Having a Horn-like form, each rule corresponds to a Prolog clause; unification and substitution application, built-in in the Prolog interpreter, do not have to be explicitly indicated. As a result, the Prolog prototypical implementation is, in a sense, more abstract than the original algorithm specification, as is well-known to happen (e.g., for ML-like type systems).

Only some mechanisms have to be explicitly programmed. The main one is the generation of a well-formed (minimal) environment for a term from the union of two environments corresponding to two subterms. This is obtained through a plain loop that iteratively computes a sort of fixed point of an environment-simplifying transformation, and performs at each step the necessary unifications. A (simpler) fixed point procedure is also needed for transforming *pre-types* into types through an unfolding of implicit information.

The paper is structured as follows: in Section 2 the syntax and operational semantics of the calculus are briefly recalled; in Section 3 the syntax and the intuitive meanings of types and type assumptions (environments) are described; in Section 4 the new formulation of the typing rules is given and compared with the one in [8]; in Section 5 the main aspects of the implementation are described. Finally, in Section 6, some short conclusions are drawn, with indications of future work.

2 The calculus

The syntax of terms is shown in Figure 1. Observe, w.r.t. the Calculus of Mobile Ambients, the absence of open and the presence of the new primitive to for "naked" process mobility (by "naked processes" we intend processes not of the form $m[P]$). Also, synchronous output is allowed, of which the asynchronous version is a particular case.

We follow the traditional distinction between letters m, n, ... for *ambient names* and letters x, y, ... for input variables. Formally they are however in a single syntactic category, which we call *variables*; simply, ambient names are variables that either are free and do not occur in prefix or path prefix position (e.g., $x.P$, $x.M$), or are bound by the ν-binder (but not by the input binder). The letter ξ will be used to denote a generic variable that may be an ambient name or an input variable indifferently.

Also, the *Barendregt convention* [1] is assumed to hold on variables and on *group names* (see below) for any term or set of terms one is considering: all the bound variables (or respectively the bound group names) are distinct among themselves, and distinct from all the free variables (or respectively the free group names). In this way all the problems connected with α-conversion and capturing of free names are avoided.

The syntax of raw terms, given in Figure 2, is obtained from the ordinary syntax by erasing the type in the binders of input and name restriction, and by eliminating the group restriction construct altogether. The definition of the erasure function $|\_|$, which transforms an ordinary term *trm* into a raw term $|$*trm*$|$, is obvious.

The operational semantics consists, as usual, of a reduction relation, along with a structural congruence which allows trivial syntactic restructuring of a term so that a reduction rule can next be applied.

Structural congruence, shown in Figure 3, differs from the one given in [8] in that the scope of group restriction may also extrude across prefix, input, output and replication; thus group restrictions may always be brought to the outermost position in the term. This feature is not included in the usual definitions, since it does not serve the primary purpose of allowing the formation of redexes. It is however in

trm, trm_i ::=		**terms**
	M	messages
	P	processes
$M,\ N,\ L$ ::=		**messages**
	$\xi, \ldots m, n \ldots x, y, \ldots$	variables, i.e., ambient names and input variables
	$\mathsf{in}\ M$	moves the containing ambient into ambient M
	$\mathsf{out}\ M$	moves the containing ambient out of ambient M
	$\mathsf{to}\ M$	goes out from its ambient into sibling ambient M
	$M.M'$	path
$P,\ Q,\ R$ ::=		**processes**
	0	null
	$M\,.\,P$	prefixed
	$\langle M \rangle P$	synchronous output
	$(x{:}\,W)P$	typed input
	$P \mid Q$	parallel composition
	$M[P]$	ambient
	$!\,P$	replication
	$(\nu n{:}\ \mathsf{amb}(g))P$	name restriction
	$(\nu\{\overrightarrow{\mathbf{g}{:}\mathbf{G}}\}_{(k)})P$	group restriction

where: W is a message type, g is a group name, $\nu\{\overrightarrow{\mathbf{g}{:}\mathbf{G}}\}_{(k)}$ is a concise notation for $\nu\{g_1{:}\ G_1, \ldots, g_k{:}\ G_k\}$, with $g_1, \ldots, g_k$ group names and $G_1, \ldots, G_k$ group types (see Fig. 5).

Fig. 1: Syntax

$\mathsf{trm},\ \mathsf{trm}_i$ ::=		**raw terms**
	M	messages
	P	raw processes
$\mathsf{P},\ \mathsf{Q},\ \mathsf{R}$ ::=		**raw processes**
	0	null
	$M\,.\,\mathsf{P}$	prefixed
	$\langle M \rangle \mathsf{P}$	synchronous output
	$(x)\mathsf{P}$	input
	$\mathsf{P} \mid \mathsf{Q}$	parallel composition
	$M[\mathsf{P}]$	ambient
	$!\,\mathsf{P}$	replication
	$(\nu n)\mathsf{P}$	name restriction

Fig. 2: Syntax of raw terms

natural agreement with the other defining clauses, and allows us to avoid the rising of a purely formal problem, due to the fact that the type reconstruction algorithm leaves undetermined where group restrictions might be inserted in the term. With the new definition this is immaterial: all the different typed terms that would be obtained are equivalent.

Observe that, owing to the Barendregt convention, the second rule of scope extrusion is safe even in the case ξ is an input variable. For the rest, the congruence is almost the standard relation found in [5].

equivalence:

$$P \equiv P \qquad P \equiv Q \implies Q \equiv P \qquad P \equiv Q,\ Q \equiv R \implies P \equiv R$$

congruence:

$$\begin{array}{ll} P \equiv Q \implies M\,.\,P \;\equiv\; M\,.\,Q & P \equiv Q \implies M[P] \;\equiv\; M[Q] \\ P \equiv Q \implies \langle M\rangle P \;\equiv\; \langle M\rangle Q & P \equiv Q \implies !\,P \;\equiv\; !\,Q \\ P \equiv Q \implies (x{:}\,W)P \;\equiv\; (x{:}\,W)Q & P \equiv Q \implies (\nu n{:}\ \mathsf{amb}(g))P \;\equiv\; (\nu n{:}\ \mathsf{amb}(g))Q \\ P \equiv Q \implies P\,|\,R \;\equiv\; Q\,|\,R & P \equiv Q \implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})Q \end{array}$$

prefix associativity:

$$(M.M').P \;\equiv\; M.M'.P$$

parallel composition – associativity, commutativity, zero:

$$P\,|\,Q \;\equiv\; Q\,|\,P \qquad (P\,|\,Q)\,|\,R \;\equiv\; P\,|\,(Q\,|\,R) \qquad P\,|\,\mathsf{0} \;\equiv\; P$$

replication:

$$!P \;\equiv\; P\,|\,!P \qquad !\,\mathsf{0} \;\equiv\; \mathsf{0}$$

restriction swapping and group restriction splitting :

$$n \neq m \implies (\nu n{:}\ \mathsf{amb}(g))(\nu m{:}\ \mathsf{amb}(g'))P \;\equiv\; (\nu m{:}\ \mathsf{amb}(g'))(\nu n{:}\ \mathsf{amb}(g))P$$

$$g_i \neq g'_j \& g_i \notin GN(G'_j) \& g'_j \notin GN(G_i)(1 \leq i \leq k)(1 \leq j \leq h)$$
$$\implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(\nu\{\overrightarrow{\mathbf{g}':\mathbf{G}'}\}_{(h)})P \;\equiv (\nu\{\overrightarrow{\mathbf{g}':\mathbf{G}'}\}_{(h)})(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P$$

$$g \neq g_i(1 \leq i \leq k) \implies (\nu n{:}\ \mathsf{amb}(g))(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(\nu n{:}\ \mathsf{amb}(g))P$$

$$g_{k+j} \notin GN(G_i)(1 \leq i \leq k)(1 \leq j \leq h)$$
$$\implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k+h)})P \;\equiv\; (\nu\{g_1{:}\ G_1, \ldots, g_k{:}\ G_k\})(\nu\{g_{k+1}{:}\ G_{k+1}, \ldots, g_{k+h}{:}\ G_{k+h}\})P$$

scope extrusion:

$$\begin{array}{l} n \notin AN(Q) \implies (\nu n{:}\mathsf{amb}(g))P\,|\,Q \;\equiv\; (\nu n{:}\mathsf{amb}(g))(P\,|\,Q) \\ n \neq \xi \implies \xi[(\nu n{:}\ \mathsf{amb}(g))P] \;\equiv\; (\nu n{:}\ \mathsf{amb}(g))\xi[P] \\ g_i \notin GN(Q)(1 \leq i \leq k) \implies (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P\,|\,Q \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(P\,|\,Q) \\ \xi[(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P] \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\xi[P] \\ M.(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})M.P \\ g_i \notin GN(W)(1 \leq i \leq k) \implies (x{:}W)(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})(x{:}W)P \\ \langle M\rangle(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\langle M\rangle P \\ !\,(\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \;\equiv\; (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\,!\,P \end{array}$$

equivalence to zero:

$$\begin{array}{l} (\nu n{:}\ \mathsf{amb}(g))\,\mathsf{0} \;\equiv\; \mathsf{0} \\ (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})\,\mathsf{0} \;\equiv\; \mathsf{0} \\ (\nu n{:}\,\mathsf{amb}(g))n[\mathsf{0}] \;\equiv\; \mathsf{0} \end{array}$$

where $AN(Q)$ is the set of free ambient names in Q, $GN(Q)$ is the set of free group names in Q, and $GN(G)$ is the set of free group names in G.

Fig. 3: Structural congruence

The reduction rules are reported in Figure 4. Remark, w.r.t. the original Mobile Ambients, the synchronous output, the missing open, and the rule for the to action, similar to the go primitive of Dπ or to the "migrate" instructions for strong code mobility in software agents.

A process executing a $\mathsf{to}\, m$ action moves between sibling ambients: from an ambient n, where it is initially located, to a (different) ambient of name m that is a sibling of n. It thus crosses two boundaries in one step; the boundaries are however at the same level, so that – differently from moving upward or downward – the process does not change its nesting level. A process executing an $\mathsf{in}\, m$ or $\mathsf{out}\, m$ action drives its enclosing ambient respectively into or out of the ambient m, as usual.

Basic reduction rules:

$$\begin{array}{lrcl}
\text{(R-in)} & n[\,\mathsf{in}\, m\,.\,P \,|\, Q\,] \,|\, m[R] & \rightarrow & m[\,n[\,P \,|\, Q\,] \,|\, R\,] \\
\text{(R-out)} & m[\,n[\,\mathsf{out}\, m\,.\,P \,|\, Q\,] \,|\, R\,] & \rightarrow & n[\,P \,|\, Q\,] \,|\, m[R] \\
\text{(R-to)} & n[\mathsf{to}\, m\,.\,P \,|\, Q] \,|\, m[R] & \rightarrow & n[Q] \,|\, m[P \,|\, R] \\
\text{(R-comm)} & (x\colon W)P \,|\, \langle M \rangle Q & \rightarrow & P\{x := M\} \,|\, Q
\end{array}$$

Structural reduction rules:

$$\begin{array}{lrcl}
\text{(R-in)} & P \rightarrow Q & \Rightarrow & P \,|\, R \rightarrow Q \,|\, R \\
\text{(R-amb)} & P \rightarrow Q & \Rightarrow & n[P] \rightarrow n[Q] \\
\text{(R-}\equiv\text{)} & P' \equiv P',\; P \rightarrow Q,\; Q \equiv Q' & \Rightarrow & P' \rightarrow Q' \\
\text{(R-}\nu\text{)} & P \rightarrow Q & \Rightarrow & (\nu n\colon \mathsf{amb}(g))P \rightarrow (\nu n\colon \mathsf{amb}(g))Q \\
\text{(R-}\nu\text{-group)} & P \rightarrow Q & \Rightarrow & (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})P \rightarrow (\nu\{\overrightarrow{\mathbf{g}:\mathbf{G}}\}_{(k)})Q
\end{array}$$

Fig. 4: Reduction

3 Types and environments

The syntax of types is shown in Figure 5. Term types are divided into process types and message types, corresponding to the two main syntactic categories of terms. Message types, in turn, may be ambient types or capability types but cannot be process types, since messages cannot be whole processes (the calculus is not higher-order).

For the notion of principal typing to be well defined in any case, a bottom message type $\bot_W$ is introduced; not present in [8], it corresponds to an indefinite message type.

The only components of term types, which (apart from $\bot_W$) are of the form $\mathsf{amb}(g)$, $\mathsf{cap}(g_1, g_2)$ or $\mathsf{pr}(g)$, are *group names* g; these, in turn, may be typed with *group types*, which are of the form $\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$, where $\mathscr{S}, \mathscr{C}$ and $\mathscr{E}$ are sets of group names, and T is a *communication type*.

The usage of groups (a rather unfortunate name from a purely mathematical perspective) in mobility types is a standard technique for avoiding dependencies between types and values, which would prevent or make awkward the otherwise simple typings of meaningful terms.

Intuitively, a group name g singles out a set of ambients with “the same properties", and the set of processes that can stay within such ambients: $\mathsf{amb}(g)$ is the type of ambients of group g, or g-ambients; $\mathsf{pr}(g)$ is the type of processes of group g, or g-processes; g-ambients may only contain g-processes, and g-processes may only stay in g-ambients. The type $\mathsf{cap}(g1, g2)$ is the type of an action that may be executed by a g_2-process (in a g_2-ambient) and whose continuation is a g_1-process (in a g_1-ambient).

We recall the intuitive meaning of a group type, i.e., the meaning of the assumption $g{:}\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$:

- $\mathscr{S}$ is the set of ambient groups where the ambients of group g can stay;
- $\mathscr{C}$ is the set of ambient groups that g-ambients can cross, i.e., those that they may be driven into or out of, respectively, by in or out actions; it must be $\mathscr{C} \subseteq \mathscr{S}$;
- $\mathscr{E}$ is the set of ambients that (naked) g-processes can "enter": more precisely, those to which a g-process may send its continuation by means of a to action (it is empty if naked g-processes are immobile);
- T is the (fixed) communication type (or topics of conversation) within g-ambients.

To handle the condition on the $\mathscr{S}$ components of group types in the (OUT) rule, group *pre-types* containing *starred* group names are introduced. Group pre-types $\mathsf{gr}(\mathcal{S}, \mathscr{C}, \mathscr{E}, T)$ differ from types in that the first component $\mathcal{S}$ may contain both group names and starred group names.

The meaning of a group pre-type assumption $g{:}\mathsf{gr}(\mathcal{S}, \mathscr{C}, \mathscr{E}, T)$ is the same as the one of a group type assumption, with the difference that every $g_1^* \in \mathcal{S}$ denotes all the ambient groups where g_1-ambients may stay, i.e., it denotes all the elements of the set $\mathcal{S}_1$ such that $g_1{:}\mathsf{gr}(\mathcal{S}_1, \_, \_, \_)$ (if such an assumption exists in the environment). Almost all the typing rules actually concern group pre-types; types proper without starred groups are only obtained in the final phase by explicitly unfolding the meaning of star.

If $\mathcal{G} = \mathsf{gr}(\mathcal{S}, \mathscr{C}, \mathscr{E}, T)$ is a group pre-type, we write $\mathcal{S}(\mathcal{G})$, $\mathscr{C}(\mathcal{G})$, $\mathscr{E}(\mathcal{G})$, $T(\mathcal{G})$ to respectively denote the components $\mathcal{S}$, $\mathscr{C}$, $\mathscr{E}$, T of $\mathcal{G}$; an analogous notation may of course be used for group types.

Communication types are components of group (pre-)types, but they are not term types; T is the type of communication allowed within an ambient, it may consist of a message type but it may also be the absence of communication, denoted by shh. Observe that the newly introduced indefinite message type $\perp_W$ is distinct from shh. Differently from [8], in order to stress the fact that a communication type composed of a message type is in a different category from its component (and so it is a different data type in the metalanguage), a unary type constructor com is used.

An *environment* (or a *pre-environment*) E consists of two components: a *group (pre-)environment* Γ and a *variable (and ambient) environment* Δ, as defined by the following syntax:

$$\mathrm{E} ::= \Gamma; \Delta \qquad \Gamma ::= \varnothing \mid \Gamma,\ g{:}G \text{ (or } g{:}\mathcal{G}) \qquad \Delta ::= \varnothing \mid \Delta,\ \xi{:}W$$

where ξ is a variable or an ambient name. In the sequel all the notions introduced

Type	$::=$		term type
		W	message type
		Pr	process type
W	$::=$		message type
		$\bot_W$	any message
		$\mathsf{amb}(g)$	ambient type: ambients of group g
		$\mathsf{cap}(g_1, g_2)$	capability type: capabilities that, prefixed to a process of type $\mathsf{pr}(g_1)$, turn it into a process of type $\mathsf{pr}(g_2)$
Pr	$::=$	$\mathsf{pr}(g)$	process type: processes that can stay in ambients of group g
G	$::=$	$\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T)$ with $\mathscr{C} \subseteq \mathscr{S}$	group type
T	$::=$		communication type
		shh	no communication
		$\mathsf{com}(W)$	communication of messages of type W
		$g, h, \ldots$	groups
		$\mathscr{S}, \mathscr{C}, \mathscr{E}, \ldots$	sets of groups; $\mathbb{G}$ is the universal set of groups

Fig. 5: Types

$\mathcal{G}$	$::=$	$\mathsf{gr}(\mathcal{S}, \mathscr{C}, \mathscr{E}, T)$	group pre-type
		$g, h, \ldots$	groups
		$g^*, h^*, \ldots$	starred groups
		$\mathcal{S}$	set of groups and starred groups;

Fig. 6: Group pre-types

for environments will also be implicitly considered, unless stated otherwise, as defined for pre-environments, with the obvious substitution of types with pre-types.

The *domain* of an environment is $\mathit{Dom}(\mathrm{E}) = \mathit{Dom}(\Gamma) \cup \mathit{Dom}(\Delta)$, where:

$$\mathit{Dom}(\varnothing) = \varnothing \quad \mathit{Dom}(\Gamma, g{:}G) = \mathit{Dom}(\Gamma) \cup \{g\} \quad \mathit{Dom}(\Delta, \xi{:}W) = \mathit{Dom}(\Delta) \cup \{\xi\}$$

$\mathit{GN}(G)$ denotes the set of all group names occurring in a group type G, and $\mathit{GN}(\mathrm{E})$ denotes the set of all group names occurring in E, not only in $\mathit{Dom}(\Gamma)$ but also in the components of the types in E. Environments are considered as sets of statements, therefore modulo permutations.

A variable environment Δ is *well-formed* if for each $\xi \in \mathit{Dom}(\Delta)$ there is exactly one type associated to it in Δ, i.e., there cannot exist $\xi{:}W_1,\ \xi{:}W_2 \in \Delta$ with W_1 different from W_2. We assume that all variable environments are well-formed.

Analogously, a group environment Γ is *well-formed* if for each $g \in \mathit{Dom}(\Gamma)$ there is exactly one group type G associated to it in Γ. Of course, only well-formed group environments are allowed in a typing judgement, but (potentially) non-well-formed group environments are used by the type inference procedure.

We use the standard notation $\Delta,\ \xi{:}W$ to denote a variable environment containing a statement $\xi{:}W$, assuming that $\xi \notin \mathit{Dom}(\Delta)$; analogously for $\Gamma,\ g{:}G$; etc.

Two variable environments Δ_1, Δ_2 are *compatible*, written $\Delta_1 \sharp \Delta_2$, if $\Delta_1 \cup \Delta_2$ is a well-formed environment. Two group environments Γ_1 and Γ_2 are *compatible*, also written $\Gamma_1 \sharp \Gamma_2$, if:

$$(g : \mathsf{gr}(\mathscr{S}_1, \mathscr{C}_1, \mathscr{E}_1, T_1)) \in \Gamma_1,\ (g : \mathsf{gr}(\mathscr{S}_2, \mathscr{C}_2, \mathscr{E}_2, T_2)) \in \Gamma_2 \Rightarrow T_1 = T_2.$$

We will write $\Gamma_1 \sharp \Gamma_2 \sharp \Gamma_3$ to indicate that Γ_1, Γ_2, Γ_3 are pairwise compatible; etc. The *G-union* of two *compatible* environments Γ_1 and Γ_2 is the (well-formed) environment:

$$\Gamma_1 \Cup \Gamma_2 = (\Gamma_1 \cup \Gamma_2 - \{(g{:}G) | g \in Dom(\Gamma_1) \cap Dom(\Gamma_2)\}) \cup \Gamma', \text{ where:}$$
$$\Gamma' = \{g{:}\mathsf{gr}(\mathscr{S}_1 \cup \mathscr{S}_2, \mathscr{C}_1 \cup \mathscr{C}_2, \mathscr{E}_1 \cup \mathscr{E}_2, T) \,|\, g{:}\mathsf{gr}(\mathscr{S}_1, \mathscr{C}_1, \mathscr{E}_1, T) \in \Gamma_1 \,\&\, g{:}\mathsf{gr}(\mathscr{S}_2, \mathscr{C}_2, \mathscr{E}_2, T) \in \Gamma_2\}$$

A partial order is naturally defined on communication and group types:

$$\bot_W \leq W \qquad \mathsf{shh} \leq \mathsf{com}(W) \qquad W \leq W' \Rightarrow \mathsf{com}(W) \leq \mathsf{com}(W')$$
$$\mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, T) \leq \mathsf{gr}(\mathscr{S}', \mathscr{C}', \mathscr{E}', T') \text{ if } \mathscr{S} \subseteq \mathscr{S}', \mathscr{C} \subseteq \mathscr{C}', \mathscr{E} \subseteq \mathscr{E}', T \leq T'$$

As usual, the intuitive meaning of the statement $T \leq T'$ is that T is a subtype of T', i.e., a more specialized type of communication; in the case of group types, the relationship $G \leq G'$ means that G is more restrictive than G', since the sets of ambients where one is allowed to stay, or which one is allowed to cross, etc., are smaller.

The order is extended monotonically to environments via set inclusion:

$$\begin{array}{rl} \Gamma \leq \Gamma' & \text{if } \forall\, (g{:}G) \in \Gamma \,.\, \exists\, (g{:}G') \in \Gamma' \,.\, G \leq G' \\ \Delta \leq \Delta' & \text{if } \forall (\xi{:}W) \in \Delta \,.\, \exists\, (\xi{:}W') \in \Delta' \,.\, W \leq W' \\ \Gamma; \Delta \leq \Gamma'; \Delta' & \text{if } \Gamma \leq \Gamma' \text{ and } \Delta \leq \Delta' \end{array}$$

Again, the meaning of the environment ordering $\mathrm{E} \leq \mathrm{E}'$ is that E is a more constraining set of assumptions than E'.

A notion of principal typing in agreement with [12] is then definable: a typing judgment $\mathrm{E} \vdash_{\mathrm{M}^3} P\colon \mathsf{pr}(g)$ is *principal* if for every derivable typing $\mathrm{E}' \vdash_{\mathrm{M}^3} P'\colon \mathsf{pr}(g')$ such that $|P'| = |P|$, with P and P' free from group restrictions, there exists a substitution σ from group names to group names such that $\sigma(g) = g'$ and $\sigma(\mathrm{E}) \leq \mathrm{E}'$. Analogously one can define the notion of a principal typing $\mathrm{E} \vdash$ *trm*$:$$\mathit{Type}$ for a generic term (i.e., not only for processes).

The condition that P and P' do not contain group restrictions does not go against the standard definition of principal typing, since group restrictions are simply obtained by abstracting w.r.t. group names through the rule GRPRES in a final typing step.

Observe that since term types are composed of mere group names, all the types assignable to a given term on the r.h.s. of the turnstile are identical modulo a renaming of group names. What characterizes a typing judgment as principal is therefore that the environment E is the most restrictive set of assumptions allowing to type a given (raw) term trm, i.e., the minimal set of permissions needed for the term to be well typed.

4 Typing rules

The original type assignment rules for M^3 are given in Figure 7 following the syntax presented in the previous section.

As observed in the introduction, the system does not provide an algorithm for inferring the (principal) typing of a typable term, because of the implicit weakening apparent in the rules ENV, of course coupled with the fact that the different premisses of a single rule contain equal environments.

The phenomenon is well known both in proof theory and in type systems for functional languages; so is the fact that a more algorithmic system may in such cases be obtained by delaying the application of weakening till the end of the inference process.

The new set of typing rules oriented to principal typing is given in Figure 8. In the rules (ENV), (NULL) and in those for capabilities the environment consists of the minimal mobility assumptions; in the rest of the system, the different premisses of a rule have different environments, which are combined into a new minimal environment in the conclusion.

The reason why the assumptions on the communication type are not minimal, as apparent from the presence of a generic T instead of shh in the rules for capabilities, is that such minimality, though theoretically more appropriate to the intended framework, would make the logic programming implementation more complex without need. The minimal communication type is instead obtained, only in the Prolog program, automatically through unification and then through one final step where uninstantiated communication and message types are instantiated to their respective minima, as will be explained below.

The rules (IN) and (OUT) state that a process exercising an $\mathsf{in}/\mathsf{out}\ \xi$ capability does not change its group g_2 since it does not change its enclosing g_2-ambient; the minimal mobility assumption is that g_2-ambients are (only) allowed to cross g_1-ambients (like ξ), and are therefore also allowed to stay (only) in g_1-ambients (here and in the following observe that if ξ is an input variable, by the time the action will be executed the variable will have been replaced by an ambient name).

In the old system the rules (IN) and (OUT) had several premisses. However, premisses of the forms $\mathrm{E} \vdash_{\mathrm{M}^3} g{:}G$ and $\mathrm{E} \vdash_{\mathrm{M}^3} M{:}\mathsf{amb}(g)$ may in turn only be derived through the (ENV) rules, i.e., from the fact that respectively $g{:}G$ or $\xi{:}\mathsf{amb}(g)$ is in E, with M specialized into ξ. As usual, by combining the two steps into a single rule and by everywhere replacing membership conditions with the minimal sets satisfying those conditions, the rules become axioms (i.e., rules with no premisses).

For example, in the rule (IN) the conditions $(g_2{:}G_2) \in \Gamma$ and $(\xi{:}\mathsf{amb}(g_1)) \in \Delta$ become $\mathrm{E} \equiv g_2{:}G_2;\ \xi{:}\mathsf{amb}(g_1)$; the condition $g_1 {\in} \mathscr{C}(G_2)$ becomes $\mathscr{C}(G_2) \equiv \{g_1\}$,

$$\frac{g{:}G \in \Gamma}{\Gamma;\Delta \vdash_{M^3} g : G}\ (\textsc{Env}_\Gamma) \qquad \frac{\xi{:}W \in \Delta}{\Gamma;\Delta \vdash_{M^3} \xi : W}\ (\textsc{Env}_\Delta) \qquad \frac{}{E \vdash_{M^3} 0 : \mathsf{pr}(g)}\ (\textsc{Null})$$

$$\frac{E \vdash_{M^3} g_2 : G_2 \quad E \vdash_{M^3} M : \mathsf{amb}(g_1) \quad g_1 \in \mathscr{C}(G_2)}{E \vdash_{M^3} \mathsf{in}\ M : \mathsf{cap}(g_2, g_2)}\ (\textsc{In})$$

$$\frac{E \vdash_{M^3} g_1 : G_1 \quad E \vdash_{M^3} g_2 : G_2 \quad E \vdash_{M^3} M : \mathsf{amb}(g_1) \quad g_1 \in \mathscr{C}(G_2) \quad \mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)}{E \vdash_{M^3} \mathsf{out}\ M : \mathsf{cap}(g_2, g_2)}\ (\textsc{Out})$$

$$\frac{E \vdash_{M^3} g_2 : G_2 \quad E \vdash_{M^3} M : \mathsf{amb}(g_1) \quad g_1 \in \mathscr{E}(G_2)}{E \vdash_{M^3} \mathsf{to}\ M : \mathsf{cap}(g_1, g_2)}\ (\textsc{To})$$

$$\frac{E \vdash_{M^3} M : \mathsf{cap}(g_3, g_2) \quad E \vdash_{M^3} N : \mathsf{cap}(g_1, g_3)}{E \vdash_{M^3} M.N : \mathsf{cap}(g_1, g_2)}\ (\textsc{Path})$$

$$\frac{E \vdash_{M^3} M : \mathsf{cap}(g_1, g_2) \quad E \vdash_{M^3} P : \mathsf{pr}(g_1)}{E \vdash_{M^3} M.P : \mathsf{pr}(g_2)}\ (\textsc{Prefix})$$

$$\frac{\Gamma;\Delta,\ x{:}W \vdash_{M^3} P : \mathsf{pr}(g) \quad \Gamma;\Delta \vdash_{M^3} g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, \mathsf{com}(W))}{\Gamma;\Delta \vdash_{M^3} (x{:}W)P : \mathsf{pr}(g)}\ (\textsc{Input})$$

$$\frac{E \vdash_{M^3} P : \mathsf{pr}(g) \quad E \vdash_{M^3} M : W \quad E \vdash_{M^3} g : \mathsf{gr}(\mathscr{S}, \mathscr{C}, \mathscr{E}, \mathsf{com}(W))}{E \vdash_{M^3} \langle M\rangle P : \mathsf{pr}(g)}\ (\textsc{Output})$$

$$\frac{E \vdash_{M^3} P : \mathsf{pr}(g) \quad E \vdash_{M^3} M : \mathsf{amb}(g) \quad E \vdash_{M^3} g : G \quad g' \in \mathscr{S}(G)}{E \vdash_{M^3} M[P] : \mathsf{pr}(g')}\ (\textsc{Amb})$$

$$\frac{E \vdash_{M^3} P : \mathsf{pr}(g) \quad E \vdash_{M^3} Q : \mathsf{pr}(g)}{E \vdash_{M^3} P \,|\, Q : \mathsf{pr}(g)}\ (\textsc{Par}) \qquad \frac{E \vdash_{M^3} P : \mathsf{pr}(g)}{E \vdash_{M^3} !\, P : \mathsf{pr}(g)}\ (\textsc{Repl})$$

$$\frac{\Gamma;\Delta, m : \mathsf{amb}(g') \vdash_{M^3} P : \mathsf{pr}(g)}{\Gamma;\Delta \vdash_{M^3} (\nu m{:}\, \mathsf{amb}(g'))P : \mathsf{pr}(g)}\ (\textsc{AmbRes})$$

$$\frac{\Gamma, g_1{:}G_1, \ldots, g_k{:}G_k;\Delta \vdash_{M^3} P : \mathsf{pr}(g) \quad g_i \notin GN(\Gamma \cup \Delta) \quad g_i \neq g \quad (1 \leq i \leq k)}{\Gamma;\Delta \vdash_{M^3} (\nu\{g_1{:}\, G_1, \ldots, g_k{:}\, G_k\})P : \mathsf{pr}(g)}\ (\textsc{GrpRes})$$

Fig. 7: Typing rules

and the group-type well-formedness condition $\mathscr{C} \subseteq \mathscr{S}$ becomes $\mathcal{S} \equiv \{g_1\}$.

In the case of (OUT), moreover, the g_2-ambient – being driven out of ξ – becomes a sibling of ξ, and must therefore have permission to stay wherever a $g1$-ambient like ξ is allowed to stay; this is expressed by the presence of g_1^* beside g_1 in the $\mathcal{S}$ component of the group type of g_2 (instead of the condition $\mathscr{S}(G_1) \subseteq \mathscr{S}(G_2)$ in the non-algorithmic system of Figure 7).

$$\frac{}{\varnothing;\ \xi{:}W \vdash_\circ \xi{:}W}\ (\textsc{Env}) \qquad \frac{}{\varnothing;\varnothing \vdash_\circ \mathsf{0} : \mathsf{pr}(g)}\ (\textsc{Null})$$

$$\frac{}{g_2 : \mathsf{gr}(\{g_1\},\{g_1\},\varnothing,T);\ \ \xi{:}\ \mathsf{amb}(g_1) \vdash_\circ \mathsf{in}\ \xi : \mathsf{cap}(g_2,g_2)}\ (\textsc{In})$$

$$\frac{}{g_2 : \mathsf{gr}(\{g_1,g_1^*\},\{g_1\},\varnothing,T);\ \ \xi{:}\ \mathsf{amb}(g_1) \vdash_\circ \mathsf{out}\ \xi : \mathsf{cap}(g_2,g_2)}\ (\textsc{Out})$$

$$\frac{}{g_2 : \mathsf{gr}(\varnothing,\varnothing,\{g_1\},T);\ \ \xi{:}\ \mathsf{amb}(g_1) \vdash_\circ \mathsf{to}\ \xi : \mathsf{cap}(g_1,g_2)}\ (\textsc{To})$$

$$\frac{\Gamma_1;\Delta_1 \vdash_\circ M : \mathsf{cap}(g_3,g_2) \quad \Gamma_2;\Delta_2 \vdash_\circ N : \mathsf{cap}(g_1,g_3) \quad \Gamma_1\sharp\Gamma_2 \quad \Delta_1\sharp\Delta_2}{\Gamma_1 \Cup \Gamma_2;\ \Delta_1 \cup \Delta_2 \vdash_\circ M.N : \mathsf{cap}(g_1,g_2)}\ (\textsc{Path})$$

$$\frac{\Gamma_1;\Delta_1 \vdash_\circ M : \mathsf{cap}(g_1,g_2) \quad \Gamma_2;\Delta_2 \vdash_\circ P : \mathsf{pr}(g_1) \quad \Gamma_1\sharp\Gamma_2 \quad \Delta_1\sharp\Delta_2}{\Gamma_1 \Cup \Gamma_2;\ \Delta_1 \cup \Delta_2 \vdash_\circ M.P : \mathsf{pr}(g_2)}\ (\textsc{Prefix})$$

$$\frac{\Gamma;\ \Delta \vdash_\circ P : \mathsf{pr}(g) \quad \Delta\sharp\{x{:}W\} \quad \Gamma\sharp\Gamma_0 \text{ where } \Gamma_0 \equiv \{g : \mathsf{gr}(\varnothing,\varnothing,\varnothing,\mathsf{com}(W))\}}{\Gamma \Cup \Gamma_0;\ \Delta - \{x{:}W\} \vdash_\circ (x{:}W)P{:}\ \mathsf{pr}(g))}\ (\textsc{Input})$$

$$\frac{\Gamma_1;\Delta_1 \vdash_\circ P : \mathsf{pr}(g) \ \ \Gamma_2;\Delta_2 \vdash_\circ M : W \quad \Gamma_1\sharp\Gamma_2\sharp\Gamma_0 \quad \Delta_1\sharp\Delta_2}{\Gamma_1 \Cup \Gamma_2 \Cup \Gamma_0;\ \Delta_1 \cup \Delta_2 \vdash_\circ \langle M\rangle P : \mathsf{pr}(g)}\ (\textsc{Output})$$
$$\text{where } \Gamma_0 \equiv \{g : \mathsf{gr}(\varnothing,\varnothing,\varnothing,\mathsf{com}(W))\}$$

$$\frac{\Gamma;\Delta,\ \xi{:}\ \mathsf{amb}(g) \vdash_\circ P : \mathsf{pr}(g) \quad \Gamma\sharp\Gamma_0 \text{ where } \Gamma_0 \equiv \{g : \mathsf{gr}(\{g'\},\varnothing,\varnothing,T)\}}{\Gamma \Cup \Gamma_0;\ \Delta,\ \xi{:}\ \mathsf{amb}(g) \vdash_\circ \xi[P] : \mathsf{pr}(g')}\ (\textsc{Amb})$$

$$\frac{\Gamma_1;\Delta_1 \vdash_\circ P : \mathsf{pr}(g) \ \ \Gamma_2;\Delta_2 \vdash_\circ Q : \mathsf{pr}(g) \quad \Gamma_1\sharp\Gamma_2 \quad \Delta_1\sharp\Delta_2}{\Gamma_1 \Cup \Gamma_2;\ \Delta_1 \cup \Delta_2 \vdash_\circ P \,|\, Q : \mathsf{pr}(g)}\ (\textsc{Par})$$

$$\frac{\Gamma;\Delta \vdash_\circ P : \mathsf{pr}(g)}{\Gamma;\Delta \vdash_\circ !\, P : \mathsf{pr}(g)}\ (\textsc{Repl})$$

$$\frac{\Gamma;\Delta \vdash_\circ P : \mathsf{pr}(g) \quad \Delta\sharp\{m{:}\mathsf{amb}(g')\}}{\Gamma;\ \Delta - \{m{:}\mathsf{amb}(g')\} \vdash_\circ (\nu m{:}g')P : \mathsf{pr}(g)}\ (\textsc{AmbRes})$$

Fig. 8: Rules for principal typing

The rule (TO) states that the action to ξ, if performed by a process of group g_2 (in a g_2-ambient), leaves as continuation a process of group g_1, if g_1 is the group of ξ; the minimal assumption is that (naked) *g_2-processes* are only allowed to go (i.e., send their continuations) in-to g_1-ambients.

The rules (PATH) and (PREFIX) state that the sequential composition of capabilities requires the union of the mobility assumptions separately needed for the single capabilities (where in case of different assumptions on the type of a same group g, the unions of the homologous components $\mathcal{S}$, $\mathscr{C}$, $\mathscr{E}$ of the different types are performed, as specified by the definition of the operation $\Cup$). The rule (PAR) states the same thing for parallel composition.

The central rule (AMB) is quite standard: it requires that in a term $\xi[P]$ the ambient ξ and its content P be of the same group, while the process $\xi[P]$, being a completely passive object, unable both to communicate and to move other ambients, may in turn stay in any ambient, i.e., it may be of any group g'. The minimal requirement is that g-ambients (like ξ) are allowed to stay in g'-ambients, as expressed by the addition of g' to the $\mathcal{S}$ component of the type of g.

The remaining rules are rather straightforward; only observe that in the "abstractions", i.e., in (INPUT) and (AMBRES), a set-theoretic difference is needed in the conclusion, differently from the homonymous rules in Figure 7, because of the minimality of the environment in the premiss. For example, in (INPUT) the statement $x{:}W$ cannot be assumed to be always present in the environment that types P: if x does not occur in P, no such statement occurs in the environment.

It should be clear from the above that in this system too, as in the one of Figure 7, more typings than just principal typings are derivable, since the metavariables T and W can be respectively any communication type and any message type, not just the minimal ones. However, if one considers the type system as a logical theory and derivable typing judgments as theorems of the theory, then T and W become logical variables (of distinct sorts) which may occur free in the typing of a term; free occurrences of T and W, being implicitly universally quantified, can be respectively instantiated to any communication type and any message type, in particular to the minimal ones. Thus, instantiating T to shh and W to $\bot_W$ yields the minimal typing.

This is exactly what happens in the implementation, where T and W are Prolog variables that may be left uninstantiated by the type-inference algorithm, in which case they are respectively bound to the atoms `shh` and `w` (for $\bot_W$) at the end. For example:

```
finish(T):- var(T),!,T=shh.
finish(_).
```

Also, group pre-types have to be transformed into group types by unfolding all the information conveyed by starred group names, with the final elimination of the starred names themselves. This is done by the *closure* function, which computes the least fixed point of the transformation

$$\Gamma \mapsto \{g : \mathsf{gr}(\mathcal{S}(\mathcal{G}) \cup (\bigcup_{g' \in {}^*(\mathcal{S})\ \&\ g':\mathcal{G}' \in \Gamma} \mathcal{S}(\mathcal{G}')), \mathscr{C}(\mathcal{G}), \mathscr{E}(\mathcal{G}), T(\mathcal{G})) \mid g{:}\mathcal{G} \in \Gamma\}$$

where ${}^*(\mathcal{S}) = \{g \mid g^* \in \mathcal{S}\}$, and finally erases all the starred groups.

We may therefore introduce the typing judgment proper $\Gamma; \Delta \vdash_\bullet$ *trm*:*Type*, with Γ environment proper (i.e., without stars), through the following rule:

$$\frac{\Gamma'; \Delta \vdash_\circ \textit{trm}{:}\textit{Type}}{\textit{closure}(\Gamma'); \Delta \vdash_\bullet \textit{trm}{:}\textit{Type}} \quad (\text{CLOSE})$$

where it must be reminded that trm is either a process P or a message M, and correspondingly Type is either a process type or a message type.

The typing relation $\vdash_\bullet$ can be shown to correspond to the original relation $\vdash_{\mathrm{M}^3}$ in the following sense:

- (soundness) if $\Gamma; \Delta \vdash_\bullet \mathit{trm}{:}\mathit{Type}$ holds, then also $\Gamma; \Delta \vdash_{\mathrm{M}^3} \mathit{trm}{:}\mathit{Type}$ holds;
- (completeness) if $\Gamma; \Delta \vdash_{\mathrm{M}^3} \mathit{trm}{:}\mathit{Type}$ holds, then there exists a term trm' structurally equivalent to trm (i.e., $\mathit{trm}' \equiv \mathit{trm}$) of the form

  $$\mathit{trm}' = (\nu\{g_1{:}G_1, \ldots, g_k{:}G_k\})\mathit{trm}_0$$

  where trm_0 is free from group restrictions, and there exist Γ' and Δ' such that $\Gamma'; \Delta' \vdash_{\mathrm{M}^3} \mathit{trm}_0{:}\mathit{Type}$ holds, with $\Gamma' \leq \Gamma, g_1{:}G_1, \ldots, g_k{:}G_k$ and $\Delta' \leq \Delta$.

Thus any typing derivable in the original system may be obtained from one in our new system by subsumption on the environment.

Alternatively, one can add to the new type system a third kind of judgment on top of $\vdash_\circ$ and $\vdash_\bullet$, introduced by a weakening rule and also defined by a group restriction rule identical to the one in $\vdash_{\mathrm{M}^3}$, but now applicable only at the top level:

$$\frac{\mathrm{E} \vdash_\bullet \mathit{trm} : \mathit{Type} \quad \mathrm{E} \leq \mathrm{E}'}{\mathrm{E}' \vdash \mathit{trm} : \mathit{Type}}\ (\textsc{Weak})$$

$$\frac{\Gamma, g_1{:}G_1, \ldots, g_k{:}G_k; \Delta \vdash P{:}\mathsf{pr}(g) \quad g_i \notin GN(\Gamma \cup \Delta) \quad g_i \neq g \quad (1 \leq i \leq k)}{\Gamma; \Delta \vdash (\nu\{g_1{:}\, G_1, \ldots, g_k{:}\, G_k\})P : \mathsf{pr}(g)}\ (\textsc{GrpRes})$$

With this definition, of course soundness and completeness simply become the equivalence between the two systems:

$$\mathrm{E} \vdash \mathit{trm} : \mathit{Type} \text{ holds if and only if } \mathrm{E} \vdash_{\mathrm{M}^3} \mathit{trm} : \mathit{Type} \text{ holds}$$

Observe, however, that if one is only interested in principal typings then only the relation $\vdash_\bullet$ is needed: it is immediate to see, from the above, that the principal typings of $\vdash_{\mathrm{M}^3}$ and $\vdash_\bullet$ coincide.

5 The Prolog implementation

The Prolog implementation only concerns the typing $\vdash_\bullet$; in particular, the program takes as input a raw term trm and returns the principal typing $\mathrm{E} \vdash_\bullet \mathit{trm} : \mathit{Type}$ such that $|\mathit{trm}| = \mathsf{trm}$.

Therefore, if the judgment $\mathrm{E} \vdash_\bullet \mathit{trm}{:}\mathit{Type}$ is derivable, then the Prolog program, fed with the raw term $|\mathit{trm}|$, does not in general return the original typing, nor simply a typing with a possibly "less restrictive" environment. Rather, it returns a typing $\mathrm{E}_0 \vdash_\bullet \mathit{trm}_0{:}\mathit{Type}_0$ such that, for a substitution σ, one has $\sigma(\mathrm{E}_0) \leq \mathrm{E}$, $\sigma(\mathit{trm}_0) = \mathit{trm}$ and $\sigma(\mathit{Type}_0) = \mathit{Type}$.

As remarked in section 4, this depends on the fact that when the rules are translated into Prolog clauses, all the uninstantiated logical variables are initially fresh distinct variables, while in the formal deduction of a $\vdash_{\bullet}$-judgment one may start, for example, with identical names in different leaves, i.e., with logical variables already instantiated by some σ; thus one eventually obtains a typing which is (greater than) a σ-instance of the principal typing.

A preliminary step to coding in Prolog the new typing rules is the choice of suitable representations for the group and variable environments Γ and Δ. In the working prototype they are simply implemented as lists: in particular, they are implemented as lists terminated by an unbound logical variable. A new element may then be added at the end of the list, after checking that it is not already present, by "imperatively" modifying the list through the binding of the logical variable (a standard technique of logic programming). The same kind of representation is adopted for all the set components of types in the environment.

Object-language variables and ambient names are represented by Prolog constants. Group names are represented by Prolog variables, so that the unification performed by the interpreter may be exploited when two different group names have to be equated. On the other hand, since group names are object-language entities and not actual meta-variables, the need often arises of testing their equality: in such case one must of course resort to the extralogical primitives of Prolog, like the operator `==`.

Though types both in their original form and in the new one do not contain type variables, in the program there is the natural use of type metavariables, represented by Prolog variables.

Every typing rule translates almost literally into a Prolog clause; the only difference is that the construction of the environment in the conclusion and the compatibility check between the environments in the premisses are obviously combined into a single procedure `sumunion` that tries to build the conclusion by possibly "further instantiating" the component environments.

Such operation, however, cannot be automatically performed by unification, since the set-theoretic union of two group environments may produce a non-well-formed group environment Γ where group types with different communication types are assigned to a same group name; if such communication types are unifiable, the unifier may in turn equate other groups having different communication types, and so on.

The *completion* of the environment has therefore to be explicitly programmed through an iteration that at each step tries to unify the distinct communication types generated by the previous unification step. Since the number of group names in any given environment is finite, the iteration is guaranteed to terminate, either with a failure (non-unifiable types) or with a success (no "critical type pairs" left). The relevant Prolog clause is:

```
sumunion(Env1,Env2,GEnv1,GEnv2,GEnv):-
  uniongenv(GEnv1,GEnv2,GEnvU),
  unionenv(Env1,Env2),
  comp(GEnvU,GEnv).
```

The Prolog procedure `comp`, which contains an imperative loop, tries to transform a possibly non-well-formed environment into a well-formed one, by unifying communication types and joining group sets. The unions of set components are incrementally performed at each step, together with the unification of communication types, through the clause:

```
unigt(gr(S1,C1,E1,T1), gr(S2,C2,E2,T2)):-
  union(S1,S2), union(C1,C2), union(E1,E2), uniw(T1,T2).
```

The unification between $T1$ and $T2$ has also to be done by an explicit clause, since it must check whether a new identification of group names is produced, which – being the generator of a new pair of potentially conflicting communication types – requires repeating the process:

```
uniw(T1,T2):- var(T1),!,T1=T2.
uniw(T1,T2):- var(T2),!,T1=T2.
uniw(T1,T2):- T1==T2,!.
uniw(T,T):- retract(repeating(_)),assert(repeating(true)).
```

Once all the auxiliary "non-logic" procedures have been defined, the translation of the typing rules into Prolog clauses is immediate: it suffices to eliminate all the explicit compatibility premisses and to replace the $\Cup$ operator with the procedure `sumunion`, which in trying to build the environment $\Gamma \Cup \Gamma'$ implicitly tries to instantiate the two environments Γ and Γ' in such a way that $\Gamma\sharp\Gamma'$ holds, as described above. On the other hand, the eliminated premisses of the form $\Delta\sharp\Delta'$ are automatically checked by Prolog unification in the `unionenv` clause, also invoked by `sumunion`.

For example, the rule for parallel composition trivially becomes:

```
typing(GEnv, Env, P1 par P2, pr(G)):-
  typing(GEnv1, Env1, P1, pr(G)),
  typing(GEnv2, Env, P2, pr(G)),
  sumunion(Env1,Env,GEnv1,GEnv2,GEnv).
```

Analogous is the prefix rule for sequential composition:

```
typing(GEnv, Env, M dot P, pr(G2)):-
  typing(GEnv1, Env1, P, pr(G1)),
  typing(GEnv2,Env, M, cap(G1,G2)),
  sumunion(Env1,Env,GEnv1,GEnv2,GEnv).
```

Observe that the imperative procedure `sumunion` stands simultaneously for the relation $\sharp$ and the operation $\Cup$.

In the axiom-rules for capabilities, variables and null process, one only has to remember that in the chosen representation empty sets are represented by unbound Prolog variables:

```
typing(_,_, 0, pr(_)).
typing(_,[X:W|_], X, W) :- atom(X).
typing(GEnv, Env, to(M), cap(G1,G2)):-
  GEnv = [G2:gr(_,_,[G1|_],_)|_], Env = [M:amb(G1)|_].
typing(GEnv, Env, in(M), cap(G2,G2)):-
  GEnv = [G2:gr([G1|_],[G1|_],_,_)|_], Env = [M:amb(G1)|_].
```

```
typing(GEnv, Env, out(M), cap(G2,G2)):-
  GEnv = [G2:gr([G1, star(G1)|_],[G1|_],_,_)|_],
  Env = [M:amb(G1)|_].
```

The ambient rule is an example of a rule where an explicit instance of an environment is present, namely $\{g : \mathsf{gr}(\{g'\}, \varnothing, \varnothing, T)\}$, whose translation is immediate:

```
typing(GEnv, Env, amb(M,P), pr(G1)) :-
  typing(GEnv1, Env, P, pr(G)),
  sumunion([M:amb(G)|_],Env,[G:gr([G1|_],_,_,_)|_],GEnv1,GEnv).
```

Equally straightforward are all the remaining rules.

The `typing` predicate is thus the Prolog version of the $\vdash_\circ$ relation; the final judgment $\Gamma; \Delta \vdash_\bullet$ *trm*:*Type*, with the $\vdash_\bullet$ relation, is given by the procedure `typeinfer`:

```
typeinfer(GEnv,Env,P,Type):-
  typing(GEnvstar,Env,P,Type),
  closure(GEnvstar,Genv),
  createNames(GEnv,Env,P,Type).
```

where `closure` is the iterative procedure that performs the unfolding and elimination of starred groups; it also instantiates unbound communication types to `shh`, while `createNames` transforms Prolog names, represented by Prolog variables, into Prolog constants, and instantiates unbound message types to the atom `w`, representing the type $\bot_W$.

On top of the type inference procedure, a rudimentary parser implemented via a definite clause grammar and an elementary pretty-printer ensure the translation between the concrete and the abstract syntax, respectively in input and in output.

The SWI-Prolog code is available at the URL www.di.unito.it/ elio/dart/m3.pl.

6 Conclusions

In type systems for global computing a type inference algorithm that finds the minimal set of assumptions on the environment is essential. In the case of M^3 such an algorithm was formally specified by a rather complex set of rules, whose soundness and completeness w.r.t. a non-algorithmic type system were not so transparent; they have recently been certified through a non-trivial proof [11] in Coq.

In this paper it is shown how a different but still simple formulation of the system, where weakening is eliminated or possibly confined in a *top-level* rule, is almost directly readable as a logic program for type inference. This fact is well-known particularly in the area of type systems for functional programming, but it had not been exploited, as far as the author is aware, in type systems for ambient calculi.

The method, which brings together the tradition of logic programming and the studies in type theories, is quite general and may be applied to other new type systems for mobility, thus providing quick implementations to experiment with.

The resort to “impure” features of Prolog, due to the functional character of the G-union, could have probably been avoided by using a language that integrates logic programming with functional programming, or by supplementing Prolog with a new kind of "unification" corresponding to the G-union.

Being the need of such tools limited to a small and well-defined part of the system, the standard-Prolog solution was the quickest and the simplest; however, an approach based on one of the sophisticated extensions of logic programming currently available is certainly worth investigating. In particular, it would be important to evaluate which kind of programming paradigm – standard Prolog with its impure features, logic-functional language, etc. – is more suited to yield a provably correct implementation of a type system for mobility.

Acknowledgment

I owe a special thank to Mariangiola Dezani and Mario Coppo for the many discussions and suggestions on the subject of this paper.

Thanks also to Guido Governatori for useful general discussions on logic programming during my short visit at the University of Queensland.

Finally, I thank the anonymous referees for their helpful comments and remarks.

References

[1] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. North-Holland Publishing Co., Amsterdam, revised edition, 1984.

[2] M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *TACS 2001*, volume 2215 of LNCS, pages 38–63. Springer-Verlag, 2001.

[3] M. Bugliesi, G. Castagna. Secure Safe Ambients. In *POPL '01*, pages 222–235. ACM Press, 2001.

[4] M. Bugliesi, G. Castagna. Behavioural Typing for Safe Ambients. *Computer Languages*, 28(1), pages 61–99. Elsevier, 2002.

[5] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility Types for Mobile Ambients. In *ICALP'99*, volume 1644 of LNCS, pages 230–239. Springer-Verlag, 1999.

[6] L. Cardelli and G. Ghelli and A. D. Gordon. Types for the Ambient Calculus. *Information and Computation*, 177, pages 160–194. Elsevier, 2002.

[7] L. Cardelli and A. D. Gordon. Mobile Ambients. In *FoSSaCS'98*, volume 1378 of LNCS, pages 140–155. Springer-Verlag, 1998.

[8] M. Coppo, M. Dezani-Ciancaglini, E. Giovannetti, and I. Salvo. M^3: Mobility Types for Mobile Processes in Mobile Ambients. In *CATS 2003*, volume 78 of ENTCS. Elsevier, 2003.

[9] M. Hennessy and J. Riely. Resource Access Control in Systems of Mobile Agents (extended abstract). In *HLCL'98*, volume 16(3) of ENTCS. Elsevier Science, 1998. Final version in *Information and Computation*, 173, pages 82–120. Elsevier, 2002.

[10] M. Merro and V. Sassone. Typing and Subtyping Mobility in Boxed Ambients. In *CONCUR'02*, volume 2421 of LNCS, pages 304–320. Springer-Verlag, 2002.

[11] F. Honsell, I. Scagnetto. Mobility Types in Coq. Internal Report, Dept. of Mathematics and Informatics, University of Udine.

[12] J. Wells. The Essence of Principal Typings. In *ICALP'02*, volume 2380 of LNCS, pages 913–925. Springer-Verlag, 2002.