Ecole Normale
Supérieure de Lyon

Università di Torino
Dipartimento di Informatica
Lambda Group

# Subtyping and Typing Algorithms for Mobile Ambients

Pascal Zimmer, under the supervision of Mariangiola Dezani

Report of the internship held from 7 June to 6 August 1999 at the "Dipartimento di Informatica" of the "Università di Torino", part of the "Magistère d'Informatique et Modélisation" – second year

# Contents

# Chapter 1

# Introduction

## 1.1 The Department of Computer Science of the University of Turin

The Department of Computer Science of the University of Turin leads research activities in many fields, in parallel with its teaching activities. They can be grouped in the following areas (extract of the web page of the department):

- Artificial Intelligence (Knowledge Representation and Reasoning, Machine Learning, Natural Language Processing)
- Databases and Information Systems
- Decision Making Models and Management Systems
- Informatic Technology
- Linear Programming, Integer Linear Programming, Game Theory
- Logic Programming and Automated Reasoning
- Mathematical Logic
- Methods and Tools for the Analysis and Development of Performance Oriented Computer Systems
- Modelling in Biology and Medicine
- Models and Tools for the Design of Cooperative Systems
- Multidimensional Signal Processing
- Security and Computer Networks
- Semantics and Logics of Computation

The internship took place in this last group, also called "Lambda Group". This group, usually composed of a dozen persons, has its main research interests in program analysis and transformation, proof theory, semantics of programming languages and type systems. It is also part of some international research projects: ESPRIT-TYPES, HCM (Typed $\lambda$-Calculus), TMR (Linear Logic in

Computer Science), MURST 40% (Constructive Methods in Topology, Algebra and Program Analysis, and Formal techniques for specification, analysis, verification, synthesis and transformation of software systems).

## 1.2 Subject and Motivation

With the growing development of the World-Wide Web, it becomes interesting and fruitful to investigate the problems and properties of mobile code. At this point, we should distinguish two kinds of mobility: *mobile computing*, that is to say computation in mobile devices like a laptop, and *mobile computation*, that is to say mobile code moving between different devices, like applets or agents. The *ambient calculus* was designed to model both of these aspects within a single framework. Along with mobile computation in itself, it shows how the notions of administrative domains, their crossing, firewalls, authorizations... can be formalized in a calculus. In this sense, it is more appropriate than the $\pi$-calculus ([Mil91]), even if the bases are the same (for more discussion about the problems raised by mobility and computation over wide-area networks, see [Car99a, Car99b]).

Informally, an ambient is a bounded place, with an inside and an outside, where computation happens. Many ambients can be nested so that they form a hierarchy. Each of them has a name (not necessarily distinct from other ambient names), which will be used to control access. An ambient can be moved as a whole with all the computation and subambients it contains: it can enter another ambient or exit it. It can also be opened so that its contents get visible at the current level. For more intuitions motivating the ambient calculus or its graphical vision (the *folder calculus*), we recommend reading [CG97, CG98, Car99a].

In order to prove some specific properties concerning mobility, locking, communication... in the ambient calculus, Cardelli and Gordon proposed a simple type system, undeterministic and without subtyping (see [CG99, CGG99a, CGG99b]). The aim of our work was to introduce a subtyping relation, deduce a typing algorithm and, in doing this, to make the type system more suitable for a treatment of mobile ambients as a "programming language for mobility" (departing thus from the "ambients as a specifical language for reasoning on mobility" approach).

The rest of the report is organized as follows. In Chapter 2, we will review briefly the ambient calculus and its semantics, by the way adding two new constructs to the original syntax. Then, in Chapter 3, we present our extension of the type system, introduce a subtyping ordering, and define a new typing system. In Chapter 4, we show that the set of derivable types for a term has got a minimum, and we give an algorithm to compute it efficiently. It is now possible to type-check a term of the ambient calculus automatically. The next step in Chapter 5 is to remove all type annotations from the term and try to find a type-inference algorithm. We give a complete solution for the original type system without subtyping and a partial one for the new type system. Appendix A reminds the original type system and Appendix B contains all the proofs of our main results.

# Chapter 2

# Mobile Ambients

In this chapter, we are going to briefly review the polyadic ambient calculus we will use throughout this paper. We will try to explain all its constructs and rules, but we recommend reading [CG98] (or any other paper presenting the calculus extensively) to have a more complete presentation. By the way, we are also going to extend the original calculus and introduce two new constructs.

## 2.1 Syntax

The polyadic ambient calculus is mainly composed of processes. As in many other process calculi, we have an inactive process $\mathbf{0}$ which does nothing, we can compose processes in parallel $(P \mid Q)$, we have a construct to replicate a process as many times as necessary $(!\ P)$ and we have a restriction operator $((\nu n)P)$ which introduces a new name $n$ and restricts its scope to the inside of $P$. In the $\pi$-calculus, those names represented channels; here they represent ambient names. In our calculus, we also declare the type of this ambient name $((\nu n : Amb^Y[T_n, T'_n])P)$, but we will not care about that until the next chapter.

An ambient is composed of a name $n$ and a process $P$ which is running inside the ambient. We write it $n[P]$. Here we extend the syntax of the polyadic ambient calculus with a new construction. Up to now, the locking-unlocking of an ambient was defined only in the declaration of its name. So ambients with the same name had all the same locking annotation. We kept this possibility (extending it with ordering), but we changed the syntax of ambient constructions: $n[P]$ is an unlocked ambient and $n[\![P]\!]$ is a locked one. So we can now have an explicit construct which guarantees that an ambient will be locked.

The process $M.P$ executes the action induced by the capability $M$ and then continues with the process $P$. There are three kinds of capabilities: one to enter an other ambient (*in* $n$), one to exit (*out* $n$) and one to open up an ambient (*open* $n$). To build such a capability, a process must know the name $n$. It can also have received the capability via communication (see below). Implicitly, it is impossible to reconstruct the ambient name $n$ out of one or all of these capabilities (this is important to prove the security of a firewall for example). We can also compose capabilities into paths $(M.M')$ where $\varepsilon$ is the empty path.

Here we add our second extension: the *imm* capability. A process containing it must be immobile. We added it, because also for immobility we want a language construct which obliges a process to be immobile, instead of delegating it to the types.

Finally, we have two communication primitives: $(n_1 : W_1, \ldots, n_k : W_k).P$ and $\langle M_1 \times \cdots \times M_k \rangle$. The first waits for a tuple of values of respective types $W_1, \ldots, W_k$, and binds them to the variables $n_1, \ldots, n_k$ in the continuing process $P$. The second outputs a tuple of values. Note that the output is asynchronous (no continuing process). Communicable objects can be ambient names, capabilities or paths. There could also be integers or booleans, but we do not consider them in our basic calculus.

Here is the complete syntax of our calculus. Note that it allows strange expressions like $(in\ n)[P]$ or $out\ (out\ n).P$. Rejecting those non-sense terms will be an automatic property of the type system.

$$
\begin{array}{lll}
P, Q & ::= & \text{processes} \\
 & (\nu n : Amb^{Y}[T, T'])P & \text{restriction} \\
 & \mathbf{0} & \text{inactivity} \\
 & P \mid Q & \text{composition} \\
 & !P & \text{replication} \\
 & M[P] & \text{unlocked ambient} \\
 & M[\![P]\!] & \text{locked ambient} \\
 & M.P & \text{action} \\
 & (n_1 : W_1, \ldots, n_k : W_k).P & \text{input} \\
 & \langle M_1, \ldots, M_k \rangle & \text{async output} \\
M & ::= & \text{expressions} \\
 & n & \text{name} \\
 & in\ M & \text{can enter into } M \\
 & out\ M & \text{can exit out of } M \\
 & open\ M & \text{can open } M \\
 & imm & \text{immobility} \\
 & \varepsilon & \text{null path} \\
 & M.M' & \text{composite path}
\end{array}
$$

Terms are also identified up to the consistent renaming of bound variables, in the restriction and input constructs. Thus, we can always suppose that all the ambient names and input variables are distinct.

## 2.2 Structural Congruence

Terms are identified up to an equivalence relation, $\equiv$, called structural congruence. This relation is the least equivalence relation satisfying the following rules:

$$
\begin{array}{lll}
 & P \equiv P & \text{(Struct Refl)} \\
 & P \equiv Q \;\Rightarrow\; Q \equiv P & \text{(Struct Symm)} \\
 & P \equiv Q, Q \equiv R \;\Rightarrow\; P \equiv R & \text{(Struct Trans)} \\
 & P \equiv Q \;\Rightarrow\; (\nu n : W)P \equiv (\nu n : W)Q & \text{(Struct Res)} \\
 & P \equiv Q \;\Rightarrow\; P \mid R \equiv Q \mid R & \text{(Struct Par)} \\
 & P \equiv Q \;\Rightarrow !P \equiv !Q & \text{(Struct Repl)} \\
 & P \equiv Q \;\Rightarrow\; M[P] \equiv M[Q] & \text{(Struct Amb}_{\circ}\text{)} \\
(*) & P \equiv Q \;\Rightarrow\; M[\![P]\!] \equiv M[\![Q]\!] & \text{(Struct Amb}_{\bullet}\text{)} \\
 & P \equiv Q \;\Rightarrow\; M.P \equiv M.Q & \text{(Struct Action)} \\
 & P \equiv Q \;\Rightarrow\; (n_1 : W_1, \ldots, n_k : W_k).P \equiv (n_1 : W_1, \ldots, n_k : W_k).Q & \text{(Struct Input)} \\
 & P \mid Q \equiv Q \mid P & \text{(Struct Par Comm)} \\
 & (P \mid Q) \mid R \equiv P \mid (Q \mid R) & \text{(Struct Par Assoc)} \\
 & !P \equiv P \mid !P & \text{(Struct Repl Par)}
\end{array}
$$

| | | |
|---|---|---|
| | $(\nu n:W)(\nu m:W')P \equiv (\nu m:W')(\nu n:W)P$ if $n \neq m$ | (Struct Res Res) |
| | $(\nu n:W)(P \mid Q) \equiv P \mid (\nu n:W)Q$ if $n \notin fn(P)$ | (Struct Res Par) |
| | $(\nu n:W)m[P] \equiv m[(\nu n:W)P]$ if $n \neq m$ | (Struct Res Amb$_\circ$) |
| $(*)$ | $(\nu n:W)m[\![P]\!] \equiv m[\![(\nu n:W)P]\!]$ if $n \neq m$ | (Struct Res Amb$_\bullet$) |
| | $P \mid \mathbf{0} \equiv P$ | (Struct Zero Par) |
| | $(\nu n: Amb^Y[T,T'])\mathbf{0} \equiv \mathbf{0}$ | (Struct Zero Res) |
| | $!\mathbf{0} \equiv \mathbf{0}$ | (Struct Zero Repl) |
| | $\varepsilon.P \equiv P$ | (Struct $\varepsilon$) |
| | $(M.M').P \equiv M.M'.P$ | (Struct .) |

We do not detail them. The new rules (marked by $(*)$) are just here to take the new constructions into account.

## 2.3 Reduction Rules – Operational Semantics

Here are the reduction rules of our ambient calculus. They are quite the same of the original ambient calculus. The (Red In) and (Red Out) rules can be applied to any kind of ambients, whereas the (Red Open) rule can only be applied to unlocked ambients. The only completely new rules are (Red Amb$_\bullet$) and (Red Imm), which allows to continue a process beyond the *imm* capability.

| | | |
|---|---|---|
| $(*)$ | $n[in\ m.P \mid Q] \mid m[R] \ \rightarrow\ m[n[P \mid Q] \mid R]$ | (Red In) |
| $(*)$ | $m[n[out\ m.P \mid Q] \mid R] \ \rightarrow\ n[P \mid Q] \mid m[R]$ | (Red Out) |
| | (in (Red In) and (Red Out), each occurrence of $[.]$ can be replaced by $[\![.]\!]$ ) | |
| | $open\ n.P \mid n[Q] \ \rightarrow\ P \mid Q$ | (Red Open) |
| $(*)$ | $imm.P \ \rightarrow\ P$ | (Red Imm) |
| | $(n_1:W_1,\ldots,n_k:W_k).P \mid \langle M_1,\ldots,M_k\rangle \ \rightarrow\ P\{n_1 \leftarrow M_1,\ldots,n_k \leftarrow M_k\}$ | (Red Comm) |
| | $P \ \rightarrow\ Q \ \Rightarrow\ (\nu n:W)P \ \rightarrow\ (\nu n:W)Q$ | (Red Res) |
| | $P \ \rightarrow\ Q \ \Rightarrow\ n[P] \ \rightarrow\ n[Q]$ | (Red Amb$_\circ$) |
| $(*)$ | $P \ \rightarrow\ Q \ \Rightarrow\ n[\![P]\!] \ \rightarrow\ n[\![Q]\!]$ | (Red Amb$_\bullet$) |
| | $P \ \rightarrow\ Q \ \Rightarrow\ P \mid R \ \rightarrow\ Q \mid R$ | (Red Par) |
| | $P' \equiv P, P \ \rightarrow\ Q, Q \equiv Q' \ \Rightarrow\ P' \ \rightarrow\ Q'$ | (Red $\equiv$) |

The (Red In) rule states that an ambient named $n$ containing a process with the capability *in* $m$ ready to be executed can be moved with all its contents inside an ambient named $m$ if there is one on the same level (i.e. in parallel). The (Red Out) rule does the opposite action. These moves are called "subjective moves" because the decision to move comes from the inside of $n$. One could also consider "objective moves", for which the decision is taken outside, which seem simplier:

$$in\ n.P \mid n[Q] \ \rightarrow\ n[P \mid Q]$$
$$n[out\ n.P \mid Q] \ \rightarrow\ P \mid n[Q]$$

But further developments show that these rules are dangerous because they allow to entrap any ambient into another, so that he can never go outside anymore. This would be a big lack of security. However, it is possible to encode objective moves into the calculus such that they can be performed on ambients who allow them explictly (see [CG98] for details).

The (Red Open) rule allows to open up an ambient named $n$ when there is one on the same level. As before, we could consider an other rule called "dissolution", using a capability *acid*:

$$n[acid.P \mid Q] \;\rightarrow\; P \mid Q$$

In this rule, the ambient is dissolved from the inside. It must also be rejected because we can encode objective moves by using *acid*...

The (Red Imm) rule is very simple and actually "ignores" the *imm* capability. In fact, *imm* is only useful when typing: if $imm.P$ is typable, then $P$ cannot contain moving capabilities (even by receiving them). At run-time, only this garuantee is important and we can throw *imm* away.

The (Red Comm) rule defines communication of a tuple of values: the actual values are bound to variables of the continuing process. Note that a communication can only occur if there are one output and one input instructions at the same level.

The five last rules just say that reduction (i.e. computation) can also occur beyond scope restrictions, inside ambients, or by using the structural congruence rewriting.

## 2.4 The Power of the Ambient Calculus

We give here a few example of objects that can be encoded using the ambient calculus (see [CG98] for definitions):

**Locks:** Useful to synchronize two processes.

**Firewall Access:** The ambient calculus describes the mechanism of crossing a firewall quite simply.

**Booleans and Numerals**

**Turing Machines:** They can be directly encoded in the calculus (even without using the communication primitives). This implies that the ambient calculus is Turing-powerful !

**Cells and Records**

**Packets Routed over a Network:** The encoding is very direct since the calculus has been designed for this kind of work.

**Asynchronous $\pi$-calculus,** and as a consequence, the $\lambda$-calculus.

# Chapter 3

# A Type System with Subtyping

In order to verify some properties of processes in the ambient calculus, Cardelli and Gordon proposed a first type system in [CG99] and extended it with Ghelli in [CGG99a, CGG99b]. It assured that a well-typed process could not cause certain kinds of run-time fault: exchanging values of the wrong type, moving or being opened if they are not allowed to... We will always refer to this type system as "CGG" (its types and rules are briefly reminded in Appendix A). It contained interesting ideas, but this type system was completely undeterministic and thus could not be translated into a typing algorithm, and there was not any subtyping relation.

In this chapter, we will describe a new type system, extending (in some way) the one of CGG. We will first introduce some new types and define an ordering relation on them. Then, we will give new typing rules and show some properties.

## 3.1 Type Definitions

We start by giving all the definitions of our types (we will discuss them in the next paragraphs):

$$
\begin{array}{lll}
Z & ::= & \text{mobility annotations} \\
 & \perp_{\rightarrow} & \text{mobility unknown} \\
 & \veebar & \text{immobile} \\
 & \curvearrowright & \text{mobile} \\
 & \top^{\rightarrow} & \text{mobile and immobile}
\end{array}
\qquad
\begin{array}{lll}
Y & ::= & \text{locking annotations} \\
 & \perp_{\circ} & \text{locking bottom} \\
 & \bullet & \text{locked} \\
 & \circ & \text{unlocked} \\
 & \top^{\circ} & \text{locking top}
\end{array}
$$

$$
\begin{array}{lll}
O, I & ::= & \text{input/output types} \\
 & \perp & \text{bottom value} \\
 & W_1 \times \cdots \times W_k \quad (k \geq 0) & \text{tuple} \\
 & \top & \text{top value}
\end{array}
\qquad
\begin{array}{lll}
T & ::= & \text{process type} \\
 & {}^{Z}O \rightsquigarrow I &
\end{array}
$$

$$
\begin{array}{lll}
W & ::= & \text{message types} \\
 & Amb^{Y}[T, T'] \ \text{ with } T \leq T' \text{ (see below)} & \text{ambient name} \\
 & Cap[T] & \text{capability}
\end{array}
$$

A tuple with $k = 0$ will be noted $\mathbf{1}$.

## 3.2 Intuitive Meanings

### Mobility Annotations

The meaning of the mobility annotations depends on the kind of object to which they are applied:

| Mobility annotation | In the type of a process | In the type of an ambient name |
|---|---|---|
| $\perp^{\!\!\rightarrow}$ | Mobility unknown (both are possible) | The ambient allows only processes for which the mobility is unknown (like **0** or $n[P]$) |
| $\curvearrowright$ | The process produces movement instructions | The ambient allows only mobile processes |
| $\underline{\vee}$ | The process must be immobile | The ambient allows only immobile processes |
| $\top^{\!\!\rightarrow}$ | This case will be disallowed by the type system | The ambient allows any kind of process (concerning mobility) |

### Process Type

In the type system of CGG, there was only one single term representing the type of values exchanged in the ambient ($Shh$ or a tuple). In presence of subtyping, we should now accept that outputs and inputs have different types. For example, the output of the integer 1 should be accepted by an input variable of type $Real$. So we decided to track the types of output and input values exchanged in the ambient. If a process is valid, it must then have type ${}^{Z}O \rightsquigarrow I$ with $O \leq I$ to ensure that any output can be read by any input instruction (note that this is not specified in the syntax of $T$, but will be a property of the type system; see also below). Afterwork, it appeared that Yoshida and Hennessy used the same approach for a higher-order $\pi$-calculus with subtyping ([YH99]). Moreover, a valid process should not contain both mobile and immobile instructions; consequently the mobility annotation $\top^{\!\!\rightarrow}$ is forbidden.

**Definition 1 (Validity)** *A process type* ${}^{Z}O \rightsquigarrow I$ *is said to be* valid *if* $Z \neq \top^{\!\!\rightarrow}$ *and* $O \leq I$.

### Input/Output Types

Then, we have to define what are output and input types. As before, it can be a tuple. But we had to replace $Shh$ by two different values, one for outputs ($Shh_{out} = \perp$) and one for inputs ($Shh_{in} = \top$). Then it appeared useful to consider that the meaning of these values is different for the input and output terms:

| Value | Output term ($O$) | Input term ($I$) |
|---|---|---|
| $\perp$ | No output - dumb process ($Shh_{out}$) | There can be an input of any type |
| $\top$ | There can be an output of any type | No input - deaf process ($Shh_{in}$) |

For example, if there are two outputs of different arities in parallel, the resulting process has type $\top$ for $O$. With the condition that $O \leq I$, the process is valid if and only if $I = \top$, i.e. if there are no input instruction (you can say anything only if nobody is listening). A similar argument holds by exchanging inputs and outputs. This is a different vision from [YH99] where $\top$ was forbidden as an output type and $\perp$ as an input type.

### Message Types

Finally, we have a type for messages that can be exchanged in an ambient. It can be either an ambient name or a capability. It is safe to add here other common types like $Int$, $Bool$...

A capability $Cap[T]$ contains the effects $T$ which will be released if the capability is executed.

In the type of an ambient name $Amb^Y[T, T']$, $T$ represents the maximal type allowed for processes inside the ambient (i.e. all valid processes also have type $T$) (cf. rules (Proc Amb$_\circ$) and (Proc Amb$_\bullet$) in Section 3.4), whereas $T'$ represents the maximal effect a valid process can produce (cf. rule (Exp Open)), thus the condition $T \leq T'$ to be coherent.

At first sight, it could seem strange to declare a new ambient name with $T < T'$: if we specify the maximal allowed type $T$, why would we say that worse effects $T'$ can appear when opening an ambient of that name ? In fact, we need them to be consistent with the rest of the calculus. Suppose we want to write the program $\langle n \rangle \mid \langle m \rangle \mid (x :?).P$ where $n$ and $m$ accept processes of type $T_n$ and $T_m$ respectively. What input type should we declare for $x$ ? We can use the type of the parallel output $\langle n \rangle \mid \langle m \rangle$, which is with our ordering $Amb^Y[T_n \wedge T_m, T_n \vee T_m]$. In general, there is no reason to have $T_n \wedge T_m = T_n \vee T_m$. This explains why we cannot replace $T$ and $T'$ by one single process type in an ambient name.

The meaning of the locking annotation $Y$ in an ambient name is the following:

| Locking annotation | In the type of an ambient name $n$ |
|---|---|
| $\perp_\circ$ | All three constructions $n[P]$, $n[\![P]\!]$ and $open\ n$ are allowed |
| $\circ$ | Only the constructions $n[P]$ and $open\ n$ are allowed |
| $\bullet$ | Only the construction $n[\![P]\!]$ is allowed |
| $\top^\circ$ | Ambients of name $n$ and the capability $open\ n$ are not allowed |

Note that all other constructions (like $in\ n$ or $\langle n \rangle$) are valid with any locking annotation.

Note also that the condition $O \leq I$ is not required for an ambient name. For example, $Amb^Y[{}^{\top^{\rightarrow}}\top \rightsquigarrow \perp, {}^{\top^{\rightarrow}}\top \rightsquigarrow \perp]$ is the type of an ambient name allowing all processes, whereas the ambient name $Amb^Y[{}^{\perp_{\rightarrow}}\perp \rightsquigarrow \top, {}^{\perp_{\rightarrow}}\perp \rightsquigarrow \top]$ is the most restrictive one, allowing only processes which do not have inputs or outputs, i.e. only processes behaving like $\mathbf{0}$.

## 3.3 Ordering

The main intuition in defining the order is to always respect the subsumption rule: if $P$ has got type $T$ and $T \leq T'$, then $P$ has also type $T'$.

### Mobility and Locking Annotations

For the mobility annotations:

$$\perp_{\rightarrow} \leq \veebar, \curvearrowright \leq \top^{\rightarrow}$$

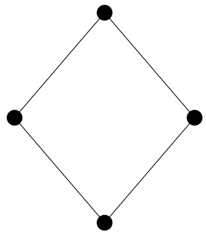
with $\veebar$ and $\curvearrowright$ uncomparable.

For the locking annotations:

$$\perp_\circ \leq \bullet, \circ \leq \top^\circ$$

with $\bullet$ and $\circ$ uncomparable.

Consequently, the ordering for both of them has exactly the following shape:

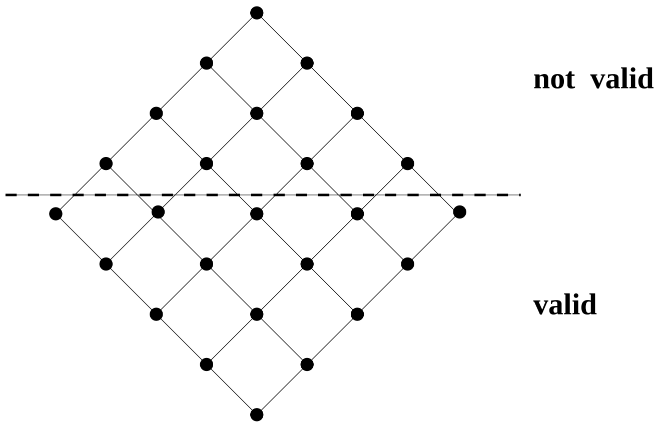


## Process Type

For the process type, we define :

$$^{Z}O \rightsquigarrow I \leq {}^{Z'}O' \rightsquigarrow I' \iff \begin{cases} Z \leq Z' \\ O \leq O' \\ I \geq I' \end{cases}$$

The ordering has the following shape:




The set of all process types has a complete lattice structure, with ${}^{\perp\!\!\to}\bot \rightsquigarrow \top$ as the minimal element and ${}^{\top\!\!\to}\top \rightsquigarrow \bot$ as the maximal one. But, if we consider only valid processes, there are many maximal types: all ${}^{\curvearrowright}O \rightsquigarrow O$ and ${}^{\veebar}O \rightsquigarrow O$ for every input/output type $O$.

## Message Types

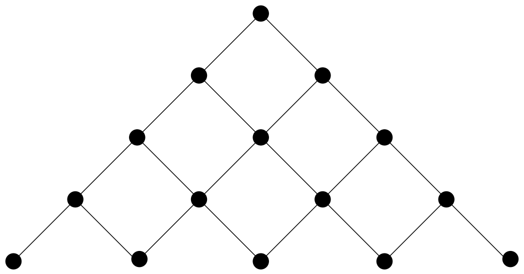
For the message types, we define :

$$Amb^{Y}[T_1, T_2] \leq Amb^{Y'}[T_1', T_2'] \iff \begin{cases} Y \leq Y' \\ T_1 \geq T_1' \\ T_2 \leq T_2' \end{cases}$$

$$Cap[T] \leq Cap[T'] \iff T \leq T'$$

There is no comparability between ambient names and capabilities. It is safe to add other useful types here with their usual ordering (for example, $Int \leq Real$), with no comparability with ambient names and capabilities.

The set of capability types has a structure similar to the process types (i.e. a complete lattice). The set of ambient names has the following shape:



It has a maximal element ($Amb^{\top\circ}[{}^{\perp\to}\perp \leadsto \top, {}^{\top\to}\top \leadsto \perp]$), but infinitely many minimal elements: all $Amb^{\perp\circ}[T,T]$ for every process type $T$.

### Input/Output Types

For the input/output types, we define the partial order :

$$\perp \leq W_1 \times \cdots \times W_k \leq \top \qquad \forall k\ \forall W_i$$

$$W_1 \times \cdots \times W_k \leq W_1' \times \cdots \times W_k' \iff W_i \leq W_i' \ \ \forall\, 1 \leq i \leq k$$

This definition induces a complete lattice structure on input/output types.

## 3.4 Typing Rules

Having defined the types and explained their ordering, we can now give the typing rules of this new type system. A feature of the present approach w.r.t. CGG is that we avoid to introduce arbitrary types in the conclusions of typing rules. However note that this derivation system is also not deterministic because of the subsumption rules and the shape of some rules (for example, in (Proc Par), the same type $T$ appears in two premises).

### Good Environment ($E \vdash \diamond$)

$$\text{(Env }\emptyset\text{)}\ \frac{}{\emptyset \vdash \diamond} \qquad\qquad \text{(Env } n\text{)}\ \frac{E \vdash \diamond \quad n \notin dom(E)}{E, n : W \vdash \diamond}$$

These two rules are exactly the same as in CGG.

### Good Expression of Type $W$ ($E \vdash M : W$)

$$\text{(Exp }\varepsilon\text{)}\ \frac{E \vdash \diamond}{E \vdash \varepsilon : Cap[{}^{\perp\to}\perp \leadsto \top]} \qquad\qquad \text{(Exp .)}\ \frac{E \vdash M : Cap[T] \quad E \vdash M' : Cap[T]}{E \vdash M.M' : Cap[T]}$$

$$\text{(Exp In)}\ \frac{E \vdash M : Amb^{Y}[T,T']}{E \vdash in\ M : Cap[{}^{\frown}\perp \leadsto \top]} \qquad\qquad \text{(Exp Out)}\ \frac{E \vdash M : Amb^{Y}[T,T']}{E \vdash out\ M : Cap[{}^{\frown}\perp \leadsto \top]}$$

$$\text{(Exp Open)}\ \frac{E \vdash M : Amb^{\circ}[T,T']}{E \vdash open\ M : Cap[T']} \qquad\qquad \text{(Exp Imm)}\ \frac{E \vdash \diamond}{E \vdash imm : Cap[{}^{\underline{\vee}}\perp \leadsto \top]}$$

$$\text{(Exp } n\text{)}\ \frac{E', n : W, E'' \vdash \diamond}{E', n : W, E'' \vdash n : W} \qquad\qquad \text{(Exp Sub)}\ \frac{E \vdash M : W \quad W \leq W'}{E \vdash M : W'}$$

In (Exp $\varepsilon$), (Exp In), (Exp Out) and (Exp Imm), the type in the conclusion of the rule is the minimal effect (or constraint) that the corresponding instruction produces. In (Exp Open), we use the maximal effect contained in the ambient name and we check that $M$ is an unlocked ambient. In (Exp Sub), we allow to upgrade the type of an expression. The other rules are identical to those of CGG.

### Good Process of Type $T$ ($E \vdash P : T$)

$$\text{(Proc Par)}\ \frac{E \vdash P : T \quad E \vdash Q : T}{E \vdash P \mid Q : T} \qquad \text{(Proc Action)}\ \frac{E \vdash M : Cap[T] \quad E \vdash P : T}{E \vdash M.P : T}$$

$$\text{(Proc Zero)}\ \frac{E \vdash \diamond}{E \vdash \mathbf{0} : {}^{\dashv}\!\bot \rightsquigarrow \top} \qquad \text{(Proc Repl)}\ \frac{E \vdash P : T}{E \vdash !P : T}$$

$$\text{(Proc Res)}\ \frac{E, n : Amb^{Y}[T_n, T'_n] \vdash P : T}{E \vdash (\nu n : Amb^{Y}[T_n, T'_n])P : T}$$

$$\text{(Proc Amb}_{\circ}\text{)}\ \frac{E \vdash M : Amb^{\circ}[T, T'] \quad E \vdash P : T}{E \vdash M[P] : {}^{\dashv}\!\bot \rightsquigarrow \top}$$

$$\text{(Proc Amb}_{\bullet}\text{)}\ \frac{E \vdash M : Amb^{\bullet}[T, T'] \quad E \vdash P : T}{E \vdash M[\![P]\!] : {}^{\dashv}\!\bot \rightsquigarrow \top}$$

$$\text{(Proc Input)}\ \frac{E, n_1 : W_1, \ldots, n_k : W_k \vdash P : {}^{Z}O \rightsquigarrow I \quad I \leq W_1 \times \cdots \times W_k}{E \vdash (n_1 : W_1, \ldots, n_k : W_k).P : {}^{Z}O \rightsquigarrow I}$$

$$\text{(Proc Output)}\ \frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k \quad (E \vdash \diamond \ \text{if}\ \ k = 0)}{E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\dashv}\!W_1 \times \cdots \times W_k \rightsquigarrow \top}$$

$$\text{(Proc Sub)}\ \frac{E \vdash P : T \quad T \leq T' \quad T'\ valid}{E \vdash P : T'}$$

(Proc Par), (Proc Action), (Proc Repl) and (Proc Res) are the same rules as those of CGG (with a syntax modification for (Proc Res)). In (Proc Zero), we use the minimal process type.

In (Proc Amb$_{\circ}$), we check that $M$ is an unlocked ambient name and that $P$ has the type of an allowed process inside $M$ (with the subsumption rule, we can always upgrade it or decrease the type in the ambient name so that they match). Like for $\mathbf{0}$, we use the minimal process type in the conclusion of the rule. (Proc Amb$_{\bullet}$) is similar.

In (Proc Input), we just need to check that the input type of both process $P$ and conclusion is below the type generated by the input (i.e. is more specific since the ordering for input types is contravariant). This is valid: every input type is accepted in the conclusion provided that it covers the input $W_1 \times \cdots \times W_k$. If $P$ has a bigger input type ($\top$ for $\mathbf{0}$ for example), it must first be upgraded with the subsumption rule before applying (Proc Input).

In (Proc Output), we just give the minimal effect: the output of a type $W_1 \times \cdots \times W_k$. Since the output is asynchronous, there is no condition to check like in (Proc Input).

(Proc Sub) is the classical subsumption rule, with the additional condition that the new process type must be valid (we are explicitly typing a process here and not a capability or an ambient name).

## 3.5 Results

Our first result will be to show that there cannot be run-time faults:

**Theorem 2 (Validity)** *If* $E \vdash P : T$*, then* $T$ *is valid.*

By this theorem, we are sure there will never be an exchange of incompatible values during execution of a well-typed process. And it cannot contain instructions requiring both mobility and immobility (like in $imm.in\ n.P$). Another desired property is that we do not want an ambient to be opened if it is locked. This property is a direct result of the type system: the instruction $open\ n$ can be typed only if we can prove that $n$ is unlocked.

Then, the following theorem is probably the most important to validate our type system:

**Theorem 3 (Subject reduction)** *If* $E \vdash P : T$ *and* $P \rightarrow Q$*, then* $E \vdash Q : T$*.*

## 3.6 Link with the Type System of Cardelli-Ghelli-Gordon

Now that we have proved the same main properties, we are going to show why our type system extends CGG and try to explain why it types a little more, before further practical developments in the next chapter. What we would like to show now is that if a process is typable in the system of CGG, then it is also in the new system, and its type is less than the one in CGG.

To formalize this notion, we must first define a translation from the system of CGG to the new system. Let $f$ denote this translation.

$$f(Shh) = \bot \rightsquigarrow \top$$
$$f(W_1 \times \cdots \times W_k) = f(W_1) \times \cdots \times f(W_k) \rightsquigarrow f(W_1) \times \cdots \times f(W_k)$$

$$f(Amb^Y[{}^Z U]) = Amb^Y[{}^Z f(U), {}^Z f(U)]$$
$$f(Cap[{}^Z U]) = Cap[{}^Z f(U)]$$

We extend naturally $f$ to environments, and for processes $f$ consists in replacing the type annotations in restrictions and inputs, and using $n[P]$ or $n[\![P]\!]$ according to the declaration of $n$ ($n[P]$ if $n$ is unlocked, $n[\![P]\!]$ if $n$ is locked).

Then, we have the following theorem:

**Theorem 4 (Link)**

- *If* $E \vdash_{CGG} \diamond$*, then* $f(E) \vdash \diamond$*.*
- *If* $E \vdash_{CGG} M : W$*, then* $f(E) \vdash M : f(W)$*.*
- *If* $E \vdash_{CGG} P : {}^Z U$*, then* $f(E) \vdash f(P) : {}^Z f(U)$*.*

The reciprocal is false: there are many processes typable in the new system which were not typable in CGG. A lot of them are due to the syntax and type extensions of our system. But there are also some examples not requiring the new syntax. First, because outputs and inputs can now have different values: $\langle 1 \rangle \mid (x : Real).P$ is not typable in CGG (adding $Int \leq Real$ to our type system).

We can also now output different types in parallel:

$$(\nu n : Amb^\circ[{}^{\veebar} Shh])(\nu m : Amb^\circ[{}^{\curvearrowright} Shh])(\langle n \rangle \mid \langle m \rangle)$$

is not typable in CGG, but

$$\begin{aligned} & f(\;\; (\nu n : Amb^\circ[{}^{\stackrel{\vee}{-}}Shh])(\nu m : Amb^\circ[{}^{\curvearrowright}Shh])(\langle n\rangle \mid \langle m\rangle) \;\;) \\ = \;\; & (\nu n : Amb^\circ[{}^{\stackrel{\vee}{-}}\bot \rightsquigarrow \top, {}^{\stackrel{\vee}{-}}\bot \rightsquigarrow \top])(\nu m : Amb^\circ[{}^{\curvearrowright}\bot \rightsquigarrow \top, {}^{\curvearrowright}\bot \rightsquigarrow \top])(\langle n\rangle \mid \langle m\rangle) \end{aligned}$$

is typable and has got type ${}^{\lrcorner\!\to}Amb^\circ[{}^{\lrcorner\!\to}\bot \rightsquigarrow \top, {}^{\overset{\to}{\top}}\bot \rightsquigarrow \top] \rightsquigarrow \top$ in the new system.

But there are even some derivations giving a process a type $T$ where neither $T$ nor the derivation use any new type constructs, but such that $T$ cannot be derived in CGG. The simplest example of this shape is $\langle 1\rangle$ (here we will add $Int \leq Real$ to our type system, but we could do the same with ambient names or capability). In our new type system, we can derive (we do not worry about mobility annotations for simplicity): $E \vdash \langle 1\rangle : Int \rightsquigarrow \top$. This is the most "natural" typing (in fact, the minimal one). But with the help of the subsumption rule and since $Int \leq Real \leq \top$, we can also derive $E \vdash \langle 1\rangle : Real \rightsquigarrow Real$, that is to say $E \vdash \langle 1\rangle : f(Real)$. And of course we will never be able to derive $E \vdash_{CGG} \langle 1\rangle : Real$ in CGG...

This shows definitely that our extension is not conservative but slightly and subtly different from CGG. This is the price to pay to obtain a more consistent and powerful type system, and to be able to go further...

# Chapter 4

# A First Typing Algorithm

In this chapter, we are going to deduce a typing algorithm from the typing system we introduced in the previous one. Then, we will see that this algorithm returns exactly all the types (in a certain sense) that could be derived.

## 4.1 Meet and Join Operations

First, we need to define meet and join operations on types. Those definitions are deduced from the ordering relation.

### Mobility and Locking Annotations

Since these two structures are complete lattices (4 points with a lozenge-like ordering), there is no problem to define the meet and join operations on them.

### Process Type

The ordering leads to the following definition for meet and join operations:

$$^{Z}O \rightsquigarrow I \wedge {}^{Z'}O' \rightsquigarrow I' \triangleq {}^{Z \wedge Z'}O \wedge O' \rightsquigarrow I \vee I'$$

$$^{Z}O \rightsquigarrow I \vee {}^{Z'}O' \rightsquigarrow I' \triangleq {}^{Z \vee Z'}O \vee O' \rightsquigarrow I \wedge I'$$

### Message Types

We can only define partial meet and join operations, since there are some uncomparable types:

$$Amb^{Y}[T_1, T_2] \wedge Amb^{Y'}[T_1', T_2'] \triangleq Amb^{Y \wedge Y'}[T_1 \vee T_1', T_2 \wedge T_2']$$
$$\text{if } T_1 \vee T_1' \leq T_2 \wedge T_2' \text{ (or equivalently if } T_1 \leq T_2' \text{ and } T_1' \leq T_2)$$

$$Amb^{Y}[T_1, T_2] \vee Amb^{Y'}[T_1', T_2'] \triangleq Amb^{Y \vee Y'}[T_1 \wedge T_1', T_2 \vee T_2']$$

$$Cap[T] \wedge Cap[T'] \triangleq Cap[T \wedge T']$$

$$Cap[T] \vee Cap[T'] \triangleq Cap[T \vee T']$$

The other cases are undefined.

### Input/Output Types

The ordering gives the definitions for meet and join operations (which are always defined, since the input/output types have a lattice structure) :

$$W_1 \times \cdots \times W_k \wedge W'_1 \times \cdots \times W'_{k'} \triangleq \begin{cases} W_1 \wedge W'_1 \times \cdots \times W_1 \wedge W'_k & \text{if } k = k' \text{ and } W_i \wedge W'_i \\ & \text{is defined } \forall\ 1 \leq i \leq k \\ \bot & \text{otherwise} \end{cases}$$

$$W_1 \times \cdots \times W_k \vee W'_1 \times \cdots \times W'_{k'} \triangleq \begin{cases} W_1 \vee W'_1 \times \cdots \times W_1 \vee W'_k & \text{if } k = k' \text{ and } W_i \vee W'_i \\ & \text{is defined } \forall\ 1 \leq i \leq k \\ \top & \text{otherwise} \end{cases}$$

(the obvious cases with $\bot$ and $\top$ are omitted for simplicity)

## 4.2 Typing Rules

**Definition 5** *An environment is said to be* well-formed *if all the names it contains are different. This is of course equivalent to $E \vdash \diamond$. Algorithmically, it just consists in checking that all the names are different.*

For any well-formed environment $E$, we define an algorithm returning the type of expressions and processes by the following rules. For every undefined case, we will say that the algorithm *fails*. Note that even if we write it as derivation rules for simplicity, it can also be expressed directly in an algorithmical way. Note also that the algorithm can be implemented in a parallel way when there are several recursive calls (for instance in (Type Par)).

### Type of an Expression $M$ $(Type(E, M) = W)$

$$\text{(Type In)}\ \frac{Type(E, M) = Amb^Y[T, T']}{Type(E, in\ M) = Cap[{}^{\frown}\bot \rightsquigarrow \top]} \qquad \text{(Type Out)}\ \frac{Type(E, M) = Amb^Y[T, T']}{Type(E, out\ M) = Cap[{}^{\frown}\bot \rightsquigarrow \top]}$$

$$\text{(Type Open)}\ \frac{Type(E, M) = Amb^Y[T, T'] \quad Y \leq \circ}{Type(E, open\ M) = Cap[T']} \qquad \text{(Type } n\text{)}\ \frac{}{Type((E', n : W, E''), n) = W}$$

$$\text{(Type } \varepsilon\text{)}\ \frac{}{Type(E, \varepsilon) = Cap[{}^{\downarrow}\bot \rightsquigarrow \top]} \qquad \text{(Type Imm)}\ \frac{}{Type(E, imm) = Cap[{}^{\veebar}\bot \rightsquigarrow \top]}$$

$$\text{(Type .)}\ \frac{Type(E, M) = Cap[T] \quad Type(E, M') = Cap[T']}{Type(E, M.M') = Cap[T \vee T']}$$

For each message type, we always return the minimal type required by this capability (for example, $Cap[{}^{\frown}\bot \rightsquigarrow \top]$ for *in* $M$). In (Type Open), we return the maximal effects $T'$ which can appear when opening an ambient of that name and we check that this ambient is unlocked by $Y \leq \circ$.

**Type of a Process** $P$ **(**$Type(E, P) = T$**)**

$$\text{(Type Par)}\ \frac{Type(E,P) = T \quad Type(E,Q) = T' \quad T \vee T' \ valid}{Type(E, P \mid Q) = T \vee T'}$$

$$\text{(Type Action)}\ \frac{Type(E,M) = Cap[T] \quad Type(E,P) = T' \quad T \vee T' \ valid}{Type(E, M.P) = T \vee T'}$$

$$\text{(Type Zero)}\ \frac{}{Type(E,\mathbf{0}) = {}^{\lrcorner}\bot \rightsquigarrow \top} \qquad \text{(Type Repl)}\ \frac{Type(E,P) = T}{Type(E, !P) = T}$$

$$\text{(Type Res)}\ \frac{Type((E, n : Amb^Y[T_n, T'_n]), P) = T}{Type(E, (\nu n : Amb^Y[T_n, T'_n])P) = T}$$

$$\text{(Type Amb}_\circ\text{)}\ \frac{Type(E,M) = Amb^Y[T,T'] \quad Type(E,P) = T'' \quad T'' \leq T \quad Y \leq \circ}{Type(E, M[P]) = {}^{\lrcorner}\bot \rightsquigarrow \top}$$

$$\text{(Type Amb}_\bullet\text{)}\ \frac{Type(E,M) = Amb^Y[T,T'] \quad Type(E,P) = T'' \quad T'' \leq T \quad Y \leq \bullet}{Type(E, M[\![P]\!]) = {}^{\lrcorner}\bot \rightsquigarrow \top}$$

$$\text{(Type Input)}\ \frac{Type((E, n_1 : W_1, \ldots, n_k : W_k), P) = {}^Z O \rightsquigarrow I \quad O \leq W_1 \times \cdots \times W_k}{Type(E, (n_1 : W_1, \ldots, n_k : W_k).P) = {}^Z O \rightsquigarrow I \wedge W_1 \times \cdots \times W_k}$$

$$\text{(Type Output)}\ \frac{Type(E, M_1) = W_1 \quad \cdots \quad Type(E, M_k) = W_k}{Type(E, \langle M_1, \ldots, M_k \rangle) = {}^{\lrcorner}W_1 \times \cdots \times W_k \rightsquigarrow \top}$$

In (Type Par), we just take the join of the two sub-processes types, ensuring first that the resulting process is still valid.

In (Type Amb$_\circ$), we must check that the type $T''$ of $P$ is accepted by this ambient ($T'' \leq T$) and that the ambient can be opened ($Y \leq \circ$).

In (Type Input), we add the information of the input instruction by returning the meet of $I$ and $W_1 \times \cdots \times W_k$. We must also check that $O \leq W_1 \times \cdots \times W_k$ in $P$, to ensure that $O \leq I \wedge W_1 \times \cdots \times W_k$ in the resulting process.

In (Type Output), we just put the information of an output of type $W_1 \times \cdots \times W_k$. Since there is no continuation, there is nothing to check here.

The other rules are similar or (quite) natural.

## 4.3 Results

The first theorem shows that this algorithm is sound relatively to the type system we defined previously.

**Theorem 6 (Soundness)**

- *If* $Type(E, M) = W$*, then* $E \vdash M : W$.
- *If* $Type(E, P) = T$*, then* $E \vdash P : T$.

Then, we show that the algorithm is complete by checking that it succeeds when the term is typable, and that the returned type is less than the derived one.

**Theorem 7 (Completeness)**

- *If $E \vdash M : W$, then the algorithm succeeds on $M$ and $Type(E, M) \leq W$.*
- *If $E \vdash P : T$, then the algorithm succeeds on $P$ and $Type(E, P) \leq T$.*

From those two theorems, we easily deduce the property of minimal type for our type system and that the algorithm is able to compute it efficiently.

**Corollary 8 (Minimal Type)** *The set of all possible types for a typable expression or process has a minimum and this minimum is precisely the type returned by the algorithm.*

As a consequence, we can consider that the algorithm is valid: out of the returned type, we know all derivable types in our type system.

Finally, we can state a theorem similar to Theorem 3 concerning subject reduction:

**Theorem 9 (Subject Reduction)** *If $Type(E, P) = T$ and $P \rightarrow Q$, then the algorithm succeeds on $Q$ and $Type(E, Q) \leq T$.*

# Chapter 5

# Type Inference

In the previous chapter, we described a deterministic typing algorithm. This is a satisfactory result, to be compared to the undeterministic type system we had before. But, up to now, this algorithm performs only type-checking: the programmer must still annotate explicitly all ambient names and input variables with their types. To go one step further, the natural extension would be to remove these type annotations and try to design a type-inference algorithm. Unfortunately, even if at first sight this seems to require only minor modifications, some new and difficult problems appear if we want to keep the subtyping relation. So we will have to go a little back and restrict our problem to the original type system of CGG. For this system, we will show that it can be completely and efficiently solved.

## 5.1 Background

For the syntax, we will consider the calculus we studied since the beginning, that is with the two new constructs and the associated reduction rules (they do not bring any new difficulties). For the type system, we will nearly take the typing rules of CGG, as they are described in Appendix A. We modify them only to handle the new constructs. In practice, we add the rule:

$$\text{(CGG Exp Imm)}\ \frac{E \vdash_{CGG} \diamond}{E \vdash_{CGG} imm : Cap[\overset{\vee}{-}U]}$$

and we replace the rule (CGG Proc Amb) by two others:

$$\text{(CGG Proc Amb}_\circ\text{)}\ \frac{E \vdash_{CGG} M : Amb^\circ[T] \quad E \vdash_{CGG} P : T}{E \vdash_{CGG} M[P] : T'}$$

$$\text{(CGG Proc Amb}_\bullet\text{)}\ \frac{E \vdash_{CGG} M : Amb^\bullet[T] \quad E \vdash_{CGG} P : T}{E \vdash_{CGG} M[\![P]\!] : T'}$$

Instead of simply removing the type annotations, we keep them but allow to write type variables instead. For this, we must extend the definitions of types by adding an infinite set of variables for each of them. More generally, for the same letter, the lower case one will denote a type variable and the upper case one will denote a metavariable (as before).

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $Z$ | ::= | mobility | annotations | $Y$ | ::= | locking | annotations |
| | $\veebar$ | | immobile | | $\bullet$ | | locked |
| | $\curvearrowright$ | | mobile | | $\circ$ | | unlocked |
| | $z$ | | variable | | $y$ | | variable |

| | | | |
|---|---|---|---|
| $U$ | ::= | exchange types | |
| | $Shh$ | | silent |
| | $W_1 \times \cdots \times W_k \quad (k \geq 0)$ | | tuple |
| | $u$ | | variable |

| | | | |
|---|---|---|---|
| $T$ | ::= | process types | |
| | ${}^{Z}U$ | | process |
| | $t$ | | variable |

| | | | |
|---|---|---|---|
| $W$ | ::= | message types | |
| | $Amb^{Y}[T]$ | | ambient name |
| | $Cap[T]$ | | capability |
| | $w$ | | variable |

Now we can write expressions like $(x : w).P$ or $(\nu n : Amb^{y}[t])P$, or even $(x : Cap[{}^{\curvearrowright}u]).P$. In fact, we allow to mix both type variables and explicit types in a same term or even in a same type expression. By this mean, we get a more generic algorithm, and this property can be useful in practice: for example, if you want to check an unsecure code, you should be able to constraint some of its types by specifying them explicitly before applying the type-inference algorithm. Note also that one can express equality constraints between types just by using the same variable: in $(x : w).P \mid (y : w).Q$, the input variables $x$ and $y$ must have the same type.

## 5.2 Substitutions and Principal Unifier

**Definition 10** *A* substitution *is a total map from the set of all type variables (of any kind) to types of the same kind. We will denote them by the letters $\sigma$, $\theta$, $\rho$... The notation $\{x_1 \to X_1, \ldots, x_k \to X_k\}$ represents the substitution $\sigma$ such that $\sigma(x_i) = X_i$ for all $i$ and $\sigma(x) = x$ for all other type variables $x$. The empty substitution $\{\}$ is the identity function and will be noted* **id**. *Finally, the composition of substitutions is defined in exactly the same way as functions.*

We extend naturally substitutions to complex types (and not only type variables), to processes (by replacing type annotations in input and restriction constructions) and to environments. More precisely:

$$\sigma(\veebar) = \veebar \qquad \sigma(\bullet) = \bullet \qquad \sigma(Shh) = Shh$$
$$\sigma(\curvearrowright) = \curvearrowright \qquad \sigma(\circ) = \circ \qquad \sigma(W_1 \times \cdots \times W_k) = \sigma(W_1) \times \cdots \times \sigma(W_k)$$

$$\sigma({}^{Z}U) = {}^{\sigma(Z)}\sigma(U)$$

$$\sigma(Amb^{Y}[T]) = Amb^{\sigma(Y)}[\sigma(T)] \qquad \sigma(\emptyset) = \emptyset$$
$$\sigma(Cap[T]) = Cap[\sigma(T)] \qquad \sigma(E, n : W) = \sigma(E), n : \sigma(W)$$

$$\sigma(\mathbf{0}) = \mathbf{0} \qquad \sigma(P \mid Q) = \sigma(P) \mid \sigma(Q)$$
$$\sigma(!P) = !\,\sigma(P) \qquad \sigma(M.P) = M.\sigma(P)$$
$$\sigma(M[P]) = M[\sigma(P)] \qquad \sigma(M[\![P]\!]) = M[\![\sigma(P)]\!]$$

$$\sigma((\nu n : Amb^Y[T])P) = (\nu n : Amb^{\sigma(Y)}[\sigma(T)])\sigma(P)$$
$$\sigma((n_1 : W_1, \ldots, n_k : W_k).P) = (n_1 : \sigma(W_1), \ldots, n_k : \sigma(W_k)).\sigma(P)$$
$$\sigma(\langle M_1, \ldots, M_k \rangle) = \langle M_1, \ldots, M_k \rangle$$

The following proposition is essential for the algorithm. It states that a typing derivation remains valid if we replace some of the type variables in every place it appears.

**Proposition 11** *For any substitution $\sigma$, we have:*

- *If $E \vdash_{CGG} P : T$, then $\sigma(E) \vdash_{CGG} \sigma(P) : \sigma(T)$.*
- *If $E \vdash_{CGG} M : W$, then $\sigma(E) \vdash_{CGG} M : \sigma(W)$.*

An *unifier* of two types $X_1$ and $X_2$ is a substitution $\sigma$ such that $\sigma(X_1) = \sigma(X_2)$. Since the types in our system are simple trees, we know that there is a sound and complete unification algorithm for those types. It returns the *principal unifier* of two types (when it exists; otherwise it fails). We will call it $mgu(.,.)$ (it is not difficult to write its rules explicitly). The two following classical propositions state the properties we need:

**Proposition 12 (Soundness)** *If $mgu(X, Y) = \rho$, then $\rho(X) = \rho(Y)$.*

**Proposition 13 (Completeness)** *If $\sigma(X) = \sigma(Y)$, then $mgu(X, Y)$ succeeds, returning $\rho$, and there exists a substitution $\rho'$ such that $\rho'\rho = \sigma$.*

## 5.3 The Algorithm

We can know give the rules of the typing algorithm. They are used to infer judgements of the forms $Infer(E, M) = (W, \sigma)$ and $Infer(E, P) = (T, \sigma)$, where $W$ or $T$ is the most generic type possible for $M$ or $P$ (possibly containing type variables), and $\sigma$ is a substitution representing the constraints on the type variables in $M$ or $P$.

In the following rules, the premises must be read (and applied) from left to right. We do not detail how the algorithm gets *new* type variables. We will only consider that whenever a variable is declared **new**, it is different from all type variables previously used. In practice this can be achieved by using a global counter to number new type variables.

### Type-Inference for an Expression $M$ ($Infer(E, M)$)

For any well-formed environment $E$, we define:

$$(\text{Infer } \varepsilon)\ \frac{t \textbf{ new}}{Infer(E, \varepsilon) = (Cap[t], \mathbf{id})}$$

$$(\text{Infer .})\ \frac{\begin{array}{c} Infer(E, M) = (W, \sigma) \quad t \textbf{ new} \quad mgu(W, Cap[t]) = \rho \\ Infer(\rho\sigma(E), M') = (W', \sigma') \quad mgu(W', Cap[\sigma'\rho(t)]) = \rho' \end{array}}{Infer(E, M.M') = (\rho'(W'), \rho'\sigma'\rho\sigma)}$$

$$(\text{Infer In})\ \frac{Infer(E, M) = (W, \sigma) \quad y, t \textbf{ new} \quad mgu(W, Amb^y[t]) = \rho \quad u \textbf{ new}}{Infer(E, in\ M) = (Cap[\frown u], \rho\sigma)}$$

$$\text{(Infer Out)}\quad \frac{Infer(E,M)=(W,\sigma) \quad y,t\ \mathbf{new} \quad mgu(W,Amb^{y}[t])=\rho \quad u\ \mathbf{new}}{Infer(E,out\ M)=(Cap[\curvearrowright u],\rho\sigma)}$$

$$\text{(Infer Open)}\quad \frac{Infer(E,M)=(W,\sigma) \quad t\ \mathbf{new} \quad mgu(W,Amb^{\circ}[t])=\rho}{Infer(E,open\ M)=(Cap[\rho(t)],\rho\sigma)}$$

$$\text{(Infer Imm)}\quad \frac{u\ \mathbf{new}}{Infer(E,imm)=(Cap[\veebar u],\mathbf{id})}$$

$$\text{(Infer }n\text{)}\quad \frac{}{Infer((E,n:W,E'),n)=(W,\mathbf{id})}$$

**Type-Inference for a Process $P$ ($Infer(E,P)$)**

$$\text{(Infer Par)}\quad \frac{Infer(E,P)=(T,\sigma) \quad Infer(\sigma(E),\sigma(Q))=(T',\sigma') \quad mgu(\sigma'(T),T')=\rho}{Infer(E,P \mid Q)=(\rho(T'),\rho\sigma'\sigma)}$$

$$\text{(Infer Action)}\quad \frac{Infer(E,M)=(W,\sigma) \quad Infer(\sigma(E),\sigma(P))=(T,\sigma') \quad mgu(\sigma'(W),Cap[T])=\rho}{Infer(E,M.P)=(\rho(T),\rho\sigma'\sigma)}$$

$$\text{(Infer Zero)}\quad \frac{t\ \mathbf{new}}{Infer(E,\mathbf{0})=(t,\mathbf{id})} \qquad \text{(Infer Repl)}\quad \frac{Infer(E,P)=(T,\sigma)}{Infer(E,!P)=(T,\sigma)}$$

$$\text{(Infer Res)}\quad \frac{Infer((E,n:Amb^{Y}[T]),P)=(T',\sigma)}{Infer(E,(\nu n:Amb^{Y}[T])P)=(T',\sigma)}$$

$$\text{(Infer Amb}_\circ\text{)}\quad \frac{\begin{array}{c}Infer(E,M)=(W,\sigma) \quad Infer(\sigma(E),\sigma(P))=(T,\sigma') \\ mgu(\sigma'(W),Amb^{\circ}[T])=\rho \quad t\ \mathbf{new}\end{array}}{Infer(E,M[P])=(t,\rho\sigma'\sigma)}$$

$$\text{(Infer Amb}_\bullet\text{)}\quad \frac{\begin{array}{c}Infer(E,M)=(W,\sigma) \quad Infer(\sigma(E),\sigma(P))=(T,\sigma') \\ mgu(\sigma'(W),Amb^{\bullet}[T])=\rho \quad t\ \mathbf{new}\end{array}}{Infer(E,M[\![P]\!])=(t,\rho\sigma'\sigma)}$$

$$\text{(Infer Input)}\quad \frac{\begin{array}{c}Infer((E,n_1:W_1,\ldots,n_k:W_k),P)=(T,\sigma) \\ z\ \mathbf{new} \quad mgu(T,{}^{z}\sigma(W_1)\times\cdots\times\sigma(W_k))=\rho\end{array}}{Infer(E,(n_1:W_1,\ldots,n_k:W_k).P)=(\rho(T),\rho\sigma)}$$

$$\text{(Infer Output)}\quad \frac{\begin{array}{c}Infer(E,M_1)=(W_1,\sigma_1) \quad Infer(\sigma_1(E),M_2)=(W_2,\sigma_2) \\ \cdots \quad Infer(\sigma_{k-1}\ldots\sigma_1(E),M_k)=(W_k,\sigma_k) \quad z\ \mathbf{new}\end{array}}{Infer(E,\langle M_1,\ldots,M_k\rangle)=({}^{z}\sigma_k\ldots\sigma_2(W_1)\times\cdots\times\sigma_k(W_{k-1})\times W_k,\sigma_k\ldots\sigma_1)}$$

## 5.4 Results

The soundness of this inference algorithm is demonstrated by the following result:

**Theorem 14 (Soundness)** *If $Infer(E, P) = (T, \sigma)$, then $\sigma(E) \vdash_{CGG} \sigma(P) : T$.*

Using Proposition 11, we can then derive $\sigma'\sigma(E) \vdash_{CGG} \sigma'\sigma(P) : \sigma'(T)$ for any substitution $\sigma'$. We will say that all these derivations are *solutions* returned by the inference algorithm.

Then, we show that the algorithm is complete and returns a principal type (when it exists):

**Theorem 15 (Completeness)** *If there exists a type $T$ such that $\sigma(E) \vdash_{CGG} \sigma(P) : T$ (i.e. if the process $P$ is typable in the environment $E$ after performing some substitutions on type variables), the inference algorithm $Infer(E, P)$ succeeds and $\sigma(E) \vdash_{CGG} \sigma(P) : T$ is one of the returned solutions.*

## 5.5 Type Inference with Subtyping

Returning back to the original problem, can we do the same as above with the type system with subtyping ? Up to now, this problem remains open. Adding subtyping brings many problems, mainly because there is no minimal type for ambient names, but infinitely many minimals (all $Amb^{\perp_\circ}[T, T]$ for every possible $T$). Another problem comes from the ordering constraints we get due to ambient names and valid processes. Suppose we want to type $n$. We should give it all possible types $Amb^y[t, t']$ with $t \leq t'$, which is not expressible only with substitutions as we did before. And if we give it only minimal types ($Amb^y[t, t]$ for every $t$), using implicitly the subsumption rules to get the others, then *open* $n$ has every possible type $Cap[t]$ for every $t$, which would be equivalent to say it has the maximal type $Cap[^{\vec{\top}}\top \rightsquigarrow \perp]$...

To put it in a nutshell, the same approach as before is not possible. However, it is already possible to build a set of constraints on type variables. A solution would be an affectation of all type variables to types without variables such that all the constraints are satisfied. To do this, we start by defining the type system:

| | | | |
|---|---|---|---|
| $Z$ | $::=$ | mobility | annotations |
| | $\vec{\perp}$ | | mobility unknown |
| | $\underline{\vee}$ | | immobile |
| | $\curvearrowright$ | | mobile |
| | $\vec{\top}$ | | mobile and immobile |
| | $z$ | | variable |

| | | | |
|---|---|---|---|
| $Y$ | $::=$ | locking | annotations |
| | $\perp_\circ$ | | locking bottom |
| | $\bullet$ | | locked |
| | $\circ$ | | unlocked |
| | $\top^\circ$ | | locking top |
| | $y$ | | variable |

| | | | |
|---|---|---|---|
| $O, I$ | $::=$ | input/output types | |
| | $\perp$ | | bottom value |
| | $W_1 \times \cdots \times W_k \quad (k \geq 0)$ | | tuple |
| | $\top$ | | top value |
| | $o, i$ | | variables |

| | | | |
|---|---|---|---|
| $T$ | $::=$ | process types | |
| | $^Z O \rightsquigarrow I$ | | process |
| | $t$ | | variable |

| | | | |
|---|---|---|---|
| $W$ | $::=$ | message types | |
| | $Amb^Y[T, T']$ with $T \leq T'$ | | ambient name |
| | $Cap[T]$ | | capability |
| | $w$ | | variable |

We now return a value $S$ representing the constraints the type variables must satisfy. There are now two judgements:

- $Constr(E, P) = (T, S)$
- $Constr(E, M) = (W, S)$

$S$ can contain equalities, inequalities, comparisons beetween two types of the same kind, and expressions composed with $\vee$ and $\wedge$. Implicitly, every type variable appearing in the following rules is a new type variable (generated out of a global counter for instance).

$$\text{(Constr } \varepsilon) \quad \frac{}{Constr(E, \varepsilon) = (Cap[{}^z o \rightsquigarrow i], \{\})}$$

$$\text{(Constr .)} \quad \frac{Constr(E, M) = (W, S) \quad Constr(E, M') = (W', S')}{Constr(E, M.M') = (Cap[t], S \cup S' \cup \{W \leq Cap[t], W' \leq Cap[t]\})}$$

$$\text{(Constr In)} \quad \frac{Constr(E, M) = (W, S)}{Constr(E, in\ M) = (Cap[{}^z o \rightsquigarrow i], S \cup \{z \geq \curvearrowright, W \leq Amb^y[t, t'], t \leq t'\})}$$

$$\text{(Constr Out)} \quad \frac{Constr(E, M) = (W, S)}{Constr(E, out\ M) = (Cap[{}^z o \rightsquigarrow i], S \cup \{z \geq \curvearrowright, W \leq Amb^y[t, t'], t \leq t'\})}$$

$$\text{(Constr Open)} \quad \frac{Constr(E, M) = (W, S)}{Constr(E, open\ M) = (Cap[t'], S \cup \{W \leq Amb^{\circ}[t, t'], t \leq t'\})}$$

$$\text{(Constr Imm)} \quad \frac{}{Constr(E, imm) = (Cap[{}^z o \rightsquigarrow i], \{z \geq \underline{\vee}\})}$$

$$\text{(Constr } n) \quad \frac{}{Constr((E, n : W, E'), n) = (W, \{\})}$$

$$\text{(Constr Par)} \quad \frac{Constr(E, P) = (T, S) \quad Constr(E, Q) = (T', S')}{Constr(E, P \mid Q) = (t, S \cup S' \cup \{T \leq t, T' \leq t, t = {}^z o \rightsquigarrow i, z \neq \top, o \leq i\})}$$

$$\text{(Constr Action)} \quad \frac{Constr(E, M) = (W, S) \quad Constr(E, P) = (T, S')}{Constr(E, M.P) = (t, S \cup S' \cup \{W \leq Cap[t], T \leq t, t = {}^z o \rightsquigarrow i, z \neq \top, o \leq i\})}$$

$$\text{(Constr Zero)} \quad \frac{}{Constr(E, \mathbf{0}) = ({}^z o \rightsquigarrow i, \{z \neq \top, o \leq i\})}$$

$$\text{(Constr Repl)} \quad \frac{Constr(E, P) = (T, S)}{Constr(E, !P) = (T, S)}$$

$$\text{(Constr Res)} \quad \frac{Constr((E, n : Amb^{Y_n}[T_n, T'_n]), P) = (T, S)}{Constr(E, (\nu n : Amb^{Y_n}[T_n, T'_n])P) = (T, S \cup \{T_n \leq T'_n\})}$$

$$\text{(Constr Amb}_\circ\text{)}\ \frac{Constr(E,M) = (W,S) \quad Constr(E,P) = (T,S')}{Constr(E,M[P]) = ({}^{z}o \leadsto i, S \cup S' \cup \{W \le Amb^{\circ}[T,t'], T \le t', z \ne \overset{\rightarrow}{\top}, o \le i\})}$$

$$\text{(Constr Amb}_\bullet\text{)}\ \frac{Constr(E,M) = (W,S) \quad Constr(E,P) = (T,S')}{Constr(E,M[\![P]\!]) = ({}^{z}o \leadsto i, S \cup S' \cup \{W \le Amb^{\bullet}[T,t'], T \le t', z \ne \overset{\rightarrow}{\top}, o \le i\})}$$

$$\text{(Constr Input)}\ \frac{Constr((E, n_1 : W_1, \ldots, n_k : W_k), P) = (T,S)}{\begin{array}{l} Constr(E, (n_1 : W_1, \ldots, n_k : W_k).P) = \\ \qquad ({}^{z}o \leadsto i, S \cup \{T \le {}^{z}o \leadsto i, z \ne \overset{\rightarrow}{\top}, o \le i, i \le W_1 \times \cdots \times W_k\}) \end{array}}$$

$$\text{(Constr Output)}\ \frac{Constr(E,M_1) = (W_1,S_1) \quad \cdots \quad Constr(E,M_k) = (W_k,S_k)}{\begin{array}{l} Constr(E, \langle M_1, \ldots, M_k \rangle) = \\ \qquad ({}^{z}o \leadsto i, \bigcup_{i=1}^{k} S_i \cup \{z \ne \overset{\rightarrow}{\top}, o \le i, W_1 \times \cdots \times W_k \le o\}) \end{array}}$$

It would be quite easy to show that a process is typable if and only if the corresponding set of constraints has got a solution. But now we face an other problem: is there a way to solve algorithmically this set of constraints ? Up to now, all our attempts failed...

Some similar problems (absence of a minimal type) appeared in the type system of Abadi and Cardelli for object calculus. In this case, Jens Palsberg gave a solution in [Pal95], by building a graph of constraints and checking some properties on it. Maybe the same approach would be possible with the ambient calculus, but our attempts in this way also failed because the ordering relation is much more complex than in the object calculus.

# Chapter 6

# Conclusion

We have extended the previous type system for mobile ambients with new types and with a subtyping relation. We gave the corresponding typing rules and deduced a type-checking algorithm. We also gave a type-inference algorithm for CGG, but the problem of solving the constraints set in the system with subtyping remains open.

These algorithms are efficient and could be implemented quite easily. To our knowledge, there are two "implementations" of ambients so far: a Java applet from L. Cardelli and a translation into the join-calculus in a modified version of Objective Caml ([FS99]). None of them use types for now.

An other primitive was introduced by Cardelli-Ghelli-Gordon in [CGG99a, CGG99b]: the primitive *go*, which performs objective moves. To prevent the dangerous effects we discussed in Chapter 2, they extended the type system so that the type of an ambient name says explicitly if the ambient allows objective moves. We did not keep this primitive to simplify the notations for ambient names, but we checked that all our work and algorithms could be easily extended so as to include *go*.

## Acknowledgements

This work is the result of a two-months internship in the University of Turin, Italy. I would like to express my gratitude to Mariangiola Dezani, who supervised the internship and proposed the subject. Many thanks also to Paola Giannini, Ferruccio Damiani and all the "Lambda Group" in general for their welcome in the laboratory and for some very useful suggestions and pointers. Last but not least, thanks to Giorgio Ghelli and Luigi Liquori for their encouragements.

# Bibliography

[Car99a] L. Cardelli. Abstractions for Mobile Computation. *Secure Internet Programming: Security Issues for Distributed and Mobile Objects*, 1999.

[Car99b] L. Cardelli. Wide Area Computation. In *ICALP'99 Invited Paper*, April 1999.

[CG97] L. Cardelli and A. D. Gordon. A Calculus of mobile Ambients. 1997. Slides.

[CG98] L. Cardelli and A. D. Gordon. Mobile Ambients. In *Proceedings FoSSaCS'98*, volume LNCS 1378, pages 140–155. Springer, 1998.

[CG99] L. Cardelli and A. D. Gordon. Types for Mobile Ambients. In *Proceedings of the 26th ACM Symposium on Principles of Programming Languages*, pages 79–92. ACM, January 1999.

[CGG99a] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility Types for Mobile Ambients. Technical Report MSR-TR-99-32, Microsoft Research, June 1999.

[CGG99b] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility Types for Mobile Ambients. In *Proceedings of ICALP'99*, volume LNCS, April 1999.

[FS99] C. Fournet and A. Schmitt. An Implementation of Ambients in JoCaml. In *Proceedings MOS'99*, April 1999.

[Mil91] R. Milner. The Polyadic $\pi$-Calculus: a Tutorial. Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[Pal95] J. Palsberg. Efficient Inference of Object Types. *Information and Computation*, 1995.

[YH99] N. Yoshida and M. Hennessy. Subtyping and Locality in Distributed Higher Order Processes. Technical Report 01/99, University of Sussex, May 1999.

A lot of papers and slides about mobile ambients can also be downloaded on the web page of L. Cardelli : `http://www.luca.demon.co.uk`.

# Appendix A

# The Type System of Cardelli-Gordon-Ghelli

This annex presents the original type system, as described by Cardelli-Ghelli-Gordon in [CGG99a] and [CGG99b]. For convenience with other notations in our study, we only did one minor modification in the syntax of types: a generic process type ${}^{Z}U$ will be noted $T$. The typing judgements in this system are noted $E \vdash_{CGG} \diamond$, $E \vdash_{CGG} M : W$ and $E \vdash_{CGG} P : T$.

## A.1 Types

| | | | |
|---|---|---|---|
| $Z$ | $::=$ | mobility | annotations |
| | $\curlyvee$ | | immobile |
| | $\curvearrowright$ | | mobile |

| | | | |
|---|---|---|---|
| $Y$ | $::=$ | locking | annotations |
| | $\bullet$ | | locked |
| | $\circ$ | | unlocked |

| | | |
|---|---|---|
| $U$ | $::=$ | exchange types |
| | $Shh$ | silent |
| | $W_1 \times \cdots \times W_k \quad (k \geq 0)$ | tuple |

| | | |
|---|---|---|
| $T$ | $::=$ | process types |
| | ${}^{Z}U$ | process |

| | | |
|---|---|---|
| $W$ | $::=$ | message types |
| | $Amb^{Y}[T]$ | ambient name |
| | $Cap[T]$ | capability |

## A.2 Typing Rules

$$\text{(CGG Env } \emptyset\text{)} \; \frac{}{\emptyset \vdash_{CGG} \diamond} \qquad \text{(CGG Env } n\text{)} \; \frac{E \vdash_{CGG} \diamond \quad n \notin dom(E)}{E, n : W \vdash_{CGG} \diamond}$$

$$\text{(CGG Exp } \varepsilon\text{)} \; \frac{E \vdash_{CGG} \diamond}{E \vdash_{CGG} \varepsilon : Cap[T]} \qquad \text{(CGG Exp .)} \; \frac{E \vdash_{CGG} M : Cap[T] \quad E \vdash_{CGG} M' : Cap[T]}{E \vdash_{CGG} M.M' : Cap[T]}$$

$$\text{(CGG Exp In)} \; \frac{E \vdash_{CGG} M : Amb^{Y}[T]}{E \vdash_{CGG} in\ M : Cap[{}^{\curvearrowright}U]} \qquad \text{(CGG Exp Out)} \; \frac{E \vdash_{CGG} M : Amb^{Y}[T]}{E \vdash_{CGG} out\ M : Cap[{}^{\curvearrowright}U]}$$

$$\text{(CGG Exp Open)}\ \frac{E \vdash_{CGG} M : Amb^{\circ}[T]}{E \vdash_{CGG} open\ M : Cap[T]} \qquad \text{(CGG Exp } n\text{)}\ \frac{E', n : W, E'' \vdash_{CGG} \diamond}{E', n : W, E'' \vdash_{CGG} n : W}$$

$$\text{(CGG Proc Par)}\ \frac{E \vdash_{CGG} P : T \quad E \vdash_{CGG} Q : T}{E \vdash_{CGG} P \mid Q : T} \qquad \text{(CGG Proc Repl)}\ \frac{E \vdash_{CGG} P : T}{E \vdash_{CGG} !P : T}$$

$$\text{(CGG Proc Action)}\ \frac{E \vdash_{CGG} M : Cap[T] \quad E \vdash_{CGG} P : T}{E \vdash_{CGG} M.P : T}$$

$$\text{(CGG Proc Res)}\ \frac{E, n : Amb^{Y}[T_n] \vdash_{CGG} P : T}{E \vdash_{CGG} (\nu n : Amb^{Y}[T_n])P : T} \qquad \text{(CGG Proc Zero)}\ \frac{E \vdash_{CGG} \diamond}{E \vdash_{CGG} \mathbf{0} : T}$$

$$\text{(CGG Proc Amb)}\ \frac{E \vdash_{CGG} M : Amb^{Y}[T] \quad E \vdash_{CGG} P : T}{E \vdash_{CGG} M[P] : T'}$$

$$\text{(CGG Proc Input)}\ \frac{E, n_1 : W_1, \ldots, n_k : W_k \vdash_{CGG} P : {}^{Z}W_1 \times \cdots \times W_k}{E \vdash_{CGG} (n_1 : W_1, \ldots, n_k : W_k).P : {}^{Z}W_1 \times \cdots \times W_k}$$

$$\text{(CGG Proc Output)}\ \frac{E \vdash_{CGG} M_1 : W_1 \quad \cdots \quad E \vdash_{CGG} M_k : W_k \quad (E \vdash_{CGG} \diamond \ \text{ if } \ k = 0)}{E \vdash_{CGG} \langle M_1, \ldots, M_k \rangle : {}^{Z}W_1 \times \cdots \times W_k}$$

# Appendix B

# Proofs

## B.1 Proof of Theorem 2

**Proof:** This is an immediate consequence by induction on the type rules for processes. $\square$

## B.2 Lemmas

Let $E \vdash J$ denote any of the three judgements:

- $E \vdash \diamond$
- $E \vdash M : W$
- $E \vdash P : T$

and we extend naturally substitutions to judgements.

After every use of a typing rule, there can be usings of subsumption rules (in a finite number because we consider only finite derivations). Because the order is transitive, we can always replace them by one single use of a subsumption rule. And because the order is reflexive (and using the Theorem 2 for (Proc Sub)), we can always add one useless subsumption rule after each use of an other typing rule. So in the following proofs, when appropriate (in Lemma 17, (Subject Congruence) and (Subject Reduction)), we will always consider that there was exactly one subsumption rule just after each other typing rule.

**Lemma 16** *If $E', n : W, E'' \vdash J$, then $n \notin dom(E', E'')$.*

**Proof:** The only way to introduce variables in the environment are the rules (Proc Res) and (Proc Input). Since we can rename the ambients and the input variables, we can consider that they are all different. Then, there can only be one single binding for $n$ in an environment. $\square$

**Lemma 17** *If $E \vdash n : W$, then $E = E', n : W', E''$ with $n \notin dom(E', E'')$ and $W' \leq W$.*

**Proof:** $E \vdash n : W$ must have been derived from (Exp Sub) and (Exp $n$) with $E = E', n : W, E''$, $E \vdash n : W'$ and $W' \leq W$. Then, by Lemma 16, we have $n \notin dom(E', E'')$. $\square$

**Lemma 18 (Implied judgement)** *If $E', E'' \vdash J$, then $E' \vdash \diamond$.*

**Proof:** By induction on $E'$.

- If $E' = \emptyset$, then $E' \vdash \diamond$ by (Env $\emptyset$).
- If $E' = E'_1, n : W$, then we have $E'_1, n : W, E'' \vdash J$. By induction hypothesis, we have $E'_1 \vdash \diamond$. By Lemma 16, $n \notin dom(E'_1)$. Then, we can derive $E' \vdash \diamond$ by (Env $n$).

$\square$

**Lemma 19 (Exchange)** *If $E', n : W', m : W'', E'' \vdash J$, then $E', m : W'', n : W', E'' \vdash J$.*

**Proof:** The only way to get values from environments is to use (Exp $n$). Then Lemma 17 implies that $E', n : W', m : W'', E''$ and $E', m : W'', n : W', E''$ are observationnally equivalent (i.e. for a same name, the returned types are the same). And it is not difficult to check that if $E', n : W', m : W'', E'' \vdash \diamond$, then $E', m : W'', n : W', E'' \vdash \diamond$ by using (Env $n$) repeatedly. $\square$

**Lemma 20 (Weakening)** *If $E', E'' \vdash J$ and $n \notin dom(E', E'')$, then $E', n : W, E'' \vdash J$.*

**Lemma 21 (Strengthening)** *If $E', n : W, E'' \vdash J$ and $n \notin fn(J)$, then $E', E'' \vdash J$.*

**Lemma 22 (Substitution)** *If $E, n : W, E' \vdash J$ and $E \vdash M : W$, then $E, E' \vdash J\{n \leftarrow M\}$.*

**Proof:** By induction on the derivation of $E, n : W, E' \vdash J$.

**(Env $\emptyset$) and (Env $n$)** Since $\diamond\{n \leftarrow M\} = \diamond$, we obtain the result by Lemma 21.

**(Exp $\varepsilon$)** Since $\varepsilon\{n \leftarrow M\} = \varepsilon$, we obtain the result by Lemma 21.

**(Exp .)** We have $E, n : W, E' \vdash M_1.M_2 : Cap[T]$ derived from $E, n : W, E' \vdash M_1 : Cap[T]$ and $E, n : W, E' \vdash M_2 : Cap[T]$. By induction hypothesis, we have $E, E' \vdash M_1\{n \leftarrow M\} : Cap[T]$ and $E, E' \vdash M_2\{n \leftarrow M\} : Cap[T]$. Since $(M_1.M_2)\ \{n \leftarrow M\} = M_1\{n \leftarrow M\}\ .\ M_2\{n \leftarrow M\}$, by (Exp .), we have $E, E' \vdash (M_1.M_2)\ \{n \leftarrow M\} : Cap[T]$.

**(Exp In)** We have $E, n : W, E' \vdash in\ M' : Cap[\frown\bot \rightsquigarrow \top]$ derived from $E, n : W, E' \vdash M' : Amb^Y[T, T']$. By induction hypothesis, we have $E, E' \vdash M'\{n \leftarrow M\} : Amb^Y[T, T']$. Since $(in\ M')\ \{n \leftarrow M\} = in\ M'\{n \leftarrow M\}$, by (Exp In), we obtain $E, E' \vdash (in\ M')\ \{n \leftarrow M\} : Cap[\frown\bot \rightsquigarrow \top]$.

**(Exp Out)** Similar to (Exp In).

**(Exp Open)** We have $E, n : W, E' \vdash open\ M' : Cap[T']$ derived from $E, n : W, E' \vdash M' : Amb^\circ[T, T']$. By induction hypothesis, we have $E, E' \vdash M'\{n \leftarrow M\} : Amb^\circ[T, T']$. Since $(open\ M')\ \{n \leftarrow M\} = open\ M'\{n \leftarrow M\}$, by (Exp Open), we obtain $E, E' \vdash (open\ M')\ \{n \leftarrow M\} : Cap[T']$.

**(Exp Imm)** Since $imm\{n \leftarrow M\} = imm$, we obtain the result by Lemma 21.

**(Exp $n$)** We have $E, n : W, E' \vdash p : W'$. If $p \neq n$, then $p\ \{n \leftarrow M\} = p$ and the result comes immediately from Lemma 21. If $p = n$, then $W \leq W'$ by Lemma 17. We have $n\ \{n \leftarrow M\} = M$ and $E \vdash M : W$. By (Exp Sub), we obtain $E \vdash M : W'$. For every name $q$ in $E'$, we can say by Lemma 16 that $q \notin dom(E)$ and that all these names are different. So by Lemma 20, we have $E, E' \vdash n\ \{n \leftarrow M\} : W'$.

**(Exp Sub)** We have $E, n : W, E' \vdash M' : W'$ derived from $E, n : W, E' \vdash M' : W''$ with $W'' \leq W'$. By induction hypothesis, $E, E' \vdash M'\{n \leftarrow M\} : W''$. By (Exp Sub), we get $E, E' \vdash M'\{n \leftarrow M\} : W'$.

**(Proc Par)** We have $E, n : W, E' \vdash P \mid Q : T$ derived from $E, n : W, E' \vdash P : T$ and $E, n : W, E' \vdash Q : T$. By induction hypothesis, $E, E' \vdash P\{n \leftarrow M\} : T$ and $E, E' \vdash Q\{n \leftarrow M\} : T$. Since $(P \mid Q)\ \{n \leftarrow M\} = P\{n \leftarrow M\} \mid Q\{n \leftarrow M\}$, we can derive $E, E' \vdash (P \mid Q)\ \{n \leftarrow M\} : T$ by (Proc Par).

**(Proc Action)** Similar to (Proc Par) with an additional *Cap* constructor.

**(Proc Zero)** Since $\mathbf{0}\{n \leftarrow M\} = \mathbf{0}$, we obtain the result by Lemma 21.

**(Proc Repl)** Since $(!P)\{n \leftarrow M\} = !\ P\{n \leftarrow M\}$, the result comes easily from the induction hypothesis and an application of (Proc Repl).

**(Proc Res)** We have $E, n : W, E' \vdash (\nu p : Amb^Y[T_p, T'_p])P : T$ (we suppose that the new ambient name $p$ is renamed in order to have $p \neq n$ and $p \notin dom(E, E')$), derived from $E, n : W, E', p : Amb^Y[T_p, T'_p] \vdash P : T$. By induction hypothesis, we have $E, E', p : Amb^Y[T_p, T'_p] \vdash P\{n \leftarrow M\} : T$. Then, by (Proc Res), we obtain $E, E' \vdash ((\nu p : Amb^Y[T_p, T'_p])P)\ \{n \leftarrow M\} : T$.

**(Proc Amb$_\circ$)** We have $E, n : W, E' \vdash M'[P] : {}^{\lrcorner}\bot \rightsquigarrow \top$ derived from $E, n : W, E' \vdash M' : Amb^\circ[T, T']$ and $E, n : W, E' \vdash P : T$. By induction hypothesis, we have $E, E' \vdash M'\{n \leftarrow M\} : Amb^\circ[T, T']$ and $E, E' \vdash P\{n \leftarrow M\} : T$. Then, we can derive by (Proc Amb$_\circ$) that $E, E' \vdash (M'[P])\ \{n \leftarrow M\} : {}^{\lrcorner}\bot \rightsquigarrow \top$.

**(Proc Amb$_\bullet$)** Similar to (Proc Amb$_\circ$).

**(Proc Input)** We have $E, n : W, E' \vdash (n_1 : W_1, \ldots, n_k : W_k).P : {}^Z O \rightsquigarrow I$ (by renaming, we can suppose that $n \neq n_i$ for all $i$). This comes from $E, n : W, E', n_1 : W_1, \ldots, n_k : W_k \vdash P : {}^Z O \rightsquigarrow I$ and $I \leq W_1 \times \cdots \times W_k$. By induction hypothesis, we have $E, E', n_1 : W_1, \ldots, n_k : W_k \vdash P\{n \leftarrow M\} : {}^Z O \rightsquigarrow I$. We can then apply (Proc Input) and obtain $E, E' \vdash ((n_1 : W_1, \ldots, n_k : W_k).P)\ \{n \leftarrow M\} : {}^Z O \rightsquigarrow I$.

**(Proc Output)** We have $E, n : W, E' \vdash \langle M_1, \ldots, M_k \rangle : {}^{\lrcorner} W_1 \times \cdots \times W_k \rightsquigarrow \top$. If $k = 0$, then the result comes easily by Lemma 21. Otherwise, we have $E, n : W, E' \vdash M_i : W_i$ for all $i$. By induction hypothesis, we get $E, E' \vdash M_i\{n \leftarrow M\} : W_i$ for all $i$. Then, by (Proc Output), $E, E' \vdash \langle M_1, \ldots, M_k \rangle\ \{n \leftarrow M\} : {}^{\lrcorner} W_1 \times \cdots \times W_k \rightsquigarrow \top$.

**(Proc Sub)** We have $E, n : W, E' \vdash P : T$ derived from $E, n : W, E' \vdash P : T'$ with $T' \leq T$ and $T$ valid. By induction hypothesis, $E, E' \vdash P : T'$. Then, by (Proc Sub), $E, E' \vdash P : T$.

□

**Proposition 23 (Subject Congruence)** *If $E \vdash P : T$ and $P \equiv Q$ (or $Q \equiv P$), then $E \vdash Q : T$.*

**Proof:** By mutual induction on the derivations of $P \equiv Q$ and $Q \equiv P$ (we will not detail the case $Q \equiv P$ when the reasoning is the same, i.e. for symmetrical rules).

**(Struct Refl)** Trivial.

**(Struct Symm)** By an immediate induction.

**(Struct Trans)** Suppose $P \equiv Q$ was deduced from $P \equiv R$ and $R \equiv Q$. By induction hypothesis, we have $E \vdash R : T$, and then $E \vdash Q : T$.

**(Struct Res)** Let $P = (\nu n : Amb^Y[T_n, T'_n])P'$ and $Q = (\nu n : Amb^Y[T_n, T'_n])Q'$ with $P' \equiv Q'$. $E \vdash P : T$ must have been derived from (Proc Sub) and (Proc Res) with $E, n : Amb^Y[T_n, T'_n] \vdash P' : T'$, $T' \leq T$ and $T$ valid. By induction hypothesis, we have $E, n : Amb^Y[T_n, T'_n] \vdash Q' : T'$, and then $E \vdash Q : T$ by (Proc Res) and (Proc Sub).

**(Struct Par)** Let $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \equiv Q'$. We have $E \vdash P : T$ derived from (Proc Sub) and (Proc Par) with $E \vdash P' : T'$, $E \vdash R : T'$, $T' \leq T$ and $T$ valid. By induction hypothesis, we have $E \vdash Q' : T'$. We can then derive by (Proc Par) and (Proc Sub) that $E \vdash Q : T$.

**(Struct Repl)** Let $P = !P'$ and $Q = !Q'$ with $P' \equiv Q'$. $E \vdash P : T$ must have been derived from (Proc Sub) and (Proc Repl) with $E \vdash P' : T'$, $T' \leq T$ and $T$ valid. By induction hypothesis, we have $E \vdash Q' : T'$. Then, by (Proc Repl) and (Proc Sub), $E \vdash Q : T$.

**(Struct Amb$_\circ$)** Let $P = M[P']$ and $Q = M[Q']$ with $P' \equiv Q'$. $E \vdash P : T$ must have been derived from (Proc Sub) with $E \vdash P : {}^{\lrcorner}\bot \rightsquigarrow \top$ and $T$ valid, which was derived from (Proc Amb$_\circ$) with $E \vdash M : Amb^\circ[T_M, T'_M]$ and $E \vdash P' : T_M$. By induction hypothesis, we have $E \vdash Q' : T_M$. We can derive by (Proc Amb$_\circ$) that $E \vdash Q : {}^{\lrcorner}\bot \rightsquigarrow \top$. Then, $E \vdash Q : T$ by (Proc Sub).

**(Struct Amb$_\bullet$)** Similar to (Struct Amb$_\circ$).

**(Struct Action)** Similar to (Struct Par) with an additional $Cap$ constructor.

**(Struct Input)** Let $P = (n_1 : W_1, \ldots, n_k : W_k).P'$ and $Q = (n_1 : W_1, \ldots, n_k : W_k).Q'$ with $P' \equiv Q'$. We have $E \vdash P : T$ derived from (Proc Sub) and (Proc Input) with $E, n_1 : W_1, \ldots, n_k : W_k \vdash P' : {}^Z O \rightsquigarrow I$, with the conditions $I \leq W_1 \times \cdots \times W_k$, ${}^Z O \rightsquigarrow I \leq T$ and $T$ valid. By induction hypothesis, we have $E, n_1 : W_1, \ldots, n_k : W_k \vdash Q' : {}^Z O \rightsquigarrow I$. Then, by (Proc Input) and (Proc Sub), $E \vdash Q : T$.

**(Struct Par Comm)** Trivial because (Proc Par) is a symmetric rule.

**(Struct Par Assoc)** Let $P = (A \mid B) \mid C$ and $Q = A \mid (B \mid C)$. Suppose $E \vdash P : T$ and $P \equiv Q$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash A \mid B : T'$, $E \vdash C : T'$, $T' \leq T$ and $T$ valid. The first judgement must have been derived from (Proc Sub) and (Proc Par) with $E \vdash A : T''$, $E \vdash B : T''$, $T'' \leq T'$ and $T'$ valid. We can derive $E \vdash A : T'$ and $E \vdash B : T'$ by (Proc Sub). Then, we get $E \vdash B \mid C : T'$, followed by $E \vdash A \mid (B \mid C) : T'$ by (Proc Par). And finally $E \vdash Q : T$ by (Proc Sub). The case $Q \equiv P$ is similar.

**(Struct Repl Par)** Suppose $E \vdash\ !P : T$. This must have been derived from (Proc Sub) and (Proc Repl) with $E \vdash P : T'$, $T' \leq T$ and $T$ valid. Then, $E \vdash\ !P : T'$ by (Proc Repl). And $E \vdash P \mid !P : T$ by (Proc Par) and (Proc Sub). Reciprocally, suppose $E \vdash P \mid !P : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash P : T'$, $E \vdash\ !P : T'$, $T' \leq T$ and $T$ valid. From the second judgement, we get $E \vdash\ !P : T$ by (Proc Sub).

**(Struct Res Res)** Let $P = (\nu n : W)(\nu m : W')A$ and $Q = (\nu m : W')(\nu n : W)A$ with $n \neq m$. $E \vdash P : T$ must have been derived from (Proc Sub) and (Proc Res) with $E, n : W \vdash (\nu m : W')A : T'$, $T' \leq T$ and $T$ valid. This must have been derived from (Proc Sub) and (Proc Res) with $E, n : W, m : W' \vdash A : T''$, $T'' \leq T'$ and $T'$ valid. By Lemma 19, we have $E, m : W', n : W \vdash A : T''$. Then, using (Proc Res) and (Proc Sub) twice each, we get $E \vdash Q : T$.

**(Struct Res Par)** Let $P = (\nu n : W)(A \mid B)$ and $Q = A \mid (\nu n : W)B$ with $n \notin fn(A)$.

- Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Res) with $E, n : W \vdash A \mid B : T'$, $T' \leq T$ and $T$ valid. This must have been derived from (Proc Sub) and (Proc Par) with $E, n : W \vdash A : T''$, $E, n : W \vdash B : T''$, $T'' \leq T'$ and $T'$ valid. From the first judgement, we get $E \vdash A : T''$ by Lemma 21 since $n \notin fn(A)$. From the second judgement, we derive $E \vdash (\nu n : W)B : T''$ by (Proc Res). Then, we can derive by (Proc Par) that $E \vdash Q : T''$. Finally, $E \vdash Q : T$ by (Proc Sub) because $T'' \leq T$ and $T$ is valid.
- Reciprocally, suppose $E \vdash Q : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash A : T'$, $E \vdash (\nu n : W)B : T'$, $T' \leq T$ and $T$ valid. From the first judgement, we get $E, n : W \vdash A : T'$ by Lemma 20 and renaming. The second judgement must have been derived from (Proc Sub) and (Proc Res) with $E, n : W \vdash B : T''$, $T'' \leq T'$ and $T'$ valid. We have $E, n : W \vdash B : T'$ by (Proc Sub) and $E, n : W \vdash A \mid B : T'$ by (Proc Par). Finally, $E \vdash Q : T$ by (Proc Res) and (Proc Sub).

**(Struct Res Amb$_\circ$)** Let $P = (\nu n : W)m[A]$ and $Q = m[(\nu n : W)A]$ with $n \neq m$.

- Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Res) with $E, n : W \vdash m[A] : T'$, $T' \leq T$ and $T$ valid. This must have been derived from (Proc Sub) with $E, n : W \vdash m[A] : {}^{\lrcorner}\bot \rightsquigarrow \top$ and $T'$ valid, which was derived from (Proc Amb$_\circ$) with $E, n : W \vdash m : Amb^\circ[T_m, T'_m]$ and $E, n : W \vdash A : T_m$. By (Proc Res), we get $E \vdash (\nu n : W)A : T_m$. Since $n \neq m$, by Lemma 21, we have $E \vdash m : Amb^\circ[T_m, T'_m]$. Then, we can derive by (Proc Amb$_\circ$) that $E \vdash Q : {}^{\lrcorner}\bot \rightsquigarrow \top$. Finally, $E \vdash Q : T$ by (Proc Sub).
- Reciprocally, suppose $E \vdash Q : T$. This must have been derived from $E \vdash Q : {}^{\lrcorner}\bot \rightsquigarrow \top$ with $T$ valid, which was derived from (Proc Amb$_\circ$) with $E \vdash m : Amb^\circ[T_m, T'_m]$ and $E \vdash (\nu n : W)A : T_m$. The second judgement must have been derived from (Proc Sub) and (Proc Res) with $E, n : W \vdash A : T'$, $T' \leq T_m$ and $T_m$ valid. By (Proc Sub), we get $E, n : W \vdash A : T_m$ and by Lemma 20 and renaming, we have $E, n : W \vdash m : Amb^\circ[T_m, T'_m]$. Then, by (Proc Amb$_\circ$), we can derive $E, n : W \vdash m[A] : {}^{\lrcorner}\bot \rightsquigarrow \top$. Finally, $E \vdash P : T$ by (Proc Res) and (Proc Sub).

**(Struct Res Amb$_\bullet$)** Similar to (Struct Res Amb$_\circ$).

**(Struct Zero Par)** Suppose $E \vdash P \mid \mathbf{0} : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash P : T'$, $E \vdash \mathbf{0} : T'$, $T' \leq T$ and $T$ valid. Then, $E \vdash P : T$ by (Proc Sub). Reciprocally, suppose $E \vdash P : T$. By Lemma 18, $E \vdash \diamond$. Then, by (Proc Zero), $E \vdash \mathbf{0} : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$. By Theorem 2, $T$ is valid. We can derive $E \vdash \mathbf{0} : T$ by (Proc Sub). Finally, $E \vdash P \mid \mathbf{0} : T$ by (Proc Par).

**(Struct Zero Res)** Suppose $E \vdash (\nu n : Amb^Y[T_n, T'_n])\mathbf{0} : T$. By Lemma 18, $E \vdash \diamond$ and by Theorem 2, $T$ is valid. Then, we can derive $E \vdash \mathbf{0} : T$ by (Proc Zero) and (Proc Sub). Reciprocally, suppose $E \vdash \mathbf{0} : T$. By renaming, we can suppose $n \notin dom(E)$. Then, $E, n : Amb^Y[T_n, T'_n] \vdash \mathbf{0} : T$ by Lemma 20. Finally, $E \vdash (\nu n : Amb^Y[T_n, T'_n])\mathbf{0} : T$ by (Proc Res).

**(Struct Zero Repl)** Suppose $E \vdash !\mathbf{0} : T$. By Lemma 18, $E \vdash \diamond$, and $T$ is valid by Theorem 2. Then, $E \vdash \mathbf{0} : T$ by (Proc Zero) and (Proc Sub). Reciprocally, suppose $E \vdash \mathbf{0} : T$. We get $E \vdash !\mathbf{0} : T$ by (Proc Res).

**(Struct $\varepsilon$)** Similar to (Struct Zero Par).

**(Struct .)** Similar to (Struct Par Assoc).

□

## B.3 Proof of Theorem 3

**Proof:** By induction on the derivation of $P \rightarrow Q$.

**(Red In)** Let $P = n[in\ m.A \mid B] \mid m[C]$ and $Q = m[n[A \mid B] \mid C]$ (the proof with $[\![.]\!]$ in place of $[.]$ is similar). Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash n[in\ m.A \mid B] : T'$, $E \vdash m[C] : T'$, $T' \leq T$ and $T$ valid. Those two judgements must have been derived from (Proc Sub) with $E \vdash n[in\ m.A \mid B] : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$, $E \vdash m[C] : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$ and $T'$ valid, which were both derived from (Proc Amb$_\circ$) with $E \vdash n : Amb^\circ[T_n, T'_n]$, $E \vdash in\ m.A \mid B : T_n$, and with $E \vdash m : Amb^\circ[T_m, T'_m]$ and $E \vdash C : T_m$. Moreover, $E \vdash in\ m.A \mid B : T_n$ must have been derived from (Proc Sub) and (Proc Par) with $E \vdash in\ m.A : T''$, $E \vdash B : T''$, $T'' \leq T_n$ and $T_n$ valid. The first judgement must have been derived from (Proc Sub) and (Proc Action) with $E \vdash in\ m : Cap[T''']$, $E \vdash A : T'''$, $T''' \leq T''$ and $T''$ valid. Then, we have $E \vdash A : T''$ by (Proc Sub), $E \vdash A \mid B : T''$ by (Proc Par), and $E \vdash A \mid B : T_n$ by (Proc Sub). We can derive $E \vdash n[in\ m.A \mid B] : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$ by (Proc Amb$_\circ$). By Theorem 2 applied to $C$, $T_m$ is valid. So $E \vdash n[in\ m.A \mid B] : T_m$ by (Proc Sub). Then, $E \vdash n[in\ m.A \mid B] \mid C : T_m$ by (Proc Par). And $E \vdash Q : T$ by (Proc Amb$_\circ$) and (Proc Sub).

**(Red Out)** Let $P = m[n[out\ m.A \mid B] \mid C]$ and $Q = n[A \mid B] \mid m[C]$ (the proof with $[\![.]\!]$ in place of $[.]$ is similar). Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) with $E \vdash P : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$ and $T$ valid, which was derived from (Proc Amb$_\circ$) with $E \vdash m : Amb^\circ[T_m, T'_m]$ and $E \vdash n[out\ m.A \mid B] \mid C : T_m$. The second judgement must have been derived from (Proc Sub) and (Proc Par) with $E \vdash n[out\ m.A \mid B] : T'$, $E \vdash C : T'$, $T' \leq T_m$ and $T_m$ valid. The first judgement must have been derived from (Proc Sub) with $E \vdash n[out\ m.A \mid B] : {}^{\lrcorner}\!\!\bot \rightsquigarrow \top$ and $T'$ valid, which was again derived from (Proc Amb$_\circ$) with $E \vdash n : Amb^\circ[T_n, T'_n]$ and $E \vdash out\ m.A \mid B : T_n$. This second judgement must have been derived from (Proc Sub) and (Proc Par) with $E \vdash out\ m.A : T''$, $E \vdash B : T''$, $T'' \leq T_n$ and $T_n$ valid. The first judgement must have been derived from (Proc Sub) and (Proc Action) with $E \vdash out\ m : Cap[T''']$, $E \vdash A : T'''$, $T''' \leq T''$ and $T''$ valid. By (Proc Sub), we have $E \vdash A : T''$. By (Proc Par), $E \vdash A \mid B : T''$. By (Proc Sub), $E \vdash A \mid B : T_n$. By (Proc Amb$_\circ$) and (Proc Sub), $E \vdash n[A \mid B] : T$. Moreover, $E \vdash C : T_m$ by (Proc Sub). Then $E \vdash m[C] : T$ by (Proc Amb$_\circ$) and (Proc Sub). Finally, $E \vdash Q : T$ by (Proc Par).

**(Red Open)** Let $P = open\ n.A \mid n[B]$ and $Q = A \mid B$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash open\ n.A : T'$, $E \vdash n[B] : T'$, $T' \leq T$ and $T$ valid. The first judgement must have been derived from (Proc Sub) and (Proc Action) with $E \vdash open\ n : Cap[T'']$, $E \vdash A : T''$, $T'' \leq T'$ and $T'$ valid. Then, $E \vdash A : T$ by (Proc Sub) since $T'' \leq T$. The first judgement was derived from (Exp Sub) and (Exp Open) with $E \vdash n : Amb^{\circ}[T_1, T_1']$ and $T_1' \leq T''$. Moreover, the second judgement above ($E \vdash n[B] : T'$) must have been derived from (Proc Sub) with $E \vdash n[B] : {}^{\dashv}\bot \rightsquigarrow \top$ and $T'$ valid, which must have been derived from (Proc Amb$_\circ$) with $E \vdash n : Amb^{\circ}[T_2, T_2']$ and $E \vdash B : T_2$. By Lemma 17, $E = E_1, n : Amb^Y[T_n, T_n'], E_2$, and we have $Amb^Y[T_n, T_n'] \leq Amb^{\circ}[T_1, T_1']$ and $Amb^Y[T_n, T_n'] \leq Amb^{\circ}[T_2, T_2']$. This implies $T_2 \leq T_n \leq T_n' \leq T_1'$. Then $T_2 \leq T'' \leq T$ and we can derive $E \vdash B : T$ by (Proc Sub). Finally, $E \vdash Q : T$ by (Proc Par).

**(Red Imm)** Let $P = imm.P'$ and $Q = P'$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Action) with $E \vdash imm : Cap[T']$, $E \vdash P' : T'$, $T' \leq T$ and $T$ valid. Then, $E \vdash Q : T$ by (Proc Sub).

**(Red Comm)** Let $P = (n_1 : W_1, \ldots, n_k : W_k).P' \mid \langle M_1, \ldots, M_k \rangle$ and $Q = P'\{n_1 \leftarrow M_1, \ldots, n_k \leftarrow M_k\}$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash \langle M_1, \ldots, M_k \rangle : T'$, $E \vdash (n_1 : W_1, \ldots, n_k : W_k).P' : T'$, $T' \leq T$ and $T$ valid. Let us write $T' = {}^{Z'}O' \rightsquigarrow I'$. The first judgement must have been derived from (Proc Sub) with $E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\dashv}W_1' \times \cdots \times W_k' \rightsquigarrow \top$ with $W_1' \times \cdots \times W_k' \leq O'$, and this was derived from $E \vdash M_i : W_i'$ for $1 \leq i \leq k$. The second judgement above must have been derived from (Proc Sub) and (Proc Input) with $E, n_1 : W_1, \ldots, n_k : W_k \vdash P' : {}^{Z}O \rightsquigarrow I$, with the conditions $I \leq W_1 \times \cdots \times W_k$, ${}^{Z}O \rightsquigarrow I \leq T'$ and $T'$ valid. By Theorem 2, we have $O' \leq I'$. Then, we have the following inequalities: $W_1' \times \cdots \times W_k' \leq O' \leq I' \leq I \leq W_1 \times \cdots \times W_k$. This implies $W_i' \leq W_i$ for all $1 \leq i \leq k$. We can derive $E \vdash M_i : W_i$ by (Exp Sub). By $k$ applications of Lemma 22, we get $E \vdash P'\{n_1 \leftarrow M_1, \ldots, n_k \leftarrow M_k\} : {}^{Z}O \rightsquigarrow I$. Since ${}^{Z}O \rightsquigarrow I \leq T' \leq T$, we obtain $E \vdash Q : T$ by (Proc Sub).

**(Red Res)** Let $P = (\nu n : Amb^Y[T_n, T_n'])\ P'$ and $Q = (\nu n : Amb^Y[T_n, T_n'])\ Q'$ with $P' \to Q'$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Res) with $E, n : Amb^Y[T_n, T_n'] \vdash P' : T'$, $T' \leq T$ and $T$ valid. By induction hypothesis, we have $E, n : Amb^Y[T_n, T_n'] \vdash Q' : T'$. Then, by (Proc Res) and (Proc Sub), we can derive $E \vdash Q : T$.

**(Red Amb$_\circ$)** Let $P = n[P']$ and $Q = n[Q']$ with $P' \to Q'$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) with $E \vdash P : {}^{\dashv}\bot \rightsquigarrow \top$ and $T$ valid, which was derived from (Proc Amb$_\circ$) with $E \vdash n : Amb^{\circ}[T_n, T_n']$ and $E \vdash P' : T_n$. By induction hypothesis, we have $E \vdash Q' : T_n$. Then, we can derive by (Proc Amb$_\circ$) and (Proc Sub) that $E \vdash Q : T$.

**(Red Amb$_\bullet$)** Similar to (Red Amb$_\circ$).

**(Red Par)** Let $P = P' \mid R$ and $Q = Q' \mid R$ with $P' \to Q'$. Suppose $E \vdash P : T$. This must have been derived from (Proc Sub) and (Proc Par) with $E \vdash P' : T'$, $E \vdash R : T'$, $T' \leq T$ and $T$ valid. By induction hypothesis, we have $E \vdash Q' : T'$. Then, $E \vdash Q : T$ by (Proc Par) and (Proc Sub).

**(Red $\equiv$)** Let $P \equiv P'$ and $Q' \equiv Q$ with $P' \to Q'$. Suppose $E \vdash P : T$. Then by (Subject Congruence) (Proposition 23), we have $E \vdash P' : T$. By induction hypothesis, we have $E \vdash Q' : T$. Again by (Subject Congruence), we get $E \vdash Q : T$.

$\square$

## B.4 Proof of Theorem 4

**Proof:** By induction on the derivation in CGG.

**(CGG Env $\emptyset$)** We can always derive $f(\emptyset) \vdash \diamond$ since $f(\emptyset) = \emptyset$.

**(CGG Env $n$)** Suppose $E, n : W \vdash_{CGG} \diamond$ derived from $E \vdash_{CGG} \diamond$ and $n \notin dom(E)$. By induction hypothesis, we have $f(E) \vdash \diamond$. It is trivial to see that $n \notin dom(f(E))$. Then, we can derive $f(E), n : f(W) \vdash \diamond$. Finally, $f(E, n : W) \vdash \diamond$.

**(CGG Exp $\varepsilon$)** Suppose $E \vdash_{CGG} \varepsilon : Cap[{}^{Z}U]$ derived from $E \vdash_{CGG} \diamond$. By induction hypothesis, we have $f(E) \vdash \diamond$. Then, by (Exp $\varepsilon$), we can derive $f(E) \vdash \varepsilon : Cap[{}^{\dashv}\bot \rightsquigarrow \top]$. Finally, $f(E) \vdash \varepsilon : Cap[{}^{Z}f(U)] = f(Cap[{}^{Z}U])$ by (Exp Sub).

**(CGG Exp .)** Suppose $E \vdash_{CGG} M.M' : Cap[^Z U]$ derived from $E \vdash_{CGG} M : Cap[^Z U]$ and $E \vdash_{CGG} M' : Cap[^Z U]$. By induction hypothesis, we have $f(E) \vdash M : f(Cap[^Z U]) = Cap[^Z f(U)]$ and $f(E) \vdash M' : f(Cap[^Z U]) = Cap[^Z f(U)]$. By (Exp .), we can derive $f(E) \vdash M.M' : Cap[^Z f(U)] = f(Cap[^Z U])$.

**(CGG Exp In)** Suppose $E \vdash_{CGG} in\ M : Cap[^\frown U']$ derived from $E \vdash_{CGG} M : Amb^Y[^Z U]$. By induction hypothesis, we have $f(E) \vdash M : f(Amb^Y[^Z U]) = Amb^Y[^Z f(U), ^Z f(U)]$. Then, we can derive by (Exp In) $f(E) \vdash in\ M : Cap[^\frown \bot \rightsquigarrow \top]$. Finally, $f(E) \vdash in\ M : Cap[^\frown f(U')] = f(Cap[^\frown U'])$ by (Exp Sub).

**(CGG Exp Out)** Similar to (CGG Exp In).

**(CGG Exp Open)** Suppose $E \vdash_{CGG} open\ M : Cap[^Z U]$ derived from $E \vdash_{CGG} M : Amb^\circ[^Z U]$. By induction hypothesis, we have $f(E) \vdash M : f(Amb^\circ[^Z U]) = Amb^\circ[^Z f(U), ^Z f(U)]$. Then, we can derive by (Exp Open) $f(E) \vdash open\ M : Cap[^Z f(U)] = f(Cap[^Z U])$.

**(CGG Exp $n$)** Suppose $E, n : W, E' \vdash_{CGG} n : W$ derived from $E, n : W, E' \vdash_{CGG} \diamond$. By induction hypothesis, we have $f(E, n : W, E') \vdash \diamond$, that is to say $f(E), n : f(W), f(E') \vdash \diamond$. By (Exp $n$), we can then derive $f(E, n : W, E') \vdash n : f(W)$.

**(CGG Proc Par)** Suppose $E \vdash_{CGG} P \mid Q : {}^Z U$ derived from $E \vdash_{CGG} P : {}^Z U$ and $E \vdash_{CGG} Q : {}^Z U$. By induction hypothesis, we have $f(E) \vdash f(P) : {}^Z f(U)$ and $f(E) \vdash f(Q) : {}^Z f(U)$. Then, we can derive $f(E) \vdash f(P) \mid f(Q) : {}^Z f(U)$ by (Proc Par). And $f(P) \mid f(Q) = f(P \mid Q)$.

**(CGG Proc Action)** Suppose $E \vdash_{CGG} M.P : {}^Z U$ derived from $E \vdash_{CGG} M : Cap[^Z U]$ and $E \vdash_{CGG} P : {}^Z U$. By induction hypothesis, we have $f(E) \vdash M : f(Cap[^Z U]) = Cap[^Z f(U)]$ and $f(E) \vdash f(P) : {}^Z f(U)$. We can then derive by (Proc Action) that $f(E) \vdash M.f(P) : {}^Z f(U)$. And $M.f(P) = f(M.P)$.

**(CGG Proc Zero)** Suppose $E \vdash_{CGG} \mathbf{0} : {}^Z U$ derived from $E \vdash_{CGG} \diamond$. By induction hypothesis, we have $f(E) \vdash \diamond$. By (Proc Zero), we can derive $f(E) \vdash \mathbf{0} : {}^{\lrcorner}\bot \rightsquigarrow \top$. It is trivial to see that ${}^Z f(U)$ is valid by considering the definition of $f$. Then, since $\mathbf{0} = f(\mathbf{0})$, we get the result by (Proc Sub): $f(E) \vdash f(\mathbf{0}) : {}^Z f(U)$.

**(CGG Proc Repl)** Suppose $E \vdash_{CGG} !P : {}^Z U$ derived from $E \vdash_{CGG} P : {}^Z U$. By induction hypothesis, we have $f(E) \vdash f(P) : {}^Z f(U)$. With $f(!P) = !f(P)$, we can derive by (Proc Repl) that $f(E) \vdash f(!P) : {}^Z f(U)$.

**(CGG Proc Res)** Suppose $E \vdash_{CGG} (\nu n : Amb^Y[^Z U])P : {}^{Z'} U'$ derived from $E, n : Amb^Y[^Z U] \vdash_{CGG} P : {}^{Z'} U'$. By induction hypothesis, we have $f(E, n : Amb^Y[^Z U]) \vdash f(P) : {}^{Z'} f(U')$. This can be rewritten as $f(E), n : Amb^Y[^Z f(U), ^Z f(U)] \vdash f(P) : {}^{Z'} f(U')$. Then, by (Proc Res), we can derive $f(E) \vdash (\nu n : Amb^Y[^Z f(U), ^Z f(U)])f(P) : {}^{Z'} f(U')$. This can be rewritten as $f(E) \vdash f((\nu n : Amb^Y[^Z U])P) : {}^{Z'} f(U')$.

**(CGG Proc Amb)** Suppose $E \vdash_{CGG} M[P] : {}^{Z'} U'$ derived from $E \vdash_{CGG} M : Amb^Y[^Z U]$ and $E \vdash_{CGG} P : {}^Z U$. By induction hypothesis, we have $f(E) \vdash M : f(Amb^Y[^Z U]) = Amb^Y[^Z f(U), ^Z f(U)]$. By induction hypothesis, we also have $f(E) \vdash f(P) : {}^Z f(U)$. If $Y = \circ$, we can derive by (Proc Amb$_\circ$) that $f(E) \vdash M[f(P)] : {}^{\lrcorner}\bot \rightsquigarrow \top$. Since ${}^{Z'} f(U')$ is valid, we obtain $f(E) \vdash f(M[P]) : {}^{Z'} f(U')$ by (Proc Sub). If $Y = \bullet$, we have to replace (Proc Amb$_\circ$) by (Proc Amb$_\bullet$) and $M[f(P)]$ by $M[\![f(P)]\!]$.

**(CGG Proc Input)** Suppose $E \vdash_{CGG} (n_1 : W_1, \ldots, n_k : W_k).P : {}^Z W_1 \times \cdots \times W_k$ derived from $E, n_1 : W_1, \ldots, n_k : W_k \vdash_{CGG} P : {}^Z W_1 \times \cdots \times W_k$. By induction hypothesis, we have $f(E), n_1 : f(W_1), \ldots, n_k : f(W_k) \vdash f(P) : {}^Z f(W_1 \times \cdots \times W_k) = {}^Z f(W_1) \times \cdots \times f(W_k) \rightsquigarrow f(W_1) \times \cdots \times f(W_k)$. We can then derive by (Proc Input) that $f(E) \vdash (n_1 : f(W_1), \ldots, n_k : f(W_k)).f(P) : {}^Z f(W_1) \times \cdots \times f(W_k) \rightsquigarrow f(W_1) \times \cdots \times f(W_k)$. This can be rewritten as $f(E) \vdash f((n_1 : W_1, \ldots, n_k : W_k).P) : {}^Z f(W_1 \times \cdots \times W_k)$.

**(CGG Proc Output)** Suppose $E \vdash_{CGG} \langle M_1, \ldots, M_k \rangle : {}^Z W_1 \times \cdots \times W_k$. If $k > 0$, this was derived from $E \vdash_{CGG} M_i : W_i$ for all $1 \leq i \leq k$. By induction hypothesis, we have $f(E) \vdash M_i : f(W_i)$. We can derive by (Proc Output) that $f(E) \vdash \langle M_1, \ldots, M_k \rangle : {}^{\lrcorner} f(W_1) \times \cdots \times f(W_k) \rightsquigarrow \top$. Then, $f(E) \vdash f(\langle M_1, \ldots, M_k \rangle) : {}^Z f(W_1 \times \cdots \times W_k)$ by (Proc Sub) since ${}^Z f(W_1 \times \cdots \times W_k)$ is valid. If $k = 0$, the initial judgement was derived from $E \vdash_{CGG} \diamond$. By induction hypothesis, we have $f(E) \vdash \diamond$. By (Proc Output), we can derive $f(E) \vdash \langle\rangle : {}^{\lrcorner}\mathbf{1} \rightsquigarrow \top$. Then, $f(E) \vdash f(\langle\rangle) : {}^Z f(\mathbf{1})$ by (Proc Sub).

□

## B.5 Proof of Theorem 6

**Proof:** By induction on the rules of the algorithm.

**(Type $\varepsilon$)** If $Type(E,\varepsilon) = Cap[{}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top]$, necessarily $E \vdash \diamond$. Then, we can derive by (Exp $\varepsilon$) that $E \vdash \varepsilon : Cap[{}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top]$.

**(Type .)** We have $Type(E, M.M') = Cap[T \vee T']$ derived from $Type(E, M) = Cap[T]$ and $Type(E, M') = Cap[T']$. By induction hypothesis, we have $E \vdash M : Cap[T]$ and $E \vdash M' : Cap[T']$. By (Exp Sub), we can derive $E \vdash M : Cap[T \vee T']$ and $E \vdash M' : Cap[T \vee T']$. Finally, $E \vdash M.M' : Cap[T \vee T']$ by (Exp .).

**(Type In)** We have $Type(E, in\ M) = Cap[{}^{\curvearrowright}\bot \rightsquigarrow \top]$ derived from $Type(E, M) = Amb^Y[T, T']$. By induction hypothesis, $E \vdash M : Amb^Y[T, T']$. Then, by (Exp In), $E \vdash in\ M : Cap[{}^{\curvearrowright}\bot \rightsquigarrow \top]$.

**(Type Out)** Similar to (Type In).

**(Type Open)** We have $Type(E, open\ M) = Cap[T']$ derived from $Type(E, M) = Amb^Y[T, T']$ with $Y \leq \circ$. By induction hypothesis, $E \vdash M : Amb^Y[T, T']$. Then, by (Exp Sub), $E \vdash M : Amb^\circ[T, T']$. Finally, by (Exp Open), $E \vdash open\ M : Cap[T']$.

**(Type Imm)** If $Type(E, imm) = Cap[{}^{\underline{\vee}}\bot \rightsquigarrow \top]$, necessarily $E \vdash \diamond$. Then, we can derive by (Exp Imm) that $E \vdash imm : Cap[{}^{\underline{\vee}}\bot \rightsquigarrow \top]$.

**(Type $n$)** If $Type((E, n : W, E'), n) = W$, necessarily $E, n : W, E' \vdash \diamond$. Then, we can derive by (Exp $n$) that $E, n : W, E' \vdash n : W$.

**(Type Par)** We have $Type(E, P \mid Q) = T \vee T'$ derived from $Type(E, P) = T$ and $Type(E, Q) = T'$, with $T \vee T'$ valid. By induction hypothesis, $E \vdash P : T$ and $E \vdash Q : T'$. By (Proc Sub), we can derive $E \vdash P : T \vee T'$ and $E \vdash Q : T \vee T'$. Finally, $E \vdash P \mid Q : T \vee T'$ by (Proc Par).

**(Type Action)** We have $Type(E, M.P) = T \vee T'$ derived from $Type(E, M) = Cap[T]$ and $Type(E, P) = T'$, with $T \vee T'$ valid. By induction hypothesis, $E \vdash M : Cap[T]$ and $E \vdash P : T'$. By (Exp Sub), $E \vdash M : Cap[T \vee T']$, and by (Proc Sub), $E \vdash P : T \vee T'$. Finally, $E \vdash M.P : T \vee T'$ by (Proc Action).

**(Type Zero)** If $Type(E, \mathbf{0}) = Cap[{}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top]$, necessarily $E \vdash \diamond$. Then, we can derive by (Proc Zero) that $E \vdash \mathbf{0} : Cap[{}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top]$.

**(Type Repl)** We have $Type(E, !P) = T$ derived from $Type(E, P) = T$. By induction hypothesis, $E \vdash P : T$. Then, $E \vdash !P : T$ by (Proc Repl).

**(Type Res)** We have $Type(E, (\nu n : Amb^Y[T_n, T'_n])P) = T$ derived from $Type((E, n : Amb^Y[T_n, T'_n]), P) = T$. By induction hypothesis, $E, n : Amb^Y[T_n, T'_n] \vdash P : T$. Then, $E \vdash (\nu n : Amb^Y[T_n, T'_n])P : T$ by (Proc Res).

**(Type Amb$_\circ$)** We have $Type(E, M[P]) = {}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top$ derived from $Type(E, M) = Amb^Y[T, T']$ and from $Type(E, P) = T''$, with the conditions $T'' \leq T$ and $Y \leq \circ$. By induction hypothesis, $E \vdash M : Amb^Y[T, T']$ and $E \vdash P : T''$. By (Exp Sub), we can derive $E \vdash M : Amb^\circ[T'', T']$. Finally, $E \vdash M[P] : {}^{\llcorner\!\!\rightarrow}\bot \rightsquigarrow \top$ by (Proc Amb$_\circ$).

**(Type Amb$_\bullet$)** Similar to (Type Amb$_\circ$).

**(Type Input)** We have $Type(E, (n_1 : W_1, \ldots, n_k : W_k).P) = {}^ZO \rightsquigarrow I \wedge W_1 \times \cdots \times W_k$ derived from $Type((E, n_1 : W_1, \ldots, n_k : W_k), P) = {}^ZO \rightsquigarrow I$, with the condition $O \leq W_1 \times \cdots \times W_k$. By induction hypothesis, $E, n_1 : W_1, \ldots, n_k : W_k \vdash P : {}^ZO \rightsquigarrow I$. By Theorem 2, $Z \neq \overset{\rightarrow}{\top}$ and $O \leq I$. Then, we have $O \leq I \wedge W_1 \times \cdots \times W_k$, so that ${}^ZO \rightsquigarrow I \wedge W_1 \times \cdots \times W_k$ is valid. Since ${}^ZO \rightsquigarrow I \leq {}^ZO \rightsquigarrow I \wedge W_1 \times \cdots \times W_k$, we can derive by (Proc Sub) that $E, n_1 : W_1, \ldots, n_k : W_k \vdash P : {}^ZO \rightsquigarrow I \wedge W_1 \times \cdots \times W_k$. Checking that $I \wedge W_1 \times \cdots \times W_k \leq W_1 \times \cdots \times W_k$, we can finally derive by (Proc Input) that $E \vdash (n_1 : W_1, \ldots, n_k : W_k).P : {}^ZO \rightsquigarrow I \wedge W_1 \times \cdots \times W_k$.

**(Type Output)** We have $Type(E, \langle M_1, \ldots, M_k \rangle) = {}^{\Rsh}W_1 \times \cdots \times W_k \rightsquigarrow \top$. If $k = 0$, necessarily, $E \vdash \diamond$ and then we can derive by (Proc Output) that $E \vdash \langle\rangle : {}^{\Rsh}\mathbf{1} \rightsquigarrow \top$. Otherwise, if $k > 0$, the judgement above was derived from $Type(E, M_i) = W_i$. By induction hypothesis, $E \vdash M_i : W_i$. Then, by (Proc Output), $E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\Rsh}W_1 \times \cdots \times W_k \rightsquigarrow \top$.

$\square$

## B.6 Proof of Theorem 7

**Proof:** By induction on the derivation of $E \vdash M : W$ or $E \vdash P : T$ (by Lemma 18, $E$ is always well-formed, so the algorithm cannot fail for this reason).

**(Exp $\varepsilon$)** We have $E \vdash \varepsilon : Cap[{}^{\Rsh}\bot \rightsquigarrow \top]$. We can always derive $Type(E, \varepsilon) = Cap[{}^{\Rsh}\bot \rightsquigarrow \top]$ by (Type $\varepsilon$).

**(Exp .)** We have $E \vdash M.M' : Cap[T]$ derived from $E \vdash M : Cap[T]$ and $E \vdash M' : Cap[T]$. By induction hypothesis, we get $Type(E, M) \leq Cap[T]$ and $Type(E, M') \leq Cap[T]$. Considering the ordering of the message types, there exist $T' \leq T$ and $T'' \leq T$ such that $Type(E, M) = Cap[T']$ and $Type(E, M') = Cap[T'']$. Then, we have $Type(E, M.M') = Cap[T' \vee T'']$ by (Type .), and $Cap[T' \vee T''] \leq Cap[T]$.

**(Exp In)** We have $E \vdash in\ M : Cap[{}^{\curvearrowright}\bot \rightsquigarrow \top]$ derived from $E \vdash M : Amb^Y[T, T']$. By induction hypothesis, we have $Type(E, M) \leq Amb^Y[T, T']$. Considering the ordering of the message types, there exist $Y' \leq Y$, $T_1 \geq T$ and $T_2 \leq T'$ such that $Type(E, M) = Amb^{Y'}[T_1, T_2]$. Then, we can derive by (Type In) that $Type(E, in\ M) = Cap[{}^{\curvearrowright}\bot \rightsquigarrow \top]$.

**(Exp Out)** Similar to (Exp In).

**(Exp Open)** We have $E \vdash open\ M : Cap[T']$ derived from $E \vdash M : Amb^{\circ}[T, T']$. By induction hypothesis, we have $Type(E, M) \leq Amb^{\circ}[T, T']$. Considering the ordering of the message types, there exist $Y' \leq \circ$, $T_1 \geq T$ and $T_2 \leq T'$ such that $Type(E, M) = Amb^{Y'}[T_1, T_2]$. Then, we can derive by (Type Open) that $Type(E, open\ M) = Cap[T_2] \ \leq\ Cap[T']$.

**(Exp Imm)** We have $E \vdash imm : Cap[{}^{\veebar}\bot \rightsquigarrow \top]$. We can always derive $Type(E, imm) = Cap[{}^{\veebar}\bot \rightsquigarrow \top]$ by (Type Imm).

**(Exp $n$)** We have $E, n : W, E' \vdash n : W$. We can always derive $Type((E, n : W, E'), n) = W$ by (Type $n$).

**(Exp Sub)** We have $E \vdash M : W'$ derived from $E \vdash M : W$ and $W \leq W'$. By induction hypothesis, we have $Type(E, M) \leq W$. Then, $Type(E, M) \leq W'$.

**(Proc Par)** We have $E \vdash P \mid Q : T$ derived from $E \vdash P : T$ and $E \vdash Q : T$. By induction hypothesis, $Type(E, P) \leq T$ and $Type(E, Q) \leq T$. Let us write $Type(E, P) = T'$ and $Type(E, Q) = T''$. We have $T' \vee T'' \ \leq T$. By Theorem 2, $T$ is valid. Then, $T' \vee T''$ is also valid and we can derive $Type(E, P \mid Q) = T' \vee T'' \ \leq\ T$ by (Type Par).

**(Proc Action)** We have $E \vdash M.P : T$ derived from $E \vdash M : Cap[T]$ and $E \vdash P : T$. By induction hypothesis, $Type(E, M) \leq Cap[T]$ and $Type(E, P) \leq T$. Considering the ordering of the message types, there exists $T' \leq T$ such that $Type(E, M) = Cap[T']$. Let us write $Type(E, P) = T''$. We have $T' \vee T'' \ \leq T$. By Theorem 2, $T$ is valid. Then, $T' \vee T''$ is also valid and we can derive $Type(E, M.P) = T' \vee T'' \ \leq\ T$ by (Type Action).

**(Proc Zero)** We have $E \vdash \mathbf{0} : Cap[{}^{\Rsh}\bot \rightsquigarrow \top]$. We can always derive $Type(E, \mathbf{0}) = Cap[{}^{\Rsh}\bot \rightsquigarrow \top]$ by (Type Zero).

**(Proc Repl)** We have $E \vdash !P : T$ derived from $E \vdash P : T$. By induction hypothesis, $Type(E, P) = T' \ \leq\ T$. Then, we can derive by (Type Repl) that $Type(E,\ !P) = T' \ \leq\ T$.

**(Proc Res)** We have $E \vdash (\nu n : Amb^Y[T_n, T'_n])P : T$ derived from $E, n : Amb^Y[T_n, T'_n] \vdash P : T$. By induction hypothesis, $Type((E, n : Amb^Y[T_n, T'_n]), P) = T' \ \leq\ T$. We can then derive by (Type Res) that $Type(E, (\nu n : Amb^Y[T_n, T'_n])P) = T' \ \leq\ T$.

**(Proc Amb$_\circ$)** We have $E \vdash M[P] : {}^{\lrcorner}\bot \rightsquigarrow \top$ derived from $E \vdash M : Amb^{\circ}[T, T']$ and $E \vdash P : T$. By induction hypothesis, $Type(E, M) \leq Amb^{\circ}[T, T']$ and $Type(E, P) \leq T$. Considering the ordering of the message types, there exist $Y \leq \circ$, $T_1 \geq T$ and $T_2 \leq T'$ such that $Type(E, M) = Amb^{Y}[T_1, T_2]$. Since $Y \leq \circ$ and $Type(E, P) \leq T \leq T_1$, we can derive by (Type Amb$_\circ$) that $Type(E, M[P]) = {}^{\lrcorner}\bot \rightsquigarrow \top$.

**(Proc Amb$_\bullet$)** Similar to (Proc Amb$_\circ$).

**(Proc Input)** We have $E \vdash (n_1 : W_1, \ldots, n_k : W_k).P : {}^{Z}O \rightsquigarrow I$ derived from $E, n_1 : W_1, \ldots, n_k : W_k \vdash P : {}^{Z}O \rightsquigarrow I$ with the condition $I \leq W_1 \times \cdots \times W_k$. By induction hypothesis, we have $Type((E, n_1 : W_1, \ldots, n_k : W_k), P) \leq {}^{Z}O \rightsquigarrow I$. Let us write $Type((E, n_1 : W_1, \ldots, n_k : W_k), P) = {}^{Z'}O' \rightsquigarrow I'$. By Theorem 2, ${}^{Z}O \rightsquigarrow I$ is valid. Then, we have the inegalities: $O' \leq O \leq I \leq W_1 \times \cdots \times W_k$. We can derive by (Type Input) that $Type(E, (n_1 : W_1, \ldots, n_k : W_k).P) = {}^{Z'}O' \rightsquigarrow I' \wedge W_1 \times \cdots \times W_k \;\leq\; {}^{Z}O \rightsquigarrow I$.

**(Proc Output)** If $k > 0$, we have $E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\lrcorner}W_1 \times \cdots \times W_k \rightsquigarrow \top$ derived from $E \vdash M_i : W_i$. By induction hypothesis, we have $Type(E, M_i) \leq W_i$. Let us write $Type(E, M_i) = W_i'$. We can derive by (Type Output) that $Type(E, \langle M_1, \ldots, M_k \rangle) = {}^{\lrcorner}W_1' \times \cdots \times W_k' \rightsquigarrow \top \;\leq\; {}^{\lrcorner}W_1 \times \cdots \times W_k \rightsquigarrow \top$. If $k = 0$, we have $E \vdash \langle\rangle : {}^{\lrcorner}\mathbf{1} \rightsquigarrow \top$ and we can always derive $Type(E, \langle\rangle) = {}^{\lrcorner}\mathbf{1} \rightsquigarrow \top$.

**(Proc Sub)** We have $E \vdash P : T'$ derived from $E \vdash P : T$ with $T \leq T'$ and $T'$ valid. By induction hypothesis, $Type(E, P) \leq T$. Then, $Type(E, P) \leq T'$.

$\square$

## B.7 Proof of Theorem 9

**Proof:** Suppose $Type(E, P) = T$. By Theorem 6, we have $E \vdash P : T$. By Theorem 3, we get $E \vdash Q : T$. Then, by Theorem 7, $Type(E, Q)$ is defined and $Type(E, Q) \leq T$. $\square$

## B.8 Proof of Proposition 11

**Proof:** This proof is easy by induction on the typing derivation, so we do not detail it. $\square$

## B.9 Proof of Theorem 14

We will show the more general proposition:

- If $Infer(E, M) = (W, \sigma)$, then $\sigma(E) \vdash_{CGG} M : W$.
- If $Infer(E, P) = (T, \sigma)$, then $\sigma(E) \vdash_{CGG} \sigma(P) : T$.

**Proof:** By induction on the rules of the inference algorithm.

**(Infer $\varepsilon$)** Suppose $Infer(E, \varepsilon) = (Cap[t], \mathbf{id})$ with $t$ new. Since $E$ is well-formed, $E \vdash_{CGG} \diamond$. By (CGG Exp $\varepsilon$), we derive $\mathbf{id}(E) \vdash_{CGG} \varepsilon : Cap[t]$.

**(Infer .)** Suppose $Infer(E, M.M') = (\rho'(W'), \rho'\sigma'\rho\sigma)$, derived from the sequence $Infer(E, M) = (W, \sigma)$, $t$ new, $mgu(W, Cap[t]) = \rho$, $Infer(\rho\sigma(E), M') = (W', \sigma')$ and $mgu(W', Cap[\sigma'\rho(t)]) = \rho'$. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} M : W$ and $\sigma'\rho\sigma(E) \vdash_{CGG} M' : W'$. From the first judgement, we get $\rho'\sigma'\rho\sigma(E) \vdash_{CGG} M : \rho'\sigma'\rho(W)$. The second judgement gives $\rho'\sigma'\rho\sigma(E) \vdash_{CGG} M' : \rho'(W')$. But we have $\rho'\sigma'\rho(W) = \rho'\sigma'\rho(Cap[t]) = Cap[\rho'\sigma'\rho(t)]$ and $\rho'(W') = \rho'(Cap[\sigma'\rho(t)]) = Cap[\rho'\sigma'\rho(t)]$. Then, by (CGG Exp .), we can derive $\rho'\sigma'\rho\sigma(E) \vdash_{CGG} M.M' : Cap[\rho'\sigma'\rho(t)] = \rho'(W')$.

**(Infer In)** Suppose $Infer(E, in\ M) = (Cap[{}^{\frown}u], \rho\sigma)$, derived from the sequence $Infer(E, M) = (W, \sigma)$, $y$ and $t$ new, $mgu(W, Amb^y[t]) = \rho$ and $u$ new. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} M : W$. Then, $\rho\sigma(E) \vdash_{CGG} M : \rho(W)$. But $\rho(W) = \rho(Amb^y[t]) = Amb^{\rho(y)}[\rho(t)]$. We can then derive by (CGG Exp In) that $\rho\sigma(E) \vdash_{CGG} in\ M : Cap[{}^{\frown}u]$.

**(Infer Out)** Similar to (Infer In).

**(Infer Open)** Suppose $Infer(E, open\ M) = (Cap[\rho(t)], \rho\sigma)$, derived from the sequence $Infer(E, M) = (W, \sigma)$, $t$ new and $mgu(W, Amb^{\circ}[t]) = \rho$. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} M : W$. Then, $\rho\sigma(E) \vdash_{CGG} M : \rho(W)$. But $\rho(W) = \rho(Amb^{\circ}[t]) = Amb^{\circ}[\rho(t)]$. We can then derive by (CGG Exp Open) that $\rho\sigma(E) \vdash_{CGG} open\ M : Cap[\rho(t)]$.

**(Infer Imm)** Suppose $Infer(E, imm) = (Cap[\overset{\vee}{-}u], \mathbf{id})$ with $u$ new. Since $E$ is well-formed, $E \vdash_{CGG} \diamond$. By (CGG Exp Imm), we derive $\mathbf{id}(E) \vdash_{CGG} imm : Cap[\overset{\vee}{-}u]$.

**(Infer $n$)** Suppose $Infer((E, n : W, E'), n) = (W, \mathbf{id})$. Since $E, n : W, E'$ is well-formed, $E, n : W, E' \vdash_{CGG} \diamond$. By (CGG Exp $n$), we derive $\mathbf{id}(E, n : W, E') \vdash_{CGG} n : W$.

**(Infer Par)** Suppose $Infer(E, P \mid Q) = (\rho(T'), \rho\sigma'\sigma)$ derived from the sequence $Infer(E, P) = (T, \sigma)$, $Infer(\sigma(E), \sigma(Q)) = (T', \sigma')$ and $mgu(\sigma'(T), T') = \rho$. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} \sigma(P) : T$ and $\sigma'\sigma(E) \vdash_{CGG} \sigma'\sigma(Q) : T'$. From the first judgement, we get $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(P) : \rho\sigma'(T)$. From the second, $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(Q) : \rho(T')$. Since $\rho\sigma'(T) = \rho(T')$, we can derive by (CGG Proc Par) that $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(P \mid Q) : \rho(T')$.

**(Infer Action)** Suppose $Infer(E, M.P) = (\rho(T), \rho\sigma'\sigma)$ derived from the sequence $Infer(E, M) = (W, \sigma)$, $Infer(\sigma(E), \sigma(P)) = (T, \sigma')$ and $mgu(\sigma'(W), Cap[T]) = \rho$. By induction hypothesis, we obtain $\sigma(E) \vdash_{CGG} M : W$ and $\sigma'\sigma(E) \vdash_{CGG} \sigma'\sigma(P) : T$. From the first judgement, we get $\rho\sigma'\sigma(E) \vdash_{CGG} M : \rho\sigma'(W)$. From the second, $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(P) : \rho(T)$. Since $\rho\sigma'(W) = \rho(Cap[T]) = Cap[\rho(T)]$, we can derive by (CGG Proc Action) that $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(M.P) : \rho(T)$.

**(Infer Zero)** Suppose $Infer(E, \mathbf{0}) = (t, \mathbf{id})$ with $t$ new. Since $E$ is well-formed, $E \vdash_{CGG} \diamond$. By (CGG Proc Zero), we can derive $E \vdash_{CGG} \mathbf{0} : t$, which can be rewritten as $\mathbf{id}(E) \vdash_{CGG} \mathbf{id}(\mathbf{0}) : t$.

**(Infer Repl)** Suppose $Infer(E, !P) = (T, \sigma)$ derived from $Infer(E, P) = (T, \sigma)$. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} \sigma(P) : T$. Then, by (CGG Proc Repl), $\sigma(E) \vdash_{CGG} \sigma(!P) : T$.

**(Infer Res)** Suppose $Infer(E, (\nu n : Amb^Y[T])P) = (T', \sigma)$ derived from $Infer((E, n : Amb^Y[T]), P) = (T', \sigma)$. By induction hypothesis, we have $\sigma(E, n : Amb^Y[T]) \vdash_{CGG} \sigma(P) : T'$. This can be rewritten as $\sigma(E), n : Amb^{\sigma(Y)}[\sigma(T)] \vdash_{CGG} \sigma(P) : T'$. Then, by (CGG Proc Res), we have $\sigma(E) \vdash_{CGG} (\nu n : Amb^{\sigma(Y)}[\sigma(T)])\sigma(P) : T'$. And this can also be rewritten as $\sigma(P) \vdash_{CGG} \sigma((\nu n : Amb^Y[T])P) : T'$.

**(Infer Amb$_\circ$)** Suppose $Infer(E, M[P]) = (t, \rho\sigma'\sigma)$ derived from the sequence $Infer(E, M) = (W, \sigma)$, $Infer(\sigma(E), \sigma(P)) = (T, \sigma')$, $mgu(\sigma'(W), Amb^{\circ}[T]) = \rho$ and $t$ new. By induction hypothesis, we have $\sigma(E) \vdash_{CGG} M : W$ and $\sigma'\sigma(E) \vdash_{CGG} \sigma'\sigma(P) : T$. From the first judgement, we get $\rho\sigma'\sigma(E) \vdash_{CGG} M : \rho\sigma'(W)$. From the second judgement, we obtain $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(P) : \rho(T)$. But $\rho\sigma'(W) = \rho(Amb^{\circ}[T]) = Amb^{\circ}[\rho(T)]$. Then, by (CGG Proc Amb$_\circ$), we can derive $\rho\sigma'\sigma(E) \vdash_{CGG} \rho\sigma'\sigma(M[P]) : t$.

**(Infer Amb$_\bullet$)** Similar to (Infer Amb$_\circ$).

**(Infer Input)** Suppose $Infer(E, (n_1 : W_1, \ldots, n_k : W_k).P) = (\rho(T), \rho\sigma)$ was derived from the sequence $Infer((E, n_1 : W_1, \ldots, n_k : W_k), P) = (T, \sigma)$, $z$ new and $mgu(T, {}^z\sigma(W_1) \times \cdots \times \sigma(W_k)) = \rho$. By induction hypothesis, we have $\sigma(E, n_1 : W_1, \ldots, n_k : W_k) \vdash_{CGG} \sigma(P) : T$. Then, $\rho\sigma(E, n_1 : W_1, \ldots, n_k : W_k) \vdash_{CGG} \rho\sigma(P) : \rho(T)$, which rewrites itself as $\rho\sigma(E), n_1 : \rho\sigma(W_1), \ldots, n_k : \rho\sigma(W_k) \vdash_{CGG} \rho\sigma(P) : \rho(T)$. But $\rho(T) = \rho({}^z\sigma(W_1) \times \cdots \times \sigma(W_k)) = {}^{\rho(z)}\rho\sigma(W_1) \times \cdots \times \rho\sigma(W_k)$. Then, by (CGG Proc Input), we can derive $\rho\sigma(E) \vdash_{CGG} (n_1 : \rho\sigma(W_1), \ldots, n_k : \rho\sigma(W_k)).\rho\sigma(P) : {}^{\rho(z)}\rho\sigma(W_1) \times \cdots \times \rho\sigma(W_k)$. This can be rewritten as $\rho\sigma(E) \vdash_{CGG} \rho\sigma((n_1 : W_1, \ldots, n_k : W_k).P) : \rho(T)$.

**(Infer Output)** Suppose $Infer(E, \langle M_1, \ldots, M_k\rangle) = ({}^z\sigma_k \ldots \sigma_2(W_1) \times \cdots \times \sigma_k(W_{k-1}) \times W_k, \sigma_k \ldots \sigma_1)$ derived from the judgements $Infer(\sigma_{i-1} \ldots \sigma_1(E), M_i) = (W_i, \sigma_i)$ for $1 \leq i \leq k$, and $z$ new. By $k$ induction hypotheses, we have $\sigma_i \ldots \sigma_1(E) \vdash_{CGG} M_i : W_i$ for all $i$. Then, $\sigma_k \ldots \sigma_1(E) \vdash_{CGG} M_i :$

$\sigma_k \ldots \sigma_{i+1}(W_i)$ for all $i$. By (CGG Proc Output), we can derive $\sigma_k \ldots \sigma_1(E) \vdash_{CGG} \langle M_1, \ldots, M_k \rangle : {}^z\sigma_k \ldots \sigma_2(W_1) \times \cdots \times \sigma_k(W_{k-1}) \times W_k$.

□

## B.10 Proof of Theorem 15

In this proof, we will need a special notation for some particular substitutions: whenever we write $\theta_{new}$ (a substitution name with "new" in indice), we implicitly say that this substitution acts only on new type variables introduced by the inference algorithm and is the identity function for all other type variables.

To prove the Theorem, we need a more general lemma:

**Lemma 24**

- *If $\sigma(E) \vdash_{CGG} M : W$, then the inference algorithm succeeds: $Infer(E, M) = (W', \sigma')$, and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$ and $\theta(W') = W$.*

- *If $\sigma(E) \vdash_{CGG} \sigma(P) : T$, then the inference algorithm succeeds: $Infer(E, P) = (T', \sigma')$, and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$ and $\theta(T') = T$.*

**Proof:** By induction on the syntactic structure of $M$ or $P$ (by a reasoning similar to Lemma 18, $E$ is always well-formed, so the algorithm cannot fail for this reason).

**($\varepsilon$)** We have necessarily $\sigma(E) \vdash_{CGG} \varepsilon : Cap[T]$ derived from (CGG Exp $\varepsilon$). We can always derive $Infer(E, \varepsilon) = (Cap[t], \mathbf{id})$ where $t$ is a new type variable. Let $\theta_{new} = \{t \to T\}$ and $\theta = \theta_{new}\sigma$. Then, we can check that $\theta\ \mathbf{id} = \theta_{new}\sigma$ and $\theta(Cap[t]) = Cap[\theta_{new}\sigma(t)] = Cap[\theta_{new}(t)] = Cap[T]$ (we have $\sigma(t) = t$ because $\sigma$ is known before we choose $t$).

**($M.M'$)** We have necessarily $\sigma(E) \vdash_{CGG} M.M' : Cap[T]$ derived from $\sigma(E) \vdash_{CGG} M : Cap[T]$ and $\sigma(E) \vdash_{CGG} M' : Cap[T]$ by (CGG Exp .). By induction hypothesis on the first judgement, we have $Infer(E, M) = (W_M, \sigma_M)$ and there exist $\theta_M$ and $\theta_{M\,new}$ such that $\theta_M\sigma_M = \theta_{M\,new}\sigma$ and $\theta_M(W_M) = Cap[T]$. Let $t$ be a new type variable. We have $\{t \to T\}\ \theta_M(W_M) = \{t \to T\}\ (Cap[T]) = Cap[T]$ and $\{t \to T\}\ \theta_M(Cap[t]) = Cap[\{t \to T\}\ \theta_M(t)] = Cap[T]$ (since $t$ is a completely new type variable). Then, $mgu(W_M, Cap[t]) = \rho_M$ succeeds and there exists $\rho'_M$ such that $\rho'_M\rho_M = \{t \to T\}\ \theta_M$. Getting back to the second judgement above, we can also write $\{t \to T\}\ \theta_{M\,new}\sigma(E) \vdash_{CGG} M' : \{t \to T\}\ \theta_{M\,new}(Cap[T]) = Cap[T]$. This can be rewritten first as $\{t \to T\}\ \theta_M\sigma_M(E) \vdash_{CGG} M' : Cap[T]$, and then as $\rho'_M\rho_M\sigma_M(E) \vdash_{CGG} M' : Cap[T]$. By induction hypothesis on $\rho_M\sigma_M(E)$ and $M'$ (using $\rho'_M$ in place of $\sigma$), we have $Infer(\rho_M\sigma_M(E), M') = (W_{M'}, \sigma_{M'})$ and there exist $\theta_{M'}$ and $\theta_{M'\,new}$ such that $\theta_{M'}\sigma_{M'} = \theta_{M'\,new}\rho'_M$ and $\theta_{M'}(W_{M'}) = Cap[T]$. We have $\theta_{M'}(Cap[\sigma_{M'}\rho_M(t)]) = Cap[\theta_{M'}\sigma_{M'}\rho_M(t)] = Cap[\theta_{M'\,new}\rho'_M\rho_M(t)] = Cap[\theta_{M'\,new}\{t \to T\}\ \theta_M(t)] = Cap[\theta_{M'\,new}(T)] = Cap[T]$ and $\theta_{M'}(W_{M'}) = Cap[T]$. Then, $mgu(W_{M'}, Cap[\sigma_{M'}\rho_M(t)]) = \rho_{M'}$ succeeds and there exists $\rho'_{M'}$ such that $\rho'_{M'}\rho_{M'} = \theta_{M'}$. Then, we can derive by (Infer .) that $Infer(E, M.M') = (\rho_{M'}(W_{M'}), \rho_{M'}\sigma_{M'}\rho_M\sigma_M)$. Let $\theta_{new} = \theta_{M'\,new}\{t \to T\}\ \theta_{M\,new}$ and $\theta = \rho'_{M'}$. Then, we can check that $\theta\rho_{M'}\sigma_{M'}\rho_M\sigma_M = \rho'_{M'}\rho_{M'}\sigma_{M'}\rho_M\sigma_M = \theta_{M'}\sigma_{M'}\rho_M\sigma_M = \theta_{M'\,new}\rho'_M\rho_M\sigma_M = \theta_{M'\,new}\{t \to T\}\ \theta_M\sigma_M = \theta_{M'\,new}\{t \to T\}\ \theta_{M\,new}\sigma = \theta_{new}\sigma$, and that $\theta\rho_{M'}(W_{M'}) = \rho'_{M'}\rho_{M'}(W_{M'}) = \theta_{M'}(W_{M'}) = Cap[T]$.

**($in\ M$)** We have necessarily $\sigma(E) \vdash_{CGG} in\ M : Cap[{}^\frown U]$ derived from $\sigma(E) \vdash_{CGG} M : Amb^Y[T]$ by (CGG Exp In). By induction hypothesis, we have $Infer(E, M) = (W, \sigma')$ and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$ and $\theta(W) = Amb^Y[T]$. Let $y$ and $t$ be new type variables. We have $\{y \to Y, t \to T\}\ \theta(W) = \{y \to Y, t \to T\}(Amb^Y[T]) = Amb^Y[T]$ and $\{y \to Y, t \to T\}\ \theta(Amb^y[t]) = Amb^Y[T]$ since $y$ and $t$ are completely new. Then, $mgu(W, Amb^y[t]) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \{y \to Y, t \to T\}\ \theta$. Let $u$ be a new type variable. We can then derive by (Infer In) that $Infer(E, in\ M) = (Cap[\frown u], \rho\sigma')$. Let $\theta'_{new} = \{u \to U, y \to Y, t \to T\}\theta_{new}$ and $\theta' = \{u \to U\}\rho'$. Then, we can check that $\theta'\rho\sigma' = \{u \to U\}\rho'\rho\sigma' = \{u \to U\}\{y \to Y, t \to T\}\theta\sigma' = \{u \to U\}\{y \to Y, t \to T\}\theta_{new}\sigma = \theta'_{new}\sigma$ and $\theta'(Cap[\frown u]) = \{u \to U\}\rho'(Cap[\frown u]) = Cap[\frown U]$ (since $\rho'$ was known before $u$ was declared).

**(*out M*)** Similar to ($in\ M$).

**(*open M*)** We have necessarily $\sigma(E) \vdash_{CGG} open\ M : Cap[T]$ derived from $\sigma(E) \vdash_{CGG} M : Amb^\circ[T]$ by (CGG Exp Open). By induction hypothesis, we have $Infer(E, M) = (W, \sigma')$ and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$ and $\theta(W) = Amb^\circ[T]$. Let $t$ be a new type variable. We have $\{t \to T\}\ \theta(W) = \{t \to T\}(Amb^\circ[T]) = Amb^\circ[T]$ and $\{t \to T\}\ \theta(Amb^\circ[t]) = Amb^\circ[T]$ since $t$ is completely new. Then, $mgu(W, Amb^\circ[t]) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \{t \to T\}\ \theta$. We can then derive by (Infer Open) that $Infer(E, open\ M) = (Cap[\rho(t)], \rho\sigma')$. Let $\theta'_{new} = \{t \to T\}\theta_{new}$ and $\theta' = \rho'$. Then, we can check that $\theta'\rho\sigma' = \rho'\rho\sigma' = \{t \to T\}\theta\sigma' = \{t \to T\}\theta_{new}\sigma = \theta'_{new}\sigma$ and $\theta'(Cap[\rho(t)]) = Cap[\rho'\rho(t)] = Cap[\{t \to T\}\theta(t)] = Cap[T]$ (since $\theta$ was known before $t$ was declared).

**(*imm*)** We have necessarily $\sigma(E) \vdash_{CGG} imm : Cap[\stackrel{\vee}{-}U]$ by (CGG Exp Imm). We can always derive $Infer(E, imm) = (Cap[\stackrel{\vee}{-}u], \mathbf{id})$ where $u$ is a new type variable. Let $\theta_{new} = \{u \to U\}$ and $\theta = \theta_{new}\sigma$. Then, we can check that $\theta\ \mathbf{id} = \theta_{new}\sigma$ and $\theta(Cap[\stackrel{\vee}{-}u]) = Cap[\stackrel{\vee}{-}\theta_{new}\sigma(u)] = Cap[\stackrel{\vee}{-}\theta_{new}(u)] = Cap[\stackrel{\vee}{-}U]$ (we have $\sigma(u) = u$ because $\sigma$ is known before we choose $u$).

**(*n*)** We have necessarily $\sigma(E, n : W, E') \vdash_{CGG} n : \sigma(W)$ derived from (CGG Exp $n$). We can always derive $Infer((E, n : W, E'), n) = (W, \mathbf{id})$. Let $\theta_{new} = \mathbf{id}$ and $\theta = \sigma$. Then, we can check that $\theta\ \mathbf{id} = \sigma = \theta_{new}\sigma$ and $\theta(W) = \sigma(W)$.

**(*P* | *Q*)** We have necessarily $\sigma(E) \vdash_{CGG} \sigma(P \mid Q) : T$ derived from $\sigma(E) \vdash_{CGG} \sigma(P) : T$ and $\sigma(E) \vdash_{CGG} \sigma(Q) : T$ by (CGG Proc Par). By induction hypothesis on the first judgement, we have $Infer(E, P) = (T_P, \sigma_P)$ and there exist $\theta_P$ and $\theta_{P\,new}$ such that $\theta_P\sigma_P = \theta_{P\,new}\sigma$ and $\theta_P(T_P) = T$. Getting back to the second judgement above, we can also write $\theta_{P\,new}\sigma(E) \vdash_{CGG} \theta_{P\,new}\sigma(Q) : \theta_{P\,new}(T) = T$, that is to say $\theta_P\sigma_P(E) \vdash_{CGG} \theta_P\sigma_P(Q) : T$. By induction hypothesis on $\sigma_P(E)$ and $\sigma_P(Q)$ (with $\theta_P$ in place of $\sigma$), we have $Infer(\sigma_P(E), \sigma_P(Q)) = (T_Q, \sigma_Q)$ and there exist $\theta_Q$ and $\theta_{Q\,new}$ such that $\theta_Q\sigma_Q = \theta_{Q\,new}\theta_P$ and $\theta_Q(T_Q) = T$. Then, $\theta_Q\sigma_Q(T_P) = \theta_{Q\,new}\theta_P(T_P) = \theta_{Q\,new}(T) = T$ and $\theta_Q(T_Q) = T$. This implies that $mgu(\sigma_Q(T_P), T_Q) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \theta_Q$. Then, we can derive by (Infer Par) that $Infer(E, P \mid Q) = (\rho(T_Q), \rho\sigma_Q\sigma_P)$. Let $\theta_{new} = \theta_{Q\,new}\theta_{P\,new}$ and $\theta = \rho'$. We can check that $\theta\rho\sigma_Q\sigma_P = \rho'\rho\sigma_Q\sigma_P = \theta_Q\sigma_Q\sigma_P = \theta_{Q\,new}\theta_P\sigma_P = \theta_{Q\,new}\theta_{P\,new}\sigma = \theta_{new}\sigma$, and that $\theta\rho(T_Q) = \rho'\rho(T_Q) = \theta_Q(T_Q) = T$.

**(*M.P*)** We have necessarily $\sigma(E) \vdash_{CGG} \sigma(M.P) : T$ derived from $\sigma(E) \vdash_{CGG} M : Cap[T]$ and $\sigma(E) \vdash_{CGG} \sigma(P) : T$ by (CGG Proc Action). By induction hypothesis on the typing judgement for $M$, we have $Infer(E, M) = (W, \sigma_M)$ and there exist $\theta_M$ and $\theta_{M\,new}$ such that $\theta_M\sigma_M = \theta_{M\,new}\sigma$ and $\theta_M(W) = Cap[T]$. Getting back to the second judgement above, we can also write $\theta_{M\,new}\sigma(E) \vdash_{CGG} \theta_{M\,new}\sigma(P) : \theta_{M\,new}(T) = T$, that is to say $\theta_M\sigma_M(E) \vdash_{CGG} \theta_M\sigma_M(P) : T$. By induction hypothesis on $\sigma_M(E)$ and $\sigma_M(P)$ (with $\theta_M$ in place of $\sigma$), we have $Infer(\sigma_M(E), \sigma_M(P)) = (T_P, \sigma_P)$ and there exist $\theta_P$ and $\theta_{P\,new}$ such that $\theta_P\sigma_P = \theta_{P\,new}\theta_M$ and $\theta_P(T_P) = T$. Then, $\theta_P\sigma_P(W) = \theta_{P\,new}\theta_M(W) = \theta_{P\,new}(Cap[T]) = Cap[T]$ and $\theta_P(Cap[T_P]) = Cap[\theta_P(T_P)] = Cap[T]$. This implies that $mgu(\sigma_P(W), Cap[T_P]) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \theta_P$. Then, we can derive by (Infer Action) that $Infer(E, M.P) = (\rho(T_P), \rho\sigma_P\sigma_M)$. Let $\theta_{new} = \theta_{P\,new}\theta_{M\,new}$ and $\theta = \rho'$. We can check that $\theta\rho\sigma_P\sigma_M = \rho'\rho\sigma_P\sigma_M = \theta_P\sigma_P\sigma_M = \theta_{P\,new}\theta_M\sigma_M = \theta_{P\,new}\theta_{M\,new}\sigma = \theta_{new}\sigma$, and that $\theta\rho(T_P) = \rho'\rho(T_P) = \theta_P(T_P) = T$.

**(0)** We have necessarily $\sigma(E) \vdash_{CGG} \sigma(\mathbf{0}) : T$ derived from (CGG Proc Zero). We can always derive $Infer(E, \mathbf{0}) = (t, \mathbf{id})$ where $t$ is a new type variable. Let $\theta_{new} = \{t \to T\}$ and $\theta = \theta_{new}\sigma$. Then, we can check that $\theta\ \mathbf{id} = \theta_{new}\sigma$ and $\theta(t) = \theta_{new}\sigma(t) = \theta_{new}(t) = T$ (we have $\sigma(t) = t$ because $\sigma$ is known before we choose $t$).

**(!*P*)** We have necessarily $\sigma(E) \vdash_{CGG} \sigma(!P) : T$ derived from $\sigma(E) \vdash_{CGG} \sigma(P) : T$ by (CGG Proc Repl). By induction hypothesis, we have $Infer(E, P) = (T', \sigma')$ and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$

and $\theta(T') = T$. By (Infer Repl), we can then derive $Infer(E, !P) = (T', \sigma')$. We keep the same $\theta$ and $\theta_{new}$ since they have to satisfy exactly the same conditions.

**$((\nu n : Amb^Y[T])P)$** We have necessarily $\sigma(E) \vdash_{CGG} \sigma((\nu n : Amb^Y[T])P) : T'$ derived from $\sigma(E, n : Amb^Y[T]) \vdash_{CGG} \sigma(P) : T'$ by (CGG Proc Res). By induction hypothesis, we have $Infer((E, n : Amb^Y[T]), P) = (T_P, \sigma_P)$ and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma_P = \theta_{new}\sigma$ and $\theta(T_P) = T'$. By (Infer Res), we can then derive $Infer(E, (\nu n : Amb^Y[T])P)) = (T_P, \sigma_P)$. We keep the same $\theta$ and $\theta_{new}$ since they have to satisfy exactly the same conditions.

**$(M[P])$** We have necessarily $\sigma(E) \vdash_{CGG} \sigma(M[P]) : T$ derived from $\sigma(E) \vdash_{CGG} M : Amb^\circ[T']$ and $\sigma(E) \vdash_{CGG} \sigma(P) : T'$ by (CGG Proc Amb$_\circ$). By induction hypothesis on the first judgement, we have $Infer(E, M) = (W, \sigma_M)$ and there exist $\theta_M$ and $\theta_{M\,new}$ such that $\theta_M\sigma_M = \theta_{M\,new}\sigma$ and $\theta_M(W) = Amb^\circ[T']$. Getting back to the second judgement above, we can also write $\theta_{M\,new}\sigma(E) \vdash_{CGG} \theta_{M\,new}\sigma(P) : \theta_{M\,new}(T') = T'$, that is to say $\theta_M\sigma_M(E) \vdash_{CGG} \theta_M\sigma_M(P) : T'$. By induction hypothesis on $\sigma_M(E)$ and $\sigma_M(P)$ (with $\theta_M$ in place of $\sigma$), we have $Infer(\sigma_M(E), \sigma_M(P)) = (T_P, \sigma_P)$ and there exist $\theta_P$ and $\theta_{P\,new}$ such that $\theta_P\sigma_P = \theta_{P\,new}\theta_M$ and $\theta_P(T_P) = T'$. Then, $\theta_P\sigma_P(W) = \theta_{P\,new}\theta_M(W) = \theta_{P\,new}(Amb^\circ[T']) = Amb^\circ[T']$ and $\theta_P(Amb^\circ[T_P]) = Amb^\circ[\theta_P(T_P)] = Amb^\circ[T']$. This implies that $mgu(\sigma_P(W), Amb^\circ[T_P]) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \theta_P$. Let $t$ be a new type variable. Then, we can derive by (Infer Amb$_\circ$) that $Infer(E, M[P]) = (t, \rho\sigma_P\sigma_M)$. Let $\theta_{new} = \{t \to T\}\ \theta_{P\,new}\theta_{M\,new}$ and $\theta = \{t \to T\}\ \rho'$. We can check that $\theta\rho\sigma_P\sigma_M = \{t \to T\}\ \rho'\rho\sigma_P\sigma_M = \{t \to T\}\ \theta_P\sigma_P\sigma_M = \{t \to T\}\ \theta_{P\,new}\theta_M\sigma_M = \{t \to T\}\ \theta_{P\,new}\theta_{M\,new}\sigma = \theta_{new}\sigma$, and that $\theta(t) = \{t \to T\}\ \rho'(t) = T$ (since $\rho'$ was known before $t$ was declared).

**$(M[\![P]\!])$** Similar to $(M[P])$.

**$((n_1 : W_1, \ldots, n_k : W_k).P)$** We have necessarily $\sigma(E) \vdash_{CGG} \sigma((n_1 : W_1, \ldots, n_k : W_k).P) : {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$ derived from $\sigma(E, n_1 : W_1, \ldots, n_k : W_k) \vdash_{CGG} \sigma(P) : {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$ by (CGG Proc Input). By induction hypothesis, we have $Infer((E, n_1 : W_1, \ldots, n_k : W_k), P) = (T_P, \sigma_P)$ and there exist $\theta_P$ and $\theta_{P\,new}$ such that $\theta_P\sigma_P = \theta_{P\,new}\sigma$ and $\theta_P(T_P) = {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$. Let $z$ be a new type variable. We have $\{z \to Z\}\ \theta_P(T_P) = \{z \to Z\}\ ({}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)) = {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$ and $\{z \to Z\}\ \theta_P({}^z\sigma_P(W_1) \times \cdots \times \sigma_P(W_k)) = \{z \to Z\}\ ({}^z\theta_P\sigma_P(W_1) \times \cdots \times \theta_P\sigma_P(W_k)) = \{z \to Z\}\ ({}^z\theta_{P\,new}\sigma(W_1) \times \cdots \times \theta_{P\,new}\sigma(W_k)) = \{z \to Z\}\ ({}^z\sigma(W_1) \times \cdots \times \sigma(W_k)) = {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$ (since $z$ is a completely new type variable and all $\sigma(W_i)$ were known before $\theta_{P\,new}$ and $z$). This implies that $mgu(T_P, {}^z\sigma_P(W_1) \times \cdots \times \sigma_P(W_k)) = \rho$ succeeds and there exists $\rho'$ such that $\rho'\rho = \{z \to Z\}\ \theta_P$. Then, we can derive by (Infer Input) that $Infer(E, (n_1 : W_1, \ldots, n_k : W_k).P) = (\rho(T_P), \rho\sigma_P)$. Let $\theta_{new} = \{z \to Z\}\ \theta_{P\,new}$ and $\theta = \rho'$. Then, we can check that $\theta\rho\sigma_P = \rho'\rho\sigma_P = \{z \to Z\}\ \theta_P\sigma_P = \{z \to Z\}\ \theta_{P\,new}\sigma = \theta_{new}\sigma$, and that $\theta\rho(T_P) = \rho'\rho(T_P) = \{z \to Z\}\ \theta_P(T_P) = {}^Z\sigma(W_1) \times \cdots \times \sigma(W_k)$ (since this type was known before $z$ was declared).

**$(\langle M_1, \ldots, M_k \rangle)$** We have necessarily $\sigma(E) \vdash_{CGG} \langle M_1, \ldots, M_k \rangle : {}^Z W_1 \times \cdots \times W_k$ derived from $\sigma(E) \vdash_{CGG} M_i : W_i$ for $1 \leq i \leq k$ (by CGG Proc Output). By induction hypothesis on the judgement for $i = 1$, we have $Infer(E, M_1) = (W'_1, \sigma_1)$ and there exist $\theta_1$ and $\theta_{1\,new}$ such that $\theta_1\sigma_1 = \theta_{1\,new}\sigma$ and $\theta_1(W'_1) = W_1$. From the judgement for $i = 2$ above, we obtain $\theta_{1\,new}\sigma(E) \vdash_{CGG} M_2 : \theta_{1\,new}(W_2) = W_2$, that is to say $\theta_1\sigma_1(E) \vdash_{CGG} M_2 : W_2$. By induction hypothesis on $\sigma_1(E)$ and $M_2$ (with $\theta_1$ in place of $\sigma$), we have $Infer(\sigma_1(E), M_2) = (W'_2, \sigma_2)$ and there exist $\theta_2$ and $\theta_{2\,new}$ such that $\theta_2\sigma_2 = \theta_{2\,new}\theta_1$ and $\theta_2(W'_2) = W_2$. From the judgement for $i = 3$ above, we obtain $\theta_{2\,new}\theta_{1\,new}\sigma(E) \vdash_{CGG} M_3 : \theta_{2\,new}\theta_{1\,new}(W_3) = W_3$. This can be rewritten first as $\theta_{2\,new}\theta_1\sigma_1(E) \vdash_{CGG} M_3 : W_3$, then as $\theta_2\sigma_2\sigma_1(E) \vdash_{CGG} M_3 : W_3$. We just apply the same reasoning to $\sigma_2\sigma_1(E)$ and $M_3$, and continue like this until we obtain $\theta_k$ and $\theta_{k\,new}$. We have the relations $\theta_i\sigma_i = \theta_{i\,new}\theta_{i-1}$ (writing $\theta_0 = \sigma$) and $\theta_i(W'_i) = W_i$ for $1 \leq i \leq k$. Let $z$ be a new type variable. Then, we can derive $Infer(E, \langle M_1, \ldots, M_k \rangle) = ({}^z\sigma_k \ldots \sigma_2(W'_1) \times \cdots \times \sigma_k(W'_{k-1}) \times W'_k, \sigma_k \ldots \sigma_1)$ by (Infer Output). Let $\theta_{new} = \{z \to Z\}\ \theta_{k\,new} \ldots \theta_{1\,new}$ and $\theta = \{z \to Z\}\ \theta_k$. Then, we can check that $\theta\sigma_k \ldots \sigma_1 = \{z \to Z\}\ \theta_k\sigma_k \ldots \sigma_1 = \{z \to Z\}\ \theta_{k\,new}\theta_{k-1}\sigma_{k-1} \ldots \sigma_1 = \ldots = \{z \to Z\}\ \theta_{k\,new} \ldots \theta_{2\,new}\theta_1\sigma_1 = \{z \to Z\}\ \theta_{k\,new} \ldots \theta_{1\,new}\sigma = \theta_{new}\sigma$, and that $\theta({}^z\sigma_k \ldots \sigma_2(W'_1) \times \cdots \times \sigma_k(W'_{k-1}) \times W'_k) = \{z \to Z\}\ ({}^z\theta_k\sigma_k \ldots \sigma_2(W'_1) \times \cdots \times \theta_k\sigma_k(W'_{k-1}) \times \theta_k(W'_k)) = \{z \to Z\}\ ({}^z\theta_{k\,new} \ldots \theta_{2\,new}\theta_1(W'_1) \times \cdots \times$

$\theta_{k\,new}\theta_{k-1}(W'_{k-1}) \times \theta_k(W'_k)) = \{z \to Z\}\ ({}^{z}\theta_{k\,new} \ldots \theta_{2\,new}(W_1) \times \cdots \times \theta_{k\,new}(W_{k-1}) \times W_k) = \{z \to Z\}\ ({}^{z}W_1 \times \cdots \times W_k) = {}^{Z}W_1 \times \cdots \times W_k$.

□

Now, we can prove the Theorem 15:

**Proof:** Suppose we have $\sigma(E) \vdash_{CGG} \sigma(P) : T$ for some $\sigma$ and some type $T$. By Lemma 24, the inference algorithm succeeds, we have $Infer(E, P) = (T', \sigma')$ and there exist $\theta$ and $\theta_{new}$ such that $\theta\sigma' = \theta_{new}\sigma$ and $\theta(T') = T$. By Theorem 14, we get the solution $\sigma'(E) \vdash_{CGG} \sigma'(P) : T'$. Then, we can also derive another solution of the inference algorithm: $\theta\sigma'(E) \vdash_{CGG} \theta\sigma'(P) : \theta(T')$. This can be rewritten as $\theta_{new}\sigma(E) \vdash_{CGG} \theta_{new}\sigma(P) : T$. Since $\theta_{new}$ is a substitution on new variables only and since we knew $\sigma(E)$ and $\sigma(P)$ before applying the algorithm, we have $\theta_{new}\sigma(E) = \sigma(E)$ and $\theta_{new}\sigma(P) = \sigma(P)$, so that the above solution rewrites itself as $\sigma(E) \vdash_{CGG} \sigma(P) : T$. This shows that our starting typing judgement is always a solution returned by the inference algorithm. Consequently, the algorithm is complete. □