Towards an algebraic theory of typed mobile processes

Yuxin Deng[1] and Davide Sangiorgi[2]*

[1] INRIA and Université Paris 7, France
[2] Università di Bologna, Italy

Abstract. The impact of types on the algebraic theory of the π-calculus is studied. The type system has capability types. They allow one to distinguish between the ability to read from a channel, to write to a channel, and both to read and to write. They also give rise to a natural and powerful subtyping relation.
Two variants of typed bisimilarity are considered, both in their late and in their early version. For both of them, proof systems that are sound and complete on the closed finite terms are given. For one of the two variants, a complete axiomatisation for the open finite terms is also presented.

1 Introduction

The -calculus is the paradigmatic calculus for mobile processes. Its theory has been studied in depth [8, 12]. Relevant parts of it are the algebraic theory and the type systems. Most of the algebraic theory has been developed on the untyped calculus; the results include proof systems or axiomatisations that are sound and complete on nite processes for the main behavioral equivalences: late and early bisimilarity, late and early congruence [9, 5, 6], open bisimilarity [11], testing equivalence [1]. Much of the research on types has focused on their behavioral e ects. For instance, modi cations of the standard behavioral equivalences have been proposed so as to take types into account [10, 12].

In this paper, we study the impact of types on the algebraic theory of the -calculus. Precisely, we study axiomatisations of the typed -calculus. Although algebraic laws for typed calculi for mobility have been considered in the literature [12], we are not aware of any axiomatisation or proof system.

The type system that we consider has capability types (sometimes called I/O types) [10, 4]. These types allow us to distinguish, for instance, the capability of using a channel in input from that of using the channel in output. A capability type shows the capability of a channel and, recursively, of the channels carried by that channel. For instance, a type $a : \mathrm{iob}\, T$ (for an appropriate type expression T) says that channel a can be used only in input; moreover, any channel received at a may only be used in output | to send channels which can be used both in input and in output. Thus, process $a(x).x\langle b\rangle.b(y).b\langle y\rangle.0$ (sometimes the trailing 0 is omitted) is well-typed under the type assignment $a : \mathrm{iob}\, T; b : \mathrm{b}T$. We recall

* Work supported by EU project PROFUNDIS

that $ab:P$ is the output at a of channel b with continuation P, and that $a(x):P$ is an input at a with x a placeholder for channels received in the input whose continuation is P.

On calculi for mobility, capability types have emerged as one of the most useful forms of types, and one whose behavioral effects are most prominent. Capabilities are useful for protecting resources; for instance, in a client-server model, they can be used for preventing clients from using the access channel to the server in input and stealing messages to the server; similarly they can be used in distributed programming for expressing security constraints [4]. Capabilities give rise to subtyping: the output capability is contravariant, whereas the input capability is covariant. As an example, we show a subtyping relation in Fig. 1, where an arrow indicates the subtyping relation between two related types. The depth of nesting of capabilities is 1 for all types in diagram (a), and 2 for all types in diagram (b). (The formal definitions of types and subtyping relation will be given in Section 2.) Subtyping is useful when the π-calculus is used for object-

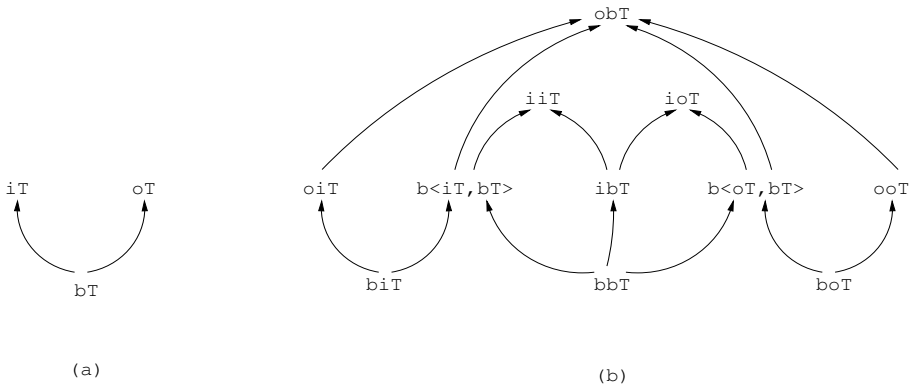

Fig. 1. Subtyping relation, with T = unit

oriented programming, or for giving semantics to object-oriented languages.

To see why the addition of capability types has semantic consequences, consider

$$P \stackrel{\text{def}}{=} c(\bar{b}c:a(y):(\bar{y} \mid c)) \qquad Q \stackrel{\text{def}}{=} c(\bar{b}c:a(y):(\bar{y}:c + c:\bar{y}))$$

These processes are not behaviorally equivalent in the untyped π-calculus. For instance, if the channel received at a is c, then P can terminate after 2 interactions with the external observer. By contrast, Q always terminates after 4 interactions with the observer. However, if we require that only the input capability of channels may be communicated at b, then P and Q are indistinguishable in any (well-typed) context. For instance, since the observer only receives the input capability on c, it cannot resend c along a: channels sent at a require at least the output capability (cf: the occurrence of $\bar{y}$). Therefore, in the typed

channel and variable parts of , respectively. A type environment is unde ned on in nitely many channels and variables (to make sure it can always be extended). We often view, and talk about, $_c$ as a set of assignments of the form $a : T$. Similarly for $_v$. We write dom( ) for the channels and variables on which is de ned (dom( ) can be in nite). Using the partial meet operation, we can extend a type environment to $u\ u : T$, which is just $; u : T$ if $u \notin \mathrm{dom}( )$, otherwise it di ers from at name u because the capability of this name is extended to be (u) u T (if (u) u T is unde ned, then so is $u\ u : T$). When $\mathrm{dom}( ) \setminus \mathrm{dom}(^0) = ;$, we use $; ^0$ to represent the union of and ^{0}. Subtyping is extended to type environments, but only considering the types of channels. So $< :$ means that $_v = _v$, $\mathrm{dom}(_c) \quad \mathrm{dom}(_c)$ and $_c(a) <: _c(a)$ for all $a \in \mathrm{dom}(_c)$. The intuition is that channels are capabilities while variables are obligations of the environment. The environment is obligated to ll in the variables at the speci ed types. Therefore we cannot allow weakening on variables as this would weaken the obligations. On the other hand we cannot use contravariant weakening because then the process would not type. That's why variables are invariant. If (u) is de ned and takes the form $i\ T$ or $bhT; Si$, then the predicate $(u)\#_i$ holds and we write $(u)_i$ for T, otherwise we write $(u) 6\#_i$, indicating that has no input capability on u. Similarly for $(u)_o$ and $(u)\#_o$ (output capability). The typing rules follow [10, 4, 3].

De nition 1. A con guration is a pair $]P$ which respects some type environment , i.e., $< :$ and $` P$.

The transition system for con gurations is in Table 2. Bound names, names and the subject of a pre x , written bn( ); n( ) and subj( ) respectively, are de ned in the usual way. We identify terms up to alpha conversion and assume $\mathrm{bn}(P) \setminus \mathrm{dom}( ) = ;$ for any con guration $]P$. In the premise of rule Red, $P \to P^0$ stands for the normal reduction relation of the untyped -calculus. In rule Out, the process sends channel b to the environment, so the latter should be dynamically extended with the capability on b thus received. For this, we use the meet operator, and exploit the following property on types:

$$R <: T \text{ and } R <: S \text{ imply } T \sqcap S \text{ de ned and } R <: T \sqcap S$$

for any type T; S and R. (This property does not hold for the capability types as in [10].)

P Q reads \P and Q are bisimilar under type environment ". The type environment is used as follows: $_c$ shows the channels that are known to the external observer testing the processes in the bisimulation game, and the types with which the observer is allowed to use such channels. By contrast, $_v$ shows the set of variables that may appear free in the processes and the types for these variables show how the observer can instantiate such variables (in closing substitutions). Therefore: the channels of $_c$ are to be used by the observer, with the types indicated in $_c$; the variables in $_v$ are to be used by the processes, but the observer can instantiate them following the types indicated in $_v$.

$$\text{Red}\ \frac{P \xrightarrow{\tau} P'}{\Delta \vdash P \xrightarrow{\tau} \Delta \vdash P'} \qquad \text{Out}\ \frac{\Delta(a)\#_{\mathrm{i}}}{\Delta \vdash \overline{a}b.P \xrightarrow{\overline{a}b} \Delta \uplus b : \Delta(a)_{\mathrm{i}} \vdash P}$$

$$\text{In}\ \frac{\Delta(a)\#_{\mathrm{o}}}{\Delta \vdash a(x:T).P \xrightarrow{a(x:T)} \Delta, x:T \vdash P} \qquad \text{Open}\ \frac{\Delta \vdash P \xrightarrow{\overline{a}b} \Delta' \vdash P' \quad a \neq b}{\Delta \vdash (\nu b:T)P \xrightarrow{\overline{a}(b:T)} \Delta' \vdash P'}$$

$$\text{Res}\ \frac{\Delta \vdash P \xrightarrow{\alpha} \Delta' \vdash P' \quad a \notin n(\alpha)}{\Delta \vdash (\nu a:T)P \xrightarrow{\alpha} \Delta' \vdash (\nu a:T)P'} \qquad \text{Sum}\ \frac{\Delta \vdash P \xrightarrow{\tau} \Delta' \vdash P'}{\Delta \vdash P + Q \xrightarrow{\tau} \Delta' \vdash P'}$$

$$\text{True}\ \frac{[\![\phi]\!] = \text{True} \quad \Delta \vdash P \xrightarrow{\tau} \Delta' \vdash P'}{\Delta \vdash \phi P Q \xrightarrow{\tau} \Delta' \vdash P'} \qquad \text{False}\ \frac{[\![\phi]\!] = \text{False} \quad \Delta \vdash Q \xrightarrow{\tau} \Delta' \vdash Q'}{\Delta \vdash \phi P Q \xrightarrow{\tau} \Delta' \vdash Q'}$$

$$\text{Par}\ \frac{\Delta \vdash P \xrightarrow{\alpha} \Delta' \vdash P' \quad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{\Delta \vdash P \mid Q \xrightarrow{\alpha} \Delta' \vdash P' \mid Q}$$

Table 2. Typed transition system

A process is *closed* if it does not have free variables; similarly a type environment is closed if it is only defined on channels. Otherwise, processes and type environments are *open*. We first define $\approx$ on the closed terms, then on the open terms. We write $|\alpha|$ for the action α where its type annotations have been stripped off. Bisimilarity is given in the "late" style [12]; we consider the "early" style in Section 5.2.

Definition 2. A family of symmetric binary relations over closed terms, indexed by type environments, and written $\{\mathcal{R}_\Delta\}_\Delta$, is a typed bisimulation whenever $P\ \mathcal{R}_\Delta\ Q$ implies that, for two configurations $\Delta \vdash P$ and $\Delta \vdash Q$,

1. if $\Delta \vdash P \xrightarrow{\alpha} \Delta' \vdash P'$ and α is not an input action, then for some Q', $\Delta \vdash Q \xrightarrow{\alpha} \Delta' \vdash Q'$, $|\alpha| = |\alpha|$ and $P' \mathcal{R}_{\Delta'} Q'$.
2. if $\Delta \vdash P \xrightarrow{a(x:T)} \Delta' \vdash P'$, then for some Q', $\Delta \vdash Q \xrightarrow{a(x:S)} \Delta' \vdash Q'$ and for all b with $c \vdash b : (a)_{\mathrm{o}}$ it holds that $P'\{b/x\}\ \mathcal{R}_{\Delta'}\ Q'\{b/x\}$.

Two processes P and Q are *typed* Δ*-bisimilar*, written $P \approx_\Delta Q$, if there exists a typed bisimulation $\{\mathcal{R}_\Delta\}_\Delta$ such that $P\ \mathcal{R}_\Delta\ Q$.

The difference w.r.t. typed bisimilarity as in [2, 3] is that, in the input clause, the type environment Δ is not extended. In other words, the knowledge of the external observer does not change through interactions with the process in which the value transmitted is supplied by the observer itself (by contrast, the knowledge does change when the value is supplied by the process; cf. rule Out in Table 2). We discuss the alternative equivalence (where the environment can be extended) in Section 5.1.

Definition 3. Two processes P and Q are bisimilar under the environment Γ, written $P \approx Q$, if $\mathrm{fv}(P, Q) \subseteq \mathrm{dom}(\Gamma_{\mathrm{v}})$ and, for all θ with $c \vdash \theta : \Gamma$, it holds that $P\{\theta/x\} \approx_{\Gamma\theta} Q\{\theta/x\}$.

Since all processes are nite, and we do not use recursive types, if P Q, the environment can always be taken to be nite (i.e., de ned only on a nite number of channels and variables): it is su cient that has enough names fresh w.r.t. P and Q, for all relevant types. This can be proved with a construction similar to that in Lemma 2. In the remainder of the paper all type environments are assumed to be nite.

3 Axioms for typed bisimilarity

The axiom system A for typed bisimilarity is in Table 3. Whenever we write P = Q it is intended that both]P and]Q are con gurations. The rules are divided into seven groups, namely those for: substitutivity, sums, looking up the type environment, conditions, restrictions, the expansion law and alpha-conversion. The rules that are new or di erent w.r.t. those of the untyped -calculus are marked with an asterisk. As in the untyped case [9], other rules involving conditions, such as

$$C8 \quad [' \_]P = 'P + P$$

are derivable. Tin* shows that an input pre x is not observable if the observer has no output capability on the subject of the input. Tout* is the symmetric rule, for output. Twea* gives us weakening for type environments. Tvar* shows that a variable can only be instantiated with channels that in the type environment have types compatible with that of the variable. Tpre* is used to replace names underneath a match. (In the untyped setting, the rule has no side condition. Here we need one to ensure well-typedness of the process resulting from the substitution, since the names in the match can have arbitrary | and possibly unrelated | types. Similarly for the conditions on types in the expansion law E.) In Ires* , di erent types $T_1; T_2$ are used for the processes in the conclusion. We cannot replace Ires* with two simpler rules such as

$$\begin{array}{l} \text{If } P = Q \text{ then } (a : T)P = (a : T)Q \\ (a : T_1)P = (a : T_2)P, \end{array}$$

for equalities like

$$(b : bi\ T)ab:b(x : i\ T):0 = _{a:iob\ T}\ (b : boT)ab:b(x : oT):0$$

could not be derived (due to the constraints given by the well-typedness of processes). Similarly for rule Iin* .

To illustrate rule Ipar* we need to introduce some notations. We de ne the depth, d(T), of a type T, to be the maximum number of nesting of capabilities in it. Let ` P. Each name in P has a type, either recorded in the syntax of P or in . If $T_1; : : : ; T_n$ are all such types, d( ; P ) is max$f$ d($T_i$) j 1 i n$g$. Now, if ]$P_i$ is a con guration, for $i = 1; 2$, then there are type environments $_i$ such that $_i <:$ and $_i$ ` P_i. In this case, we set d($P_1; P_2; _1; _2$) as max$f$ d($_1; P_1$); d($_2; P_2$)g. There are only nitely many di erent types with

depth less than or equal to $d(P_1, P_2, \Gamma_1, \Gamma_2)$, say $S_1, \ldots, S_m$, and Γ_v is defined on finitely many variables, say $x_1, \ldots, x_k$. We can pick up n fresh (hitherto unused) channels $a_{i1}, \ldots, a_{in}$ for each S_i, with n the larger value between k and the size of $P_1 + P_2$, and construct a type environment

$$Env(\Gamma; P_1, P_2, \Gamma_1, \Gamma_2) = \{a_{ij} : S_{ij} \mid 0 < i \leq m, 0 < j \leq n\}.$$

Rule Ipar* says that if Γ cannot distinguish P from Q, then it cannot distinguish $P \mid R$ from $Q \mid R$ either, provided that: (i) Γ contains enough fresh channels; (ii) R requires no capabilities beyond the knowledge of Γ. Note that we cannot do without the first condition, i.e., the rule cannot be simplified as:

$$\text{For any } \Gamma, \text{ if } P = Q \text{ and } \Gamma \vdash R \text{ then } P \mid R = Q \mid R$$

which is unsound for Γ. The point is that when comparing $P \mid R$ and $Q \mid R$, the observer may first increase his knowledge by interacting with R, then distinguish P from Q by the new knowledge. For example, let $\Gamma \stackrel{\text{def}}{=} a : bT, e : bT, b : T$ and

$$P \stackrel{\text{def}}{=} a(x : T).[x \neq b] \qquad Q \stackrel{\text{def}}{=} a(x : T).0 \qquad R \stackrel{\text{def}}{=} (c : T)\overline{e}c.$$

It is easy to see that $P = Q$ and $\Gamma \vdash R$ but $P \mid R \neq Q \mid R$. After the interaction with R, the environment evolves into $\Gamma, c : T$. Later the new channel c may be used to instantiate x, thus validating the condition $x \neq b$ and liberating the prefix.

Iin* and Iout* are the rules for substitutivity for input and output prefixes. In Iin* , well-definedness of the configurations $]a(x : T_1).P$ and $]a(x : T_2).Q$ implies the condition: $(a)_o <: T_i$ for $i = 1, 2$. (Similarly for rules Iv1 , Iin and Iin^0 below.) In Iout* , the observer knowledge of the type of b may increase when the processes emit b themselves (for the type under which b is emitted is composed with the possible type of b in Γ). Both in Iin* and in Iout* , the free names of the input and output prefixes are channels rather than variables. Below we discuss:

1. the unsoundness of the rules in which (some or all) the channels are replaced by variables;
2. other rules, that are valid for variables;
3. why these other rules are not needed in the axiom system.

Intuitively the reason for (1) is the different usages of channels and variables that appear in a type environment: the information on channels tells us how these channels are to be used by the external environment, while the information on variables tells us how these variables are to be instantiated inside the tested processes.

To see that Iin* is unsound when the subject of the prefix is a variable, take $\Gamma_c \stackrel{\text{def}}{=} a : b\circ T, b : \circ T$ and $\Gamma \stackrel{\text{def}}{=} \Gamma_c, x : b\circ T, bT$. Then we have

$$[y = b] \quad \Gamma; y : (x)_\circ \vdash 0$$

Iin*	If $P = {}_{;x\ :\ (a)_o}\ Q$ then $a(x : T_1).P = a(x : T_2).Q$
Iout*	If $P = {}_{u\,b:\ (a)_i}\ Q$ then $\bar{a}b.P = \bar{a}b.Q$
Itau	If $P = Q$ then $\tau.P = \tau.Q$
Isum	If $P = Q$ then $P + R = Q + R$
Ires*	If $P = Q$ then $(\nu a : T_1)P = (\nu a : T_2)Q$ a 62dom(Δ)
Icon	If $P = Q$ then $\phi P = \phi Q$
Ipar*	Assume Δ_0 P respects Γ_1, Δ_0 Q respects Γ_2, and $\Delta = \Delta_0, Env(\Delta_0, P, Q, \Gamma_1, \Gamma_2)$. If $P = Q$ and Δ ` R then P j $R = Q$ j R

S1	$P + 0 = P$
S2	$P + P = P$
S3	$P + Q = Q + P$
S4	$P + (Q + R) = (P + Q) + R$

Tin*	If $\Delta(a)$6# then $a(x : T).P = 0$
Tout*	If $\Delta(a)$6# then $\bar{a}u.P = 0$
Twea*	If $P = Q$ and $\Delta <: \Delta^0$ then $P = {}_0\ Q$
Tvar*	$[x$ 6= $a_1]$ $[x$ 6= $a_m]P = 0$ if f b 2 dom(Δ_c) j $\Delta(b) <: \Delta(x)$g f a_1, , a_m g
Tpre*	$[x = a]\alpha.P = [x = a](\alpha$f a/xg$).P$ if $\Delta(a) <: \Delta(x)$

C1	$\phi\ P = \psi\ P$ if ϕ () ψ
C2	$[a = b]P = [a = b]Q$ if a 6= b
C3	$\phi\ P\ P = P$
C4	$\phi\ P\ Q = : \phi\ Q\ P$
C5	$\phi(\psi P) = [\phi \wedge \psi]P$
C6	$\phi\ (P_1 + P_2)\ (Q_1 + Q_2) = \phi\ P_1\ Q_1 + \phi\ P_2\ Q_2$
C7	$\phi\ (\alpha.P) = \phi\ (\alpha.\phi P)$ if $bn(\alpha) \setminus n(\phi) = ;$

R1	$(\nu a : T)(\nu b : S)P = (\nu b : S)(\nu a : T)P$
R2	$(\nu a : T)(P + Q) = (\nu a : T)P + (\nu a : T)Q$
R3	$(\nu a : T)\alpha.P = \alpha.(\nu a : T)P$ if a 62$n(\alpha)$
R4	$(\nu a : T)\alpha.P = 0$ if $subj(\alpha) = a$
R5	$(\nu a : T)[a = u]P = 0$ if a 6= u
R6	$(\nu a : T)[u = v]P = [u = v](\nu a : T)P$ if a 6= u, v

E Assume P $\Sigma_i\ \phi_i\ \alpha_i.P_i$ and Q $\Sigma_j\ \psi_j\ \beta_j.Q_j$ where no α_i (resp. β_j) binds a name free in Q (resp. P). Let Γ ` P j Q. Then infer:

$$P \text{ j } Q = \sum_i \phi_i\ \alpha_i.(P_i \text{ j } Q) + \sum_j \psi_j\ \beta_j.(P \text{ j } Q_j) + \sum_{i\ opp\ j} [\phi_i \wedge \psi_j \wedge (u_i = v_j)]\tau.R_{ij}$$

where α_i opp β_j, u_i, v_j and R_{ij} are defined as follows:
1. α_i is $\bar{u}_i\ w$, β_j is $v_j\ (x : T)$ and $\Gamma(w) <: T$; then R_{ij} is P_i j Q_j f w/xg;
2. α_i is $\bar{u}_i\ (w : S)$, β_j is $v_j\ (x : T)$ and $S <: T$; then R_{ij} is $(\nu w : S)(P_i$ j Q_j f w/xg);
3. the converse of (1) or (2).

A	$P = Q$ if P alpha-equivalent to Q

Table 3. The axiom system A

because $(x)_o = bT$ and no c in satis es the condition $_c$ ` c : bT and can therefore instantiate y. However,

$$x(y : oT):[y = b] \quad 6 \quad x(y : oT):0:$$

To see this, let us look at the possible closing substitutions. In dom($_c$), a is the only channel satisfying $_c$ ` a : (x), and so the only substitution we need to consider is f a=xg. After applying this substitution, the resulting closed terms are not bisimilar:

$$a(y : oT):[y = b] \quad 6 \quad a(y : oT):0$$

This holds because the observer can send b along a and, after the communication, y is instantiated to be b, thus validating the condition y = b and liberating the pre x . When the subject of the pre x is a variable, the following rule is needed in place of lin* :

$$\text{Iv1} \quad \text{If } P = {}_{;y \ : \ (x)_i} Q \text{ then } x(y : T_1):P = \quad x(y : T_2):Q$$

In rule Iout* , both the subject and object of the output pre x are channels. The rule is also valid when the object is a variable. However, it is not valid if the subject is a variable. As a counterexample, let $_c \stackrel{\text{def}}{=}$ a : i T,b : bbT and $\stackrel{\text{def}}{=}$ $_c$; x : bhi T; bTi . Then we have

$$a \quad {}_{u\,a:i\,T} \; 0$$

but

$$xa:a \; 6 \quad xa:0$$

because, under the substitution f b=xg,

$$ba:a \; 6 \quad ba:0:$$

When the subject of the pre x is a variable, we need the following rule:

$$\text{Iv2} \quad \text{If } P = {}_{u\,v: \ (x)_o} Q \text{ then } xv:P = \quad xv:Q$$

We show, by means of an example, why rules lin* and Iout* are su cient in the axiom system (rules Iv1 and Iv2 are derivable). Consider the equality

$$x(y : ii \ T):y \qquad x(y : io \ T):0$$

where $\stackrel{\text{def}}{=}$ a : bib T; b : ib T; x : bib T. First, we infer

$$y = {}_o \, 0 \quad \text{for} \quad {}^0 = \ ; y \ : ib \ T \qquad (1)$$

proceeding as follows (we only report the main rules used in the reasoning; rule C8 is the derived rule given at the beginning of this section):

$$
\begin{array}{lll}
y = {}_{o}\,[y = b]y + [y \neq b]y & & \text{C8} \\
= {}_{o}\,[y = b]y & & \text{Tvar} \\
= {}_{o}\,[y = b]b & & \text{Tpre} \\
= {}_{o}\,[y = b]0 & & \text{Tin ; Icon} \\
= {}_{o}\,0 & & \text{C3}
\end{array}
$$

Then we derive $x(y : ii\ T).y = x(y : io\ T).0$ in a similar way:

$$
\begin{array}{lll}
 & x(y : ii\ T).y & \\
= & [x = a]x(y : ii\ T).y + [x \neq a]x(y : ii\ T).y & \text{C8} \\
= & [x = a]x(y : ii\ T).y & \text{Tvar} \\
= & [x = a]a(y : ii\ T).y & \text{Tpre} \\
= & [x = a]a(y : io\ T).0 & \text{(1); Iin ; Icon} \\
= & x(y : io\ T).0 & \text{Tpre ; Tvar ; C8}
\end{array}
$$

Theorem 1 (Soundness and Completeness of A). $A \vdash P = Q$ i $P \sim Q$.

The schema of the completeness proof is similar to that for the untyped π-calculus [9]. The details, however, are quite different. An example of this is the manipulation of terms underneath input and output prefixes mentioned above. We discuss below another example, related to the issue of invariance of bisimilarity under injective substitutions.

In the untyped case, the process $x \mid \bar{a}$ is equal to $x.\bar{a} + \bar{a}.x +$ when x is instantiated to a, to $x.\bar{a} + \bar{a}.x$ otherwise. This can be expressed by expanding the process by mean of conditions: that is, using conditions to make a case analysis on the possible values that the variable may take. Thus, $x \mid \bar{a}$ is expanded to $[x = a](x \mid \bar{a}) + [x \neq a](x \mid \bar{a})$. Now, underneath $[x = a]$ we know that x will be a, and therefore $x \mid \bar{a}$ can be rewritten as $x.\bar{a} + \bar{a}.x +$, whereas underneath $[x \neq a]$ we know that x will not be a and therefore $x \mid \bar{a}$ can be rewritten as $x.\bar{a} + \bar{a}.x$. In general, the expansion of a process with a free variable x produces a summand $[x \neq a_1] \cdots [x \neq a_n]P$ where $a_1, \ldots, a_n$ are all channels (different from x) that appear free in P. The mismatch $[x \neq a_1] \cdots [x \neq a_n]$ tells us that x in P will be instantiated to a fresh channel, which is sufficient for all manipulations of P involving x, since bisimulation is invariant under injective substitutions. In the typed calculus, by contrast, knowing that x is fresh may not be sufficient: we may also need the information on the type with which x will be instantiated. This type may be different from the type T of x in the type environment: x could be instantiated to a fresh channel whose type is a subtype of T (the behavioral consequences of this type information can be seen in the example at the end of Section 5.1). We have therefore adopted a strategy different from that in the proof for untyped calculi: rather than manipulating processes that begin with "complete" sequences of mismatches | as in the untyped case | we try to cancel them, using rule Tvar*; further, the conditional expansion of a process takes into account also the names that appear in the type environment.

4 A proof system for the closed terms

The system of Section 3 can be simplified if we limit ourselves to proving equalities on closed terms. With one caveat: the substitutivity rule for input is replaced by an inference rule, where (possibly several) instantiations of the bound variable of the input are considered. We call $\mathcal{P}$ the system of rules; it is presented in Table 4.

Rules Tin*, Tout*, Iout*, Itau, Isum, Ires*, Ipar*, Twea*, S1-4, R1-4, E, A as in Table 1, plus the following ones:

Iin If $P\{b/x\} = Q\{b/x\}$ for all b s.t. $\Delta_c \vdash b : \Delta(a)_o$ then $a(x : T_1).P = a(x : T_2).Q$.

Ca $\phi\, P\, Q = P$ if $[\![\phi]\!]$ = True
Cb $\phi\, P\, Q = Q$ if $[\![\phi]\!]$ = False

R $(\nu a : T)0 = 0$

Table 4. The proof system $\mathcal{P}$ for the closed terms

Note that rules Icon, Tvar*, Tpre*, R5-6 are not needed, and that the set of rules C1-7 for conditions is cut down to just Ca-b. In the previous system A, axiom R was redundant in view of C3 and R5.

Theorem 2 (Soundness and completeness of $\mathcal{P}$). $P \vdash P = Q$ iff $P \sim Q$, where P and Q are closed.

Also the proofs of soundness and completeness are simpler. The two main problems in the proofs of Section 3 (manipulations of conditions and of open terms underneath prefixes) do not arise now, because all terms are closed and because conditions are removed as soon as possible, by means of Ca-b, without any conditional expansion. Compared with proof systems for untyped λ-calculus [9], Tin* and Tout* are the main differences.

5 Other equivalences

5.1 Hennessy and Rathke's typed bisimilarity

In the input clause of (Definition 2), the type environment is not extended. By contrast, extensions are allowed in the bisimilarity used in [3]. We denote with the variant of which allows extension; its definition is obtained from that of by using the following input clause: