Static vs Dynamic Typing for Access Control in Pi-Calculus*

Michele Bugliesi, Damiano Macedonio, and Sabina Rossi

Dipartimento di Informatica, Università Ca' Foscari, Venice
{michele,mace,srossi}@dsi.unive.it

Abstract. Traditional static typing systems for the pi-calculus are built around capability types that control the read/write access right on channels and describe the type of their payload. While static typing has proved adequate for reasoning on process behavior in typed contexts, dynamic techniques have often been advocated as more effective for access control in distributed/untyped contexts.
We study the relationships between the two approaches – static versus dinyamic – by contrasting two versions of the asynchronous pi-calculus. The former, AΠ, comes with an entirely standard static typing system. The latter, AΠ@, combines static and dynamic typing: a static type system associates channels with flat types that only express capabilities, disregarding the payload type; a dynamically typed synchronization complements the static type system to guarantee type soundness. We show that AΠ@ can be encoded into AΠ in a fully abstract manner, preserving the respective behavioural equivalences of the two calculi. Besides yielding an interesting expressivity result, the encoding also sheds light on the effectiveness of dynamic typing as a mechanism for access control in untyped contexts.

1 Introduction

Static typing systems have long been established as an effective device to control the interaction among processes in the pi-calculus and related calculi [7, 8, 11, 12]. In these systems the communication channels are viewed as resources, and their types define the capabilities needed to use them. Thus, for instance $\mathsf{rw}\langle S\,;T\rangle$ is the type of a channel where one can output at type T and input at type S (provided that T is a subtype of S). The nested structure of the types makes it possible to control the way that capabilities are delivered and made available. To illustrate, a process knowing the name (or channel) a at the type $\mathsf{rw}\langle\mathsf{r}\langle S\rangle;\mathsf{rw}\langle S\,;S\rangle\rangle$ may output on a a full-fledged channel (with payload type S) and be guaranteed that any (well-typed) process inputing on a will only be reading on the channel received. This form of type-based control yields powerful techniques to reason on the behavior of processes in typed contexts: in fact, by putting enough structure on the types of the shared channels, one may gain strong control on the interaction of a process with any typed context. Unfortunately, these techniques do not scale well, if they scale at all, to more general, potentially untyped, contexts.

To address this shortcoming, [2] introduces a variant of the asynchronous pi-calculus, named AΠ@, in which the ability to control the use of channels relies on a combination

* Work partially supported by M.I.U.R (Italian Ministry of Education, University and Research) under contract n. 2005015785.

of static and dynamic typing. The types of channels are still formed around capabilities, but in ᴀPɪ@ they only exhibit the "top-level" read/write capabilities, disregarding the types of the values transmitted. Given this simple type structure, the type system is much less effective in providing control on the use of channels. To compensate that, ᴀPɪ@ includes a new form of output construct, noted $\overline{a}\langle v@B\rangle$, that relies on a type coercion to enforce the delivery of v at type B. A static typing system guarantees that v may indeed be assigned the coercion type B, while a mechanism of dynamically typed synchronization guarantees that v is received only at supertypes of B, so as to guarantee the type soundness of each exchange.

As argued in [2], the new typing system succeeds in its goal to provide reasoning methods for typed processes in arbitrary contexts. Indeed, [3] shows that the processes of ᴀPɪ@ may be implemented as low-level principals of a cryptographic process calculus based on the applied pi-calculus [1], while preserving their behavioral invariants. The present paper complements the work in [2, 3] by investigating how the combination of static and dynamic typing in ᴀPɪ@ impacts on the ability to control the behavior of processes with respect to traditional systems relying solely on static typing, as in ᴀPɪ.

We develop our approach by studying the (non)existence of *reasonable* cross-encodings between the two calculi. By adapting existing notions of reasonableness from the literature [4–6, 9, 10, 13] to the type framework, we establish the following results.

For the forward direction, we complement a result from [2] and show (i) that there exists a type preserving and *reasonable* encoding of ᴀPɪ into ᴀPɪ@ and (ii) that no encoding with such properties can be fully abstract. For the reverse direction, we show that there exists no (sub)type preserving, reasonable encoding of ᴀPɪ@ into ᴀPɪ. On the other hand, we show that there exists a divergent encoding which is fully abstract. In detail, we provide a sound and complete encoding of (monadic) ᴀPɪ@ into ᴀPɪ with recursive types. The latter result is interesting (and perhaps surprising) as it establishes a connection between ᴀPɪ and the fully abstract implementation developed in [3]. In particular, it allows us to isolate the fragment of ᴀPɪ for which the implementation of [3] is fully abstract.

Plan of the paper. §2 and §3 review the calculi and the definition of reasonable encodings. §4 provides a fully abstract encoding of (monadic) ᴀPɪ@ into ᴀPɪ, §5 concludes with final remarks. Technicalities are in the Appendix.

2 Static and Dynamic Typing in the Pi-Calculus

We presuppose two countable sets of names and variables, ranged over by $a, b, \ldots, n, m$ and $x, y, \ldots$, respectively; u, v range collectively over names and variables whenever the distinction does not matter. $\tilde{u}$, $\tilde{T}$, $\tilde{A}$ denote (possibly empty) tuples of values, and static and dynamic types, respectively; the notation $\tilde{v}@\tilde{A}$ is a shorthand for the tuple $v_1@A_1, \ldots, v_n@A_n$; a corresponding convention applies to $\tilde{v} : \tilde{A}$. Syntactically, ᴀPɪ and ᴀPɪ@ differ only in the form of the communication primitives, and in the structure of the types. The productions are as follows:

π – *Static Typing*

$$\begin{array}{rcl} \textit{Processses } P, Q & ::= & \mathbf{0} \mid P|Q \mid (\nu n{:}S)P \mid !P \mid [u{=}v]P; Q \mid \overline{u}\langle\tilde{v}\rangle \mid u(\tilde{x}).P \\ \textit{Types } S, T & ::= & \top \mid \mathsf{r}\langle\tilde{T}\rangle \mid \mathsf{w}\langle\tilde{T}\rangle \mid \mathsf{rw}\langle\tilde{S};\tilde{T}\rangle \mid X \mid \mu X.T \end{array}$$

$\pi@$ – *Dynamic Typing*

$$\begin{array}{rcl} \textit{Processses } P, Q & ::= & \mathbf{0} \mid P|Q \mid (\nu n{:}A)P \mid !P \mid [u{=}v]P; Q \mid \overline{u}\langle\tilde{v}@\tilde{A}\rangle \mid u(\tilde{x}@\tilde{B}).P \\ \textit{Types } A, B & ::= & \top \mid \mathsf{r} \mid \mathsf{w} \mid \mathsf{rw} \end{array}$$

π is a standard version of the asynchronous pi-calculus with matching, denoted by the construct $[u = v]P; Q$, and recursive capability types á la [7, 8][1]. We use the shorthand $\mathsf{rw}\langle\tilde{T}\rangle$ to mean $\mathsf{rw}\langle\tilde{T};\tilde{T}\rangle$. $\pi@$ replaces the input and output forms from π with new constructs that make explicit the types at which values should be exchanged. As anticipated, the types are reduced to the simplest structure that only represents the capabilities associated with names.

Typing and Subtyping. In both calculi, the subtype relations $<:$ are partially complete pre-orders with a meet operator $\sqcap$ and top type $\top$. In π it is defined as in [7, 8], with the standard extensions needed to handle the presence of recursive types. In $\pi@$ subtyping is generated by the axioms $\mathsf{rw} <: \{\mathsf{r}, \mathsf{w}\} <: \top$. *Type environments*, ranged over by $\Gamma, \Gamma' \ldots$, are finite mappings from names and variables to types. We write $\Gamma \vdash P$ to mean that P is well typed in Γ. The type environment $\Gamma, \langle u : T\rangle$ is $\Gamma, u : T$ if $u \notin dom(\Gamma)$; otherwise it is the type environment Γ' such that $\Gamma'(v) = \Gamma(v)$ for $v \neq u$ and $\Gamma'(u) = \Gamma(u) \sqcap T$ if $\Gamma(u) \sqcap T$ is defined. Subtyping is extended to type environments as expected: $\Gamma <: \Gamma'$ whenever $dom(\Gamma) = dom(\Gamma')$ and for all $v \in dom(\Gamma)$ it holds $\Gamma(v) <: \Gamma'(v)$.

Operational Semantics. The dynamics of π is expressed by the usual labelled transition system of the asynchronous pi-calculus. On the other hand, in order to express the dynamics for $\pi@$, the input/output labels of pi-calculus are extended with a type capability in order to force the synchronisation at the desired type. Below, we discuss the different ways of synchronisation, and refer the reader to the appendix for full details.

$$\textit{Dynamic Typing:} \quad \frac{\overline{a}\langle\tilde{v}@\tilde{A}\rangle \xrightarrow{\overline{a}\langle\tilde{v}@\tilde{A}\rangle} \mathbf{0} \quad a(\tilde{x}@\tilde{B}).P \xrightarrow{a(\tilde{v}@\tilde{B})} P\{\tilde{v}/\tilde{x}\} \quad \tilde{A} <: \tilde{B}}{P|Q \xrightarrow{\tau} P\{\tilde{v}/\tilde{x}\}|\mathbf{0}}$$

$$\textit{Static Typing:} \quad \frac{\overline{a}\langle\tilde{v}\rangle \xrightarrow{\overline{a}\langle\tilde{v}\rangle} \mathbf{0} \quad a(\tilde{x}).P \xrightarrow{a(\tilde{v})} P\{\tilde{v}/\tilde{x}\}}{P|Q \xrightarrow{\tau} P\{\tilde{v}/\tilde{x}\}|\mathbf{0}}$$

In $\pi@$ the labels carry extra information about the types at which names are exchanged. More interestingly, however, these rules show the fundamentally different nature of the interaction with the context: in π the context acquires the names emitted at the read type of the transmission channel, while in $\pi@$ the type $\tilde{B}$ is decided by the process. In addition, processes of both calculi may synchronize with a τ transition when exhibiting complementary labels that, in the case of $\pi@$, are required to agree on the type of the values exchanged, as in $a(\tilde{v}@\tilde{A})$ and $\overline{a}\langle\tilde{v}@\tilde{A}\rangle$.

[1] In fact, in [7, 8] the types are not recursive; however, as far as we see, the generalization is harmless for the results relevant to our present endeavor.

The notion of observational equivalence is based on the usual notion of barbed equivalence and is mediated by the type capabilities that the contexts possess on the names shared with the processes. It relies on the definition of *configurations* of the form $\mathcal{I} \triangleright P \xrightarrow{\alpha} \mathcal{I}' \triangleright P'$, which identify the actions of the process P that may be observed by the type environment $\mathcal{I}$. Thus $\mathcal{I}$ represents what contexts know of the names shared with P: remarkably, P may know those names at more precise types than $\mathcal{I}$, as $\mathcal{I} \triangleright P$ is a well-defined configuration only if there exists $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$.

Definition 1 (Type-indexed Relation). *A type-indexed relation $\mathcal{R}$ is a family of binary relations between processes indexed by type environments. We write $\mathcal{I} \models P \mathcal{R} Q$ to mean (i) that P and Q are related by $\mathcal{R}$ and (ii) that $\mathcal{I} \triangleright P$ and $\mathcal{I} \triangleright Q$ are configurations.*

Definition 2 (Contextuality). *A type-indexed relation $\mathcal{R}$ is contextual when:*

1. *$\mathcal{I} \models P \mathcal{R} Q$ implies $\mathcal{I}, a : A \models P \mathcal{R} Q$ with $a \notin dom(\mathcal{I})$*
2. *$\mathcal{I} \models P \mathcal{R} Q$ and $\mathcal{I} \models R$ imply $\mathcal{I} \models P|R \mathcal{R} Q|R$*
3. *$\mathcal{I}, a : A \models P \mathcal{R} Q$ implies $\mathcal{I} \models (\nu a{:}A)P \mathcal{R} (\nu a{:}A)Q$*

The definition of barbs is based on the actual capability of the context $\mathcal{I}$ to see barbs. The notation $\mathcal{I}^{\mathsf{r}}(a) \downarrow$ indicates that $\mathcal{I}$ (hence the context) has a read capability on the name a, i.e., $\mathcal{I}(a) <: \mathsf{r}$, only in that case the output action performed by the process is observable by the context. Moreover we denote by $\Longrightarrow$ the reflexive and transitive closure of $\xrightarrow{\tau}$.

Definition 3 (Barbs). *Given a configuration $\mathcal{I} \triangleright P$, we say that*

1. *$\mathcal{I} \triangleright P{\downarrow_a}$ if and only if $\mathcal{I}^{\mathsf{r}}(a) \downarrow$ and $P \xrightarrow{(\tilde{c}:\tilde{C})\bar{a}\langle \tilde{v}@\tilde{A}\rangle}$.*
2. *$\mathcal{I} \triangleright P{\Downarrow_a}$ if and only if $P \Longrightarrow P'$ and $\mathcal{I} \triangleright P'{\downarrow_a}$.*

Definition 4 (Typed Behavioural Equivalence). *The typed behavioural equivalence is the largest symmetric, contextual and type-indexed relation $\mathcal{R}$ such that $\mathcal{I} \models P \mathcal{R} Q$ implies*

1. *if $\mathcal{I} \models P{\downarrow_a}$ then $\mathcal{I} \models Q{\Downarrow_a}$*
2. *if $Q \xrightarrow{\tau} Q$ then there exists Q' such that $Q \Longrightarrow Q'$ and $\mathcal{I} \models P' \mathcal{R} Q'$.*

The typed behavioural equivalence is denoted as $\approx_{\mathrm{S}}$ in $\mathrm{A}\pi\mathrm{I}$ and as $\approx_{\mathrm{D}}$ in $\mathrm{A}\pi\mathrm{I}@$.

3 Properties of Encodings

Formally, an *encoding* is a map from terms (processes) of a *source* language to terms of a *target* language. However, when dealing with typed languages, where the behavior of systems strongly depends on the capabilities of the observing context, it is reasonable to encode not just processes but rather configurations of the form $\mathcal{I} \triangleright P$. More precisely, we build the encoding $[\![\mathcal{I} \triangleright P]\!]$ upon two independent encodings: $[\![\mathcal{I}]\!]$ for the context and $[\![P]\!]_{\Gamma}$ for the process P with respect to a type environment Γ such that $\Gamma \vdash P$.

We assume that the encoding of typing environments satisfies the following two natural requirements:

Typing Preservation - if $\Gamma \vdash P$, then there exists $\Gamma' <: [\![\Gamma]\!]$ such that $\Gamma' \vdash [\![P]\!]_\Gamma$;

Subtyping Preservation - if $\Gamma <: \Gamma'$, then $[\![\Gamma]\!] <: [\![\Gamma']\!]$.

As stated below, these two requirements are enough to guarantee that the encoding of configurations is *well defined*.

Proposition 1. *Let $[\![\ ]\!]$ be an encoding satisfying both typing and subtyping preservation. Let $\mathcal{I} \triangleright P$ be a configuration in the source calculus and Γ be a type environment such that $\Gamma <: \mathcal{I}$ and $\Gamma \vdash P$. Then $[\![\mathcal{I}]\!] \triangleright [\![P]\!]_\Gamma$ is a configuration in the target calculus.*

Proof. Let $\mathcal{I} \triangleright P$ be a configuration and $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$. By typing preservation there exists $\Gamma' <: [\![\Gamma]\!]$ such that $\Gamma' \vdash [\![P]\!]_\Gamma$. By subtyping preservation $[\![\Gamma]\!] <: [\![\mathcal{I}]\!]$, hence $\Gamma' <: [\![\mathcal{I}]\!]$. Thus $[\![\mathcal{I}]\!] \triangleright [\![P]\!]_\Gamma$ is a configuration.

As for the encoding of processes, there is no agreement on what should be a *good* notion of encoding. However, it seems reasonable to require that some intended semantic properties are preserved and/or reflected by the encoding. A criterion for being a good encoding is the popular notion of *full abstraction* which requires that the equivalence of source terms is both preserved and reflected. Preservation provides behavioral completeness while reflection provides behavioural soundness.

Let $\approx_s$ and $\approx_t$ denote equivalences of the source and target language, respectively. Then, fhe full abstraction property is formulated as follows. Let Γ, Γ' be two type environments such that $\Gamma <: \mathcal{I}$, $\Gamma' <: \mathcal{I}$, $\Gamma \vdash P$ and $\Gamma' \vdash P'$. An encoding $[\![\ ]\!]$ is *fully abstract* if it is both sound and complete:

Soundness (reflection of equivalence) -

$$[\![\mathcal{I}]\!] \models [\![P]\!]_\Gamma \approx_t [\![P']\!]_{\Gamma'} \text{ implies } \mathcal{I} \models P \approx_s P'.$$

Completeness (preservation of equivalence) -

$$\mathcal{I} \models P \approx_s P' \text{ implies } [\![\mathcal{I}]\!] \models [\![P]\!]_\Gamma \approx_t [\![P']\!]_{\Gamma'}.$$

4 Encoding APi@ into APi

In this section we presenta a fully abstract encoding of APi@ into APi. We first give the details of how this can be done for the monadic fragment of APi@. In § 5, we briefly discuss how the approach can be generalized to the polyadic calculus.

4.1 First attempt

We start with a relatively simple approach, which almost work, but not quite. The idea is to represent each name n in APi@ as a 4-tuple of names $\underline{n} = (n_{\mathsf{rw}}, n_{\mathsf{r}}, n_{\mathsf{w}}, n_{\top})$, where each component of the tuple corresponds to one of the four types at which a synchronization may take place on n. Thus, an APi@ synchronization on n at, say, the type w, will correspond in APi to a synchronization on n_{w}. Clearly, this idea must be applied consistently, hence exchanging a name in APi@ will correspond to exchanging a tuple

Table 1 Encoding functions (with $Q = u(\tilde{x}).P$)

$$
\begin{array}{rcll}
\textsc{Read}_l\,\langle Q\rangle & \stackrel{def}{=} & u(\tilde{x}).\textsc{Test}_l\,\langle Q\rangle & \\
\textsc{Test}_l\,\langle Q\rangle & \stackrel{def}{=} & l(z).[z=\mathtt{t}]\textsc{Commit}_l\,\langle Q\rangle \oplus \textsc{Undo}_l\,\langle Q\rangle\,;\,\textsc{Abort}_l\,\langle Q\rangle & z \text{ fresh} \\
\textsc{Undo}_l\,\langle Q\rangle & \stackrel{def}{=} & \bar{l}\langle\mathtt{t}\rangle|\bar{u}\langle\tilde{x}\rangle & \\
\textsc{Commit}_l\,\langle Q\rangle & \stackrel{def}{=} & \bar{l}\langle\mathtt{f}\rangle|P & \\
\textsc{Abort}_l\,\langle Q\rangle & \stackrel{def}{=} & \bar{l}\langle\mathtt{f}\rangle|\bar{u}\langle\tilde{x}\rangle & \\
R_1 \oplus R_2 & \stackrel{def}{=} & (\nu i:\mathsf{rw}\langle\,\rangle)(\bar{i}\langle\,\rangle \mid i(\,).P \mid i(\,).Q) & i \text{ fresh}
\end{array}
$$

in aPi. Hence, the output $\bar{u}\langle v@A\rangle$ can be represented as the corresponding output $\overline{u_A}\langle\underline{v}\rangle$. The input prefix requires a little more care, as in aPi@ the process $u(x@A).P$ may synchronize with any output on u at a type $B <: A$. In fact, given the present choice for the representation of the output construct, it is easily seen that the behavior of the input form $u(x@A).P$ corresponds precisely to the input guarded choice $\Sigma_{B<:A}u_B(\underline{x}).P$[2].

We may then formalize our first attempt by combining the intuitions just outlined with the encoding of guarded choice from [9]. This results into the following definition.

$$
\begin{array}{rcl}
(\!|\,\bar{u}\langle v@A\rangle\,|\!) & \stackrel{def}{=} & \overline{u_A}\langle\underline{v}\rangle \\
(\!|\,u(x@A).P\,|\!) & \stackrel{def}{=} & (\nu l:\mathsf{rw}\langle\top\rangle)\left(\bar{l}\langle\mathtt{t}\rangle|\prod_{B<:A}!\,\textsc{Read}_l\left\langle u_B(\underline{x}).(\!|\,P\,|\!)\right\rangle\right) \\
(\!|\,(\nu\, n)P\,|\!) & \stackrel{def}{=} & (\nu\,\underline{n})(\!|\,P\,|\!) \\
(\!|\,[u=v]P;\,Q\,|\!) & \stackrel{def}{=} & [u_\top = v_\top](\!|\,P\,|\!);(\!|\,Q\,|\!)
\end{array}
$$

Just as in [9], the encoding of input runs a mutual exclusion protocol, installing a local lock on a parallel composition of its branches. The protocol is implemented by the processes in Table 1, which we inherit, essentially unchanged, from [9]. The branches $\textsc{Read}_l\,\langle -\rangle$ concurrently try to test the lock after reading messages from the environment. Every branch can 'black out' and return to its initial state after it has taken the lock, just by resending the message. Just one branch will proceed with its continuation and thereby commit the input. Every other branch will then be forced to resend its message and abort its continuation.

This re-sending of messages by 'non-chosen' and 'blacked out' inputs is what makes the encoding diverge, thus making the encoding not reasonable. Given the result of Theorem (che non c'e' piu'...) this is not surprising. Unfortunately, however, the re-sending of messages is problematic also for type preservation. To see the problem notice that a aPi@ process may have just the read capability on a channel in order to perform an input, while its encoding must also be granted a write capability in order to run the mutual exclusion protocol.

The typing failure would not arise if we dropped the type r, and worked with just three types: indeed, for this fragment of aPi@ the encoding we just illustrated may be shown to be type-preserving and fully abstract. For a more general, and robust solution, one would need to move the responsability of running the mutual exclusion protocol from the (encoding of the) input process to some other process possessing the required capabilities.

[2] For uniformity with the notation adopted for names, we write $\underline{x}$ to note a quadruple of variables used to store name representations.

Table 2 Encoding of aPi@ into aPi – Types.

Client types

$\mathbb{T}_{\mathsf{rw}} \stackrel{def}{=} (\mathbb{R}, \mathbb{W}) \quad \mathbb{T}_{\mathsf{r}} \stackrel{def}{=} (\mathbb{R}, \mathsf{T}) \quad \mathbb{T}_{\mathsf{w}} \stackrel{def}{=} (\mathsf{T}, \mathbb{W}) \quad \mathbb{T}_{\top} \stackrel{def}{=} (\mathsf{T}, \mathsf{T})$

$\mathbb{R} \stackrel{def}{=} (\boldsymbol{T}_{\mathsf{r@rw}}, \boldsymbol{T}_{\mathsf{r@r}}, \boldsymbol{T}_{\mathsf{r@rw}}, \boldsymbol{T}_{\mathsf{r@}\top}) \quad \mathbb{W} \stackrel{def}{=} (\boldsymbol{T}_{\mathsf{w@rw}}, \boldsymbol{T}_{\mathsf{w@r}}, \boldsymbol{T}_{\mathsf{w@rw}}, \boldsymbol{T}_{\mathsf{w@}\top})$

$\boldsymbol{T}_{\mathsf{r@rw}} \stackrel{def}{=} \mathsf{w}\langle\mathsf{w}\langle\mathbb{R}, \mathbb{W}\rangle\rangle \qquad \boldsymbol{T}_{\mathsf{w@rw}} \stackrel{def}{=} \mathsf{w}\langle\mathbb{R}, \mathbb{W}\rangle$

$\boldsymbol{T}_{\mathsf{r@r}} \stackrel{def}{=} \mathsf{w}\langle\mathsf{w}\langle\mathbb{R}, \mathsf{T}\rangle\rangle \qquad \boldsymbol{T}_{\mathsf{w@r}} \stackrel{def}{=} \mathsf{w}\langle\mathbb{R}, \mathsf{T}\rangle$

$\boldsymbol{T}_{\mathsf{r@w}} \stackrel{def}{=} \mathsf{w}\langle\mathsf{w}\langle\mathsf{T}, \mathbb{W}\rangle\rangle \qquad \boldsymbol{T}_{\mathsf{w@w}} \stackrel{def}{=} \mathsf{w}\langle\mathsf{T}, \mathbb{W}\rangle$

$\boldsymbol{T}_{\mathsf{r@}\top} \stackrel{def}{=} \mathsf{w}\langle\mathsf{w}\langle\mathsf{T}, \mathsf{T}\rangle\rangle \qquad \boldsymbol{T}_{\mathsf{w@}\top} \stackrel{def}{=} \mathsf{w}\langle\mathsf{T}, \mathsf{T}\rangle$

Server types

$\mathbb{S} \stackrel{def}{=} (\mathbb{R}_{\mathbb{S}}, \mathbb{W}_{\mathbb{S}})$

$\mathbb{R}_{\mathbb{S}} \stackrel{def}{=} (\boldsymbol{S}_{\mathsf{r@rw}}, \boldsymbol{S}_{\mathsf{r@r}}, \boldsymbol{S}_{\mathsf{r@w}}, \boldsymbol{S}_{\mathsf{r@}\top}) \quad \mathbb{W}_{\mathbb{S}} \stackrel{def}{=} (\boldsymbol{S}_{\mathsf{w@rw}}, \boldsymbol{S}_{\mathsf{w@r}}, \boldsymbol{S}_{\mathsf{w@w}}, \boldsymbol{S}_{\mathsf{w@}\top})$

$\boldsymbol{S}_{\mathsf{r@rw}} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{w}\langle\mathbb{R}, \mathbb{W}\rangle\rangle \qquad \boldsymbol{S}_{\mathsf{w@rw}} \stackrel{def}{=} \mathsf{rw}\langle\mathbb{R}, \mathbb{W}\rangle$

$\boldsymbol{S}_{\mathsf{r@r}} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{w}\langle\mathbb{R}, \mathsf{T}\rangle\rangle \qquad \boldsymbol{S}_{\mathsf{w@r}} \stackrel{def}{=} \mathsf{rw}\langle\mathbb{R}, \mathsf{T}\rangle$

$\boldsymbol{S}_{\mathsf{r@w}} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{w}\langle\mathsf{T}, \mathbb{W}\rangle\rangle \qquad \boldsymbol{S}_{\mathsf{w@w}} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{T}, \mathbb{W}\rangle$

$\boldsymbol{S}_{\mathsf{r@}\top} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{w}\langle\mathsf{T}, \mathsf{T}\rangle\rangle \qquad \boldsymbol{S}_{\mathsf{w@}\top} \stackrel{def}{=} \mathsf{rw}\langle\mathsf{T}, \mathsf{T}\rangle$

Type Environments

$\{\!|\, \emptyset \,|\!\} \stackrel{def}{=} \mathtt{t} : \top, \mathtt{f} : \top \qquad \{\!|\, \Gamma, v : A \,|\!\} \stackrel{def}{=} \{\!|\, \Gamma \,|\!\}, (\boldsymbol{v}) : \mathbb{T}_A$

4.2 A sound encoding

The solution is inspired by [3] and based on the representation of a channel as a process that serves input and output requests, so that each exchange is the result of two separate protocols. Given a name n, we write $Chan(n)$ for the server associated with n. In the input protocol, a client willing to input on u and type A sends a read request (in the form of a private name) on the channel $n_{\mathsf{r}@A}$. In the write protocol, a process willing to output on n and type $B <: A$ sends its output on the channel $n_{\mathsf{w}@B}$. $Chan(n)$ is the only process with read and write access to $n_{\mathsf{w}@B}$ and $n_{\mathsf{r}@A}$, while clients will, at their best, have write capabilities on these names. The server $Chan(n)$ may therefore run the synchronization protocol and send back the selected message on the private name received by the reader client.

Collectively, each name n from aPi@ is translated into the 8-tuple $(\boldsymbol{n})$, where the components $n_{\mathbb{R}} = n_{\mathsf{r@rw}}, n_{\mathsf{r@r}}, n_{\mathsf{r@w}}, n_{\mathsf{r@}\top}$ are the names employed in the input protocol to communicate the input requests at the corresponding types, while the components $n_{\mathbb{W}} = n_{\mathsf{w@rw}}, n_{\mathsf{w@r}}, n_{\mathsf{w@w}}, n_{\mathsf{w@}\top}$ serve the same purpose for the output protocol. Thus, for instance, the output $\overline{n}\langle v@\mathsf{rw}\rangle$ corresponds to the output $\overline{n_{\mathsf{w@rw}}}\langle\boldsymbol{v}\rangle$ and synchronizes only with $n_{\mathsf{w@rw}}(\boldsymbol{x})$. The the input $n(x@\mathsf{rw}).P$, in turn, first performs the output $\overline{n_{\mathsf{r@rw}}}\langle l\rangle$, where l is a private channel that will be used by $Chan(n)$ to reply back with a suitable (tuple of) value(s).

Typewise, a read capability on n in aPi@ corresponds in aPi to a write capability on all the names in $n_{\mathbb{R}}$, while a write capability on n corresponds to a write capability on the names $n_{\mathbb{W}}$. The encoding of types is given in full detail in Table 2. With each type A in aPi@ we associate a corresponding tuple of types $\mathbb{T}_A$, as in $\mathbb{T}_{\mathsf{rw}} = (\mathbb{R}, \mathbb{W})$ or $\mathbb{T}_{\mathsf{w}} = (\mathsf{T}, \mathbb{W})$ where $\mathbb{R}$ and $\mathbb{W}$ are the client types associated to the names employed in the read/write protocols (according to the convention that $n_{\mathsf{r}@A} : \boldsymbol{T}_{\mathsf{r}@A}$ and $n_{\mathsf{w}@A} : \boldsymbol{T}_{\mathsf{w}@A}$)

Table 3 Encoding of aPi@ into aPi – Processes.

Clients

$$\begin{aligned}
(\!| \mathbf{0} |\!) &\stackrel{def}{=} \mathbf{0} \\
(\!| \overline{u}\langle v@A\rangle |\!) &\stackrel{def}{=} \overline{u_{w@A}}\langle \boldsymbol{v}\rangle \\
(\!| u(x@A).P |\!) &\stackrel{def}{=} (\boldsymbol{\nu} h : \mathsf{rw}\langle \mathbb{T}_A\rangle)\,(\,\overline{u_{r@A}}\langle h\rangle \,|\, h(\boldsymbol{x}).(\!| P |\!)\,) \quad \text{where the name } h \text{ is fresh} \\
(\!| P \,|\, Q |\!) &\stackrel{def}{=} (\!| P |\!) \,|\, (\!| Q |\!) \\
(\!| (\boldsymbol{\nu} a : A) P |\!) &\stackrel{def}{=} (\boldsymbol{\nu}(\boldsymbol{a}) : \mathbb{S})\,((\!| P |\!) \,|\, \textsc{Chan}(a)) \\
(\!| [u = v] P;\, Q |\!) &\stackrel{def}{=} [u_{r@r} = v_{r@r}](\!| P |\!);(\!| Q |\!) \\
(\!| \,!P |\!) &\stackrel{def}{=} \,!(\!| P |\!)
\end{aligned}$$

Channel Servers

$$\begin{aligned}
\textsc{Chan}(\boldsymbol{u}) &\stackrel{def}{=} \Pi_A \,!\, u_{r@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h) \\
\textsc{Choose}(\boldsymbol{u}, A, h) &\stackrel{def}{=} (\boldsymbol{\nu} l : \mathsf{rw}\langle \top\rangle)\left(\, \overline{l}\langle \mathsf{t}\rangle \,|\, \textstyle\prod_{B<:A} \,!\, \textsc{Read}_l \left\langle u_{w@B}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle \right\rangle\right)
\end{aligned}$$

Complete Systems

$$(\!| P |\!)_\Gamma \stackrel{def}{=} (\!| P |\!) \mid \textstyle\prod_{a \in dom(\Gamma)} \textsc{Chan}(a)$$

and $\top$ is the representation of the top type in aPi@. Notice that clients are only granted write capabilities on the channels involved in the protocols, while the channel servers know the same names at the lower types $\mathbb{R}_\mathbb{S}$ and $\mathbb{W}_\mathbb{S}$ which grant them full access to those names.

The processes of aPi@ are then encoded as the clients of Table 3, which also defines the channel servers, and the encoding of a complete system which includes channel servers for all the free names of the system. The new encoding is well-defined as stated below.

Theorem 1 (Type preservation). *If $\Gamma \vdash P$ in* aPi@, *then there exists $\Gamma' <: (\!| \Gamma |\!)$ such that $\Gamma' \vdash (\!| P |\!)_\Gamma$ in* aPi.

Also, we can show that the encoding is sound. The soundness proof follows a standard pattern, but relies on a non-standard, and lengthy proof of operational correspondence. The reason is that the encoding is not "prompt" [9], in that it takes several steps for the encoding of a process to be ready for the commit action that corresponds to the aPi@ synchronization on the channel. For this reason, the proof of soundness is not standard, but it relies on the technique introduced in [3]. Essentially, we classify the transitions of the encodings into two main groups: administrative and effective. The administrative actions, noted by Λ, represent the transitions introduced by the encoding in order to synchronize with the original process and enable the performance of the effective actions. The effective actions, noted by λ, actually correspond to actions in the encoded calculus. In this case, the administrative actions are the input actions and the internal synchronizations performed by the channel server, along with all the synchronizations between the client processes and the channel server itself. This classification of actions introduces a weaker form of behavioral equivalence, the *administrative bisimulation* $\approx_A$, defined as in Definition 4, where $\Longrightarrow$ is replaced by $\xrightarrow{A_\Lambda}$, which represents a (possibly empty) sequence of administrative actions, and $\overset{\lambda}{\Longrightarrow}$ is replaced by $\xrightarrow{A_\Lambda}\!\!\rightarrow\xrightarrow{\lambda}\!\!\rightarrow\xrightarrow{A_\Lambda}$. The definition implies that the administrative equivalence is closed under administrative re-

duction, namely if $P \xrightarrow{A_\lambda} P'$ then $P \approx_A P'$. The main property of the equivalence is the following correspondence.

Proposition 2. *Let P and Γ such that $\Gamma \vdash P$ in* aPi@. *Then*

1. *If $P \xrightarrow{\mu} P'$ in* aPi@ *then there exist an effective action λ and a process Q in* aPi *such that $\{\!|P|\!\}_\Gamma \xrightarrow{A_\lambda} \xrightarrow{\lambda} Q$ and $Q \approx_A \{\!|P'|\!\}_\Gamma$;*
2. *If $Q \approx_A \{\!|P|\!\}_\Gamma$ and $Q \xrightarrow{\alpha} Q'$ in* aPi, *then there exist a process P' and an action μ in* aPi@ *such that $P \overset{\mu}{\Longrightarrow} P'$ and $Q' \approx_A \{\!|P'|\!\}$.*

Based on this form of operational correspondence, soundness is proved in a standard way: first by observing that $\approx_A$ is finer than $\approx_s$, then by defining a type indexed relation $\mathcal{R}$ as $\mathcal{I} \models P\,\mathcal{R}\,P'$ iff $\{\!|I|\!\} \models \{\!|P|\!\}_\Gamma \approx_s \{\!|P'|\!\}_{\Gamma'}$ for some Γ, Γ' compatible with $\mathcal{I}$ and such that $\Gamma \vdash P$ and $\Gamma' \vdash P'$, and finally by showing that $\mathcal{R}$ is an asynchronous bisimulation on aPi@.

Theorem 2 (Soundness). *Let Γ and Γ' be type environments compatible with $\mathcal{I}$ such that $\Gamma \vdash P$ and $\Gamma' \vdash Q$ in* aPi@. *Then $\{\!|\mathcal{I}|\!\} \models \{\!|P|\!\}_\Gamma \approx_s \{\!|Q|\!\}_{\Gamma'}$ implies $\mathcal{I} \models P \approx_D Q$.*

The converse direction of Theorem 2 does not hold. The problem is that the properties of the communication protocols are based on certain invariants for the channel servers. Now, these invariants are verified by the servers that are statically allocated by the encoding, but this may fail to be true of the names, and the associated servers generated dynamically by the context. Given that, it is easy to find a counter-example to full abstraction.

Example 1. Take for instance $P \stackrel{def}{=} a(x@\mathsf{rw}).x(y@\mathsf{rw}).\overline{x}\langle y@\mathsf{rw}\rangle$ and $P' \stackrel{def}{=} a(x@\mathsf{rw}).\mathbf{0}$. It is easily verified that in the aPi@ one has $a : \mathsf{w} \models P \approx_D P'$. Now take $\Gamma \stackrel{def}{=} a : \mathsf{rw}$, and consider the encoding of the two processes $\{\!|P|\!\}_\Gamma = \{\!|P|\!\} \,|\, \textsc{Chan}(a)$ and $\{\!|P'|\!\}_\Gamma = \{\!|P'|\!\} \,|\, \textsc{Chan}(a)$, where:

$$\begin{array}{l} \{\!|P|\!\} = (\nu h : \mathsf{rw}\langle \mathbb{T}_{\mathsf{rw}} \rangle)(\overline{a_{\mathsf{r@rw}}}\langle h\rangle \,|\, h(\boldsymbol{x}).(\nu k : \mathsf{rw}\langle \mathbb{T}_{\mathsf{rw}} \rangle)(\overline{x_{\mathsf{r@rw}}}\langle k\rangle \,|\, k(\boldsymbol{y}).\overline{x_{\mathsf{w@rw}}}\langle \boldsymbol{y}\rangle)) \\ \{\!|P'|\!\} = (\nu h : \mathsf{rw}\langle \mathbb{T}_{\mathsf{rw}} \rangle)(\overline{a_{\mathsf{r@rw}}}\langle h\rangle \,|\, h(\boldsymbol{x}).\mathbf{0}) \end{array}$$

Now we have the following transition: $\{\!|\Gamma|\!\} \triangleright \{\!|P|\!\}_\Gamma \overset{(\boldsymbol{b}:\mathbb{S})a_{\mathsf{w@rw}}(\boldsymbol{b})}{\Longrightarrow} \{\!|\Gamma|\!\}, \boldsymbol{b} : \mathbb{S} \triangleright Q$ where $\mathbb{S}$ is the server type defined in Table 2, and $\{\!|\Gamma|\!\}, \boldsymbol{b} : \mathbb{S} \models Q \approx_s (\nu k : \mathsf{rw}\langle \mathbb{T}_{\mathsf{rw}} \rangle)(\overline{b_{\mathsf{r@rw}}}\langle k\rangle \,|\, k(\boldsymbol{y}).\overline{b_{\mathsf{w@rw}}}\langle \boldsymbol{y}\rangle)$. Since $b : \mathbb{S}$ implies $b_{\mathsf{r@rw}} : \mathsf{rw}\langle \mathsf{w}\langle \mathbb{T}_{\mathsf{rw}}\rangle\rangle$, the configuration $\{\!|\Gamma|\!\}, (\boldsymbol{b}) : \mathbb{S} \triangleright Q$ offers a barb on $b_{\mathsf{r@rw}}$, which is not offered by any of the configurations reached by $\{\!|\Gamma|\!\} \triangleright \{\!|P'|\!\}_\Gamma$. Thus $\{\!|\Gamma|\!\} \models \{\!|P|\!\}_\Gamma \not\approx_s \{\!|P'|\!\}_\Gamma$, which shows that the encoding is not complete.

4.3 A full abstract encoding

In order to recover full abstraction, we adopt the idea of [3], which introduces a *proxy server* that manages the communications between the channel server, the context and the processes. The new encoding introduces a separation between *client names*, used by the context and the translated processes to communicate, and the corresponding *proxy*

names, generated within the system and associated with system generated channels which are employed in the actual protocols for communication.

The proxy server, PROXY, maintains an association table between client and server names in order to preserve the expected interactions among clients. The read/write protocol follows the same rationale as in the previous encoding, with the difference that in this case the client must first obtain the access to the system channel with a request to PROXY. The interaction between clients and PROXY is as follows: the client presents a name to the proxy, and the proxy replies with the corresponding server name, which is cast at the (true) type of the name sent by the client. When the proxy server receives a name for the first time, it allocates a fresh server channel that will be always associated to that name, and in addition it defines a channel server for the new channel.

The definition of the proxy server, in Table 4, is essentially a typed variant of the corresponding process introduced in [3]. SERVER is a process always ready to receive inputs along the four channels p_A, one for each $A \in \{\mathsf{rw}, \mathsf{r}, \mathsf{w}, \top\}$, which are devoted to the requests from the clients. These channels are known to the clients and to the context at the type $p_A : \mathsf{w}\langle \mathsf{w}\langle \mathbb{T}_A \rangle, \mathbb{T}_A \rangle$, so to guarantee the reading capability only to SERVER. After receiving an input on p_A, the SERVER starts a search on the association table and replies with the requested system name. We omit the largely obvious details of the implementation of the association table. We limit the formalization of the the table to a process $t[\mathcal{T}]$, where $\mathcal{T}$ represents the structure of the table and the name $t : \mathsf{tbl}$ allows access to its entries. Essentially, $\mathcal{T}$ is a mapping between channel names and server names. We write $(\boldsymbol{n}, \boldsymbol{k})$ to say that the tuple of names $\boldsymbol{n}$ is associated to the tuple $\boldsymbol{k}$, meaning that the name $n_{\mathsf{r@rw}}$ is associated to $k_{\mathsf{r@rw}}$, the name $n_{\mathsf{r@r}}$ to $k_{\mathsf{r@r}}$ and so on... The proxy server The proxy refers to the table $t[\mathcal{T}]$ by means of the the process $\textsc{find}(\boldsymbol{n}, t, r)$, where $\boldsymbol{n} : \mathbb{T}_\top$ represents the entry (the client name) to find, $t : \mathsf{tbl}$ is the table name and $r : \mathsf{w}\langle \top, \mathbb{S} \rangle$ is the reply channel where to send the proxy name associated along with a boolean flag which says whether a new entry was created in the table as a consequence of the request. The process $\textsc{find}(\boldsymbol{n}, t, r)$ scrolls the entries of $t[\mathcal{T}]$ and replies on r the tuple $\boldsymbol{k}$ associated to n and $\mathtt{t}$, in case the entry for $\boldsymbol{n}$ was already defined, or $\mathtt{f}$, in case the entry for $\boldsymbol{n}$ has been defined just to satisfy the current request. Operationally we assume that

$$\textsc{find}(\boldsymbol{n}, t, r) \,|\, t[\mathcal{T}] \xrightarrow{\tau} \overline{r}\langle \mathtt{t}, \boldsymbol{k} \rangle \,|\, t[\mathcal{T}]$$

in case $(\boldsymbol{n}, \boldsymbol{k}) \in \mathcal{T}$, and that

$$\textsc{find}(\boldsymbol{n}, t, r) \,|\, t[\mathcal{T}] \xrightarrow{\tau} (\nu \boldsymbol{k} : \mathbb{S})\,(\,\overline{r}\langle \mathtt{f}, \boldsymbol{k} \rangle \,|\, t[(\boldsymbol{n}, \boldsymbol{k}) :: \mathcal{T}]\,) \quad \text{with } \boldsymbol{k} \text{ fresh in } \mathcal{T}$$

in case there is no entry for $\boldsymbol{n}$ in $\mathcal{T}$, so that a new entry is created and it is associated to a fresh tuple $\boldsymbol{k}$. The proxy server then send back the proxy name to the client and sets a channel manager for the proxy channel in case the first component of the reply on r is $\mathtt{f}$.

On the client side, the encoding is similar to the one given in Table 3. We only list the most significant clauses in Table 4, and comment on the differences with respect to the previous encoding below. First, the read and write protocol now must first link the client name to the corresponding server name, and then it proceeds as described in §4.2.

Table 4 Fully Abstract Encoding of aPi@ into aPi.

Proxy

$$\textsc{Proxy} \stackrel{def}{=} (\nu t:\mathsf{tbl})\,(\textsc{Server} \mid t[\,])$$
$$\textsc{Server} \stackrel{def}{=} \Pi_A\, !\, p_A(h,\boldsymbol{z}).(\nu r:\mathsf{rw}\langle\top,\mathbb{S}\rangle)\textsc{Send}(\boldsymbol{z},h,r)$$
$$\textsc{Send}(\boldsymbol{z},h,r) \stackrel{def}{=} \Big(\textsc{find}(\boldsymbol{z},t,r) \mid r(x,\boldsymbol{y})[x=\mathsf{t}]\overline{h}\langle\boldsymbol{y}\rangle;(\overline{h}\langle\boldsymbol{y}\rangle \mid \textsc{Chan}(\boldsymbol{y}))\Big)$$

Clients

$$(\!|\,\boldsymbol{0}\,|\!)_\Gamma \stackrel{def}{=} \boldsymbol{0}$$
$$(\!|\,\overline{u}\langle v@A\rangle\,|\!)_\Gamma \stackrel{def}{=} \textsc{link}_\Gamma\,(\boldsymbol{u},\boldsymbol{x})\ \textsc{in}\ \overline{x_{\mathsf{w}@A}}\langle\boldsymbol{v}\rangle$$
$$(\!|\,u(y@A).P\,|\!)_\Gamma \stackrel{def}{=} \textsc{link}_\Gamma\,(\boldsymbol{u},\boldsymbol{x})\ \textsc{in}\ (\nu h:\mathsf{rw}\langle\mathbb{T}_A\rangle)\Big(\overline{x_{\mathsf{r}@A}}\langle h\rangle \mid h(\boldsymbol{y}).(\!|\,P\,|\!)_{\Gamma,y:A}\Big) \quad \text{with } h \text{ fresh}$$
$$(\!|\,P\mid Q\,|\!)_\Gamma \stackrel{def}{=} (\!|\,P\,|\!)_\Gamma \mid (\!|\,Q\,|\!)_\Gamma$$
$$(\!|\,(\nu a:A)P\,|\!)_\Gamma \stackrel{def}{=} (\nu(\boldsymbol{a}):\mathbb{T}_A)(\!|\,P\,|\!)_{\Gamma,a:A}$$
$$(\!|\,[u=v]P;Q\,|\!)_\Gamma \stackrel{def}{=} [u_{\mathsf{r}@\mathsf{r}}=v_{\mathsf{r}@\mathsf{r}}](\!|\,P\,|\!)_\Gamma;(\!|\,Q\,|\!)_\Gamma$$
$$(\!|\,!\,P\,|\!)_\Gamma \stackrel{def}{=} !\,(\!|\,P\,|\!)_\Gamma$$

Complete Systems

$$(\!|\,P\,|\!)^*_\Gamma \stackrel{def}{=} (\!|\,P\,|\!)_\Gamma \mid \textsc{Proxy}$$

The link between client and server names is accomplished as follows:

$$\textsc{link}_\Gamma\,(\boldsymbol{a},\boldsymbol{x})\ \textsc{in}\ P \stackrel{def}{=} (\nu h:\mathsf{rw}\langle\mathbb{T}_A\rangle)(\,\overline{p_A}\langle h,\boldsymbol{a}\rangle \mid h(\boldsymbol{x}).P\,) \quad \text{with } A=\Gamma(a)$$

where P is (intended to be) the encoding for client processes defined as in Table 3. Secondly, the encoding of a restriction may now be defined homomorphically, as the allocation of the channel servers is entirely delegated to the proxy. Finally, note that the encoding is parametric on the typing context Γ since the reply of the proxy has to exactly match the (true) type of the channel name.

The complete system is the composition between the encoding of the aPi@ process and the proxy server. Note that we do not need to introduce channel manager, as done in §4.2. In fact a channel manager will be dynamically allocated by the proxy server as soon as a new proxy channel is created.

As for types, the encoding is unchanged from Table 2, while the encoding of contexts is defined as follows: $(\!|\,\Gamma\,|\!) \stackrel{def}{=} \{\!|\,\Gamma\,|\!\} \cup \{p_A : \mathsf{w}\langle\mathsf{w}\langle\mathbb{T}_A\rangle,\mathbb{T}_A\rangle\}_{A\in\{\mathsf{rw},\mathsf{r},\mathsf{w},\top\}}$ with $\{\!|\,\Gamma\,|\!\}$ as in Table 2. This encoding can be shown type preserving and fully abstract.

Theorem 3 (Typing preservation). *If $\Gamma \vdash P$ in aPi@, then there exists $\Gamma' <: (\!|\,\Gamma\,|\!)$ such that $\Gamma' \vdash (\!|\,\mathcal{I}\,|\!) \rhd \textsc{Proxy} \mid (\!|\,P\,|\!)_\Gamma$ in aPi.*

Theorem 4 (Full Abstraction). *Let Γ and Γ' be compatible with $\mathcal{I}$ such that $\Gamma \vdash P$ and $\Gamma' \vdash Q$. Then $\mathcal{I} \models P \approx_{\mathrm{D}} Q$ if and only if $(\!|\,\mathcal{I}\,|\!) \models \textsc{Proxy} \mid (\!|\,P\,|\!)_\Gamma \approx_{\mathrm{S}} \textsc{Proxy} \mid (\!|\,Q\,|\!)_{\Gamma'}$.*

It is now easy to see how this new encoding solves the inconvenience explained in Example 1 against completeness. SI MOSTRA MEGLIO CON LE BARBE E I CONTESTI

5 Conclusions

We have studied how to express aPi@ in aPi via a fully abstract encoding. Then we may seek an encoding from aPi into aPi@. It is a very simple observation that aPi

can be encoded into aPi@ as shown in [2]. The relevant clauses for the processes are $[\![\, \overline{u}\langle\tilde{v}\rangle \,]\!]_\Gamma \stackrel{def}{=} \overline{u}\langle\tilde{v}@|\Gamma^{\mathsf{w}}(u)|\rangle$ and $[\![\, u(\tilde{x}).P \,]\!]_\Gamma \stackrel{def}{=} u(\tilde{x}@|\Gamma^{\mathsf{r}}(u)|).[\![\, P \,]\!]_{\Gamma,\tilde{x}:|\Gamma^{\mathsf{r}}(u)|}$, where $|\Gamma(a)|$ denotes the outermost capability in $\Gamma(a)$. This encoding, which is (sub)type preserving, has been shown to be sound but not complete. As a matter of fact it can be shown the non-existence of a sound and complete encoding that satisfies an additional condition forcing the encoding of context environments to strongly respect the knowledge of the source.

We may develop a different approach in studying the (non)existence of *reasonable* cross-encodings between the two calculi by adapting existing notions of reasonableness from the literature [4–6, 9, 10, 13] to the type framework. In particular, the notion given in [5, 6], which does not coincide with full abstraction, says that a reasonable encoding (i) should preserve and reflect a basic observable behaviour, namely it should preserve and reflect barbs without introducing new ones, (ii) it should respect the computation of the original process (operational correspondence), (iii) it should not equate terms with an infinite computation and terms with only finite computations, (iv) it should not depend depend on the particular names involved in the original process (name invariance), and finally (v) it should be compositional with respect the parallel operator. Indeed the encoding provided above, from aPi to aPi@, is reasonable according to the previous requirements. On the other hand, the fully abstract encoding of §4 satisfies just name invariance. By following the same line as in [5] it can be proved that there is no reasonable encoding from aPi@ into aPi.

We have studied the relationships between two typed versions of the asynchronous pi-calculus, establishing the following results.

On the negative side, we have shown that no reasonable encoding of aPi into aPi@ can be fully abstract, and that there exists no reasonable encoding of aPi@ into aPi. Based on the analogy we have identified between typed synchronization and guarded input choice, the proof of the latter result can be re-used to substantiate a conjecture from [9] and establish the impossibility of encoding the asynchronous pi-calculus with input-guarded choice into its choice-free fragment, by a non-divergent translation preserving weak bisimulation.

On the positive side, we have given a fully abstract, albeit divergent, encoding of aPi@ into aPi. In its present form, the encoding only applies to the monadic fragment of aPi@, and requires the presence of recursive types in aPi. In fact, the same technique would work for the polyadic calculus, as long as we can count on a finite bound on the maximal arity. In that case, every aPi@ channel may be associated to different tuples of names, one for each possible arity. Simlarly, we could do without recursive types in aPi, as in the original formulation of [7, 8] by assuming a finite bound on the number of cascading re-transmission via other names. In fact, the dynamic types of aPi@ allow any channel to communicate its own name: in the general case, this requires (or at least it appears to require) types with arbitrarily deep nesting, viz, recursive types.

In spite of these limitations, the fully abstract encoding is interesting as it allows us to identify precisely the subclass of the static types of aPi that correspond to the dynamic types of aPi@ and it sheds light on how the dynamically typed synchronization of aPi@ may be simulated by a combination of untyped synchronizations on suitably designed channels.

References

1. M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 104–115. ACM press, 2001.
2. M. Bugliesi and M. Giunti. Typed processes in untyped contexts. In *Proc. of the International Symposium on Trustworthy Global Computing (TGC)*, volume 3705 of *LNCS*, pages 19–32. Springer, 2005.
3. M. Bugliesi and M. Giunti. Secure implementations of typed channel abstractions. In *Proc. of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 251–262. ACM press, 2007.
4. F. S. de Boer and C. Palamidessi. Embedding as a tool for language comparison. *Information and Computation*, 108(1):128–157, 1994.
5. D. Gorla. On the relative expressive power of asynchronous communication primitives. In *Proc. of the International Conference on Foundations of Software Science and Computation Structures (FoSSaCS)*, volume 3921 of *LNCS*, pages 47–62. Springer, 2006.
6. D. Gorla. Synchrony vs asynchrony in communication primitives. In *Proc. of the International Workshop on Expressiveness in Concurrency (EXPRESS)*, ENTCS, pages 61–73. Elsevier, 2006.
7. M. Hennessy. *A Distributed Pi-Calculus*. Cambridge University Press, 2007.
8. M. Hennessy and J. Rathke. Typed behavioural equivalences for processes in the presence of subtyping. *Mathematical Structures in Computer Science*, 14(5):651–684, 2004.
9. U. Nestmann and B C. Pierce. Decoding choice encodings. *Information and Computation*, 163(1):1–59, 2000.
10. C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.
11. B. C. Pierce and D. Sangiorgi. Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science*, 6(5):409–453, 1996.
12. B. C. Pierce and D. Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(3):531–584, 2000.
13. E. Y. Shapiro. Separating concurrent languages with categories of language embeddings (extended abstract). In *Proc. of the ACM Symposium on Theory of Computing (STOC)*, pages 198–208. ACM Press, 1991.

A Typing and transition Rules

In this section we report the subtyping, typing and transition rules for the calculi aPi and aPi@.

Table 5 Subtyping Rules for aPi.

$$\frac{}{\Sigma \vdash T <: T} \qquad \frac{\Sigma \vdash T_1 <: T_1' \ \cdots \ T_n' <: T_n'}{\Sigma \vdash (T_1, \ldots, T_n) <: (T_1', \ldots, T_n')} \qquad \frac{}{\Sigma \vdash T <: \top}$$

$$\frac{\Sigma \vdash \tilde{U}_w <: \tilde{T}_w}{\Sigma \vdash \mathsf{w}\langle \tilde{T}_w \rangle <: \mathsf{w}\langle \tilde{U}_w \rangle} \qquad \frac{\Sigma \vdash \tilde{U}_w <: \tilde{T}_w \quad \tilde{T}_w <: \tilde{T}_r \quad \tilde{T}_r <: \tilde{U}_r}{\Sigma \vdash \mathsf{rw}\langle \tilde{T}_r; \tilde{T}_w \rangle <: \mathsf{rw}\langle \tilde{U}_r; \tilde{U}_w \rangle} \qquad \frac{\Sigma \vdash \tilde{T}_r <: \tilde{U}_r}{\Sigma \vdash \mathsf{r}\langle \tilde{T}_r \rangle <: \mathsf{r}\langle \tilde{U}_r \rangle}$$

$$\frac{\Sigma \vdash \tilde{T}_w <: \tilde{T}_r}{\Sigma \vdash \mathsf{rw}\langle \tilde{T}_r; \tilde{T}_w \rangle <: \mathsf{w}\langle \tilde{T}_w \rangle} \qquad \frac{\Sigma \vdash \tilde{T}_w <: \tilde{T}_r}{\Sigma \vdash \mathsf{rw}\langle \tilde{T}_r; \tilde{T}_w \rangle <: \mathsf{r}\langle \tilde{T}_r \rangle}$$

$$\frac{\Sigma, \mu X.T_1 <: T_2 \vdash T_1\{\mu X.T_1/X\} <: T_2}{\Sigma \vdash T_1 <: T_2} \qquad \frac{\Sigma, T_1 <: \mu X.T_2 \vdash T_1 <: T_2\{\mu X.T_2/X\}}{\Sigma \vdash T_1 <: T_2}$$

Judgments are enriched by a subtyping environment of the form $\Sigma = \{S <: S', \ldots\}$.
We say that $S <: T$ if $\emptyset \vdash S <: T$

Table 6 Typing rules for aPi.

Environments

(st-empty) $\dfrac{}{\vdash \diamond}$ (st-ext) $\dfrac{\Gamma \vdash \diamond \quad \Gamma \downarrow u : T}{\Gamma, u : T \vdash \diamond}$

Capabilities

(st-sub) $\dfrac{\Gamma, u : T, \Gamma' \vdash \diamond \quad T <: T'}{\Gamma, u : T, \Gamma' \vdash u : T'}$ (st-meet) $\dfrac{\Gamma \vdash u : T \quad \Gamma \vdash u : T'}{\Gamma \vdash u : (T \sqcap T')}$ (st-tup) $\dfrac{\Gamma \vdash u_i : T_i \quad (i = 1, \ldots, n)}{\Gamma \vdash (u_1, \ldots, u_n) : (T_1, \ldots, T_n)}$

Procesees

(st-stop) $\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0}}$ (st-par) $\dfrac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$ (st-in) $\dfrac{\Gamma \vdash u : \mathsf{r}\langle \tilde{T} \rangle \quad \Gamma, \langle \tilde{x} : \tilde{T} \rangle \vdash P}{\Gamma \vdash u(\tilde{x}).P}$

(st-repl) $\dfrac{\Gamma \vdash P}{\Gamma \vdash !P}$ (st-new) $\dfrac{\Gamma, a : T \vdash P}{\Gamma \vdash (\nu a : T)P}$ (st-out) $\dfrac{\Gamma \vdash u : \mathsf{w}\langle \tilde{T} \rangle \quad \Gamma \vdash \tilde{v} : \tilde{T}}{\Gamma \vdash \overline{u}\langle \tilde{v} \rangle}$

(st-match)

$$\frac{\Gamma \vdash u_1 : T_1,\ u_2 : T_2 \quad \Gamma \vdash Q \quad \Gamma, \langle u_1 : T_1 \rangle, \langle u_2 : T_2 \rangle \vdash \diamond \quad \Gamma, \langle u_1 : T_1 \rangle, \langle u_2 : T_2 \rangle \vdash P}{\Gamma \vdash [u_1 = u_2]P; Q}$$

Table 7 Operational Semantics for aPi.

(ST-IN)
$$\frac{}{a(\tilde{x}).P \xrightarrow{a(\tilde{v})} P\{\tilde{v}/\tilde{x}\}}$$

(ST-OUT)
$$\frac{}{\overline{a}\langle\tilde{v}\rangle \xrightarrow{\overline{a}\langle\tilde{v}\rangle} \mathbf{0}}$$

(ST-OPEN)
$$\frac{P \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}\rangle} P' \quad b \in \{\tilde{v}\} \setminus \{a, \tilde{c}\}}{(\nu b:B)P \xrightarrow{(b:B,\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}\rangle} P'}$$

(ST-CTXT)
$$\frac{P \xrightarrow{\mu} P' \quad bn(\mu) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\mu} P' \mid Q}$$

(ST-COMM)
$$\frac{P \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}\rangle} P' \quad Q \xrightarrow{a(\tilde{v})} Q' \quad \tilde{c} \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\tilde{c}:\tilde{C})(P' \mid Q')}$$

(ST-COMM)
$$\frac{P \xrightarrow{a(\tilde{v})} P' \quad Q \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}\rangle} Q' \quad \tilde{c} \cap fn(P) = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\tilde{c}:\tilde{C})(P' \mid Q')}$$

(ST-RES)
$$\frac{P \xrightarrow{\mu} P' \quad b \notin n(\mu)}{(\nu b:T)P \xrightarrow{\mu} (\nu b:T)P'}$$

(ST-REPL)
$$\frac{P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P' \mid !P}$$

(ST-EQ)
$$\frac{P \xrightarrow{\mu} P'}{[u=u]P; Q \xrightarrow{\mu} P'}$$

(ST-NEQ)
$$\frac{Q \xrightarrow{\mu} Q' \quad u \neq v}{[u=v]P; Q \xrightarrow{\mu} Q'}$$

Table 8 Typing rules for aPi@.

Environments

(DYN-EMPTY)
$$\frac{}{\vdash \diamond}$$

(DYN-EXT)
$$\frac{\Gamma \vdash \diamond \quad \Gamma \downarrow u : A}{\Gamma, u : A \vdash \diamond}$$

Capabilities

(DYN-SUB)
$$\frac{\Gamma, u : A, \Gamma' \vdash \diamond \quad A <: A'}{\Gamma, u : A, \Gamma' \vdash u : A'}$$

(DYN-MEET)
$$\frac{\Gamma \vdash u : A \quad \Gamma \vdash u : A'}{\Gamma \vdash u : (A \sqcap A')}$$

(DYN-TUP)
$$\frac{\Gamma \vdash u_i : A_i \quad (i = 1, \ldots, n)}{\Gamma \vdash (u_1, \ldots, u_n) : (A_1, \ldots, A_n)}$$

Procesees

(DYN-STOP)
$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0}}$$

(DYN-PAR)
$$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \mid Q}$$

(DYN-IN)
$$\frac{\Gamma \vdash u : \mathsf{r} \quad \Gamma, \langle\tilde{x} : \tilde{A}\rangle \vdash P}{\Gamma \vdash u(\tilde{x}@\tilde{A}).P}$$

(DYN-REPL)
$$\frac{\Gamma \vdash P}{\Gamma \vdash !P}$$

(DYN-NEW)
$$\frac{\Gamma, a : A \vdash P}{\Gamma \vdash (\nu a:A)P}$$

(DYN-OUT)
$$\frac{\Gamma \vdash u : \mathsf{w} \quad \Gamma \vdash \tilde{v} : \tilde{B}}{\Gamma \vdash \overline{u}\langle\tilde{v}@\tilde{B}\rangle}$$

(DYN-MATCH)
$$\frac{\Gamma \vdash u_1 : A_1,\ u_2 : A_2 \quad \Gamma \vdash Q \quad \Gamma, \langle u_1 : A_1\rangle, \langle u_2 : A_2\rangle \vdash \diamond \quad \Gamma, \langle u_1 : A_1\rangle, \langle u_2 : A_2\rangle \vdash P}{\Gamma \vdash [u_1 = u_2]P; Q}$$

Table 9 Subtyping Rules for aPi@.

$$\frac{}{A <: A} \qquad \frac{}{\mathsf{rw} <: \mathsf{r}} \qquad \frac{}{\mathsf{rw} <: \mathsf{w}} \qquad \frac{}{A <: \top} \qquad \frac{A_1 <: A'_1 \ \cdots \ A'_n <: A'_n}{(A_1, \ldots, A_n) <: (A'_1, \ldots, A'_n)}$$

Table 10 Operational Semantics for aPi@.

(DYN-IN)
$$\frac{}{a(\tilde{x}@\tilde{A}).P \xrightarrow{a(\tilde{v}@\tilde{A})} P\{\tilde{v}/\tilde{x}\}}$$

(DYN-OUT)
$$\frac{}{\overline{a}\langle\tilde{v}@\tilde{A}\rangle \xrightarrow{\overline{a}\langle\tilde{v}@\tilde{A}\rangle} \mathbf{0}}$$

(DYN-OPEN)
$$\frac{P \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{A}@\tilde{v}\rangle} P' \quad b \in \{\tilde{v}\} \setminus \{a, \tilde{c}\}}{(\nu b:B)P \xrightarrow{(b:B.\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}@\tilde{A}\rangle} P'}$$

(ST-CTXT)
$$\frac{P \xrightarrow{\alpha} P' \quad bn(\alpha) \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

(ST-COMM)
$$\frac{P \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}@\tilde{A}\rangle} P' \quad Q \xrightarrow{a(\tilde{v}@\tilde{B})} Q' \quad \tilde{A} <: \tilde{B} \quad \tilde{c} \cap fn(Q) = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\tilde{c}:\tilde{C})(P' \mid Q')}$$

(ST-COMM)
$$\frac{P \xrightarrow{a(\tilde{v}@\tilde{A})} P' \quad Q \xrightarrow{(\tilde{c}:\tilde{C})\overline{a}\langle\tilde{v}@\tilde{B}\rangle} Q' \quad \tilde{B} <: \tilde{A} \quad \tilde{c} \cap fn(P) = \emptyset}{P \mid Q \xrightarrow{\tau} (\nu\tilde{c}:\tilde{C})(P' \mid Q')}$$

(ST-RES)
$$\frac{P \xrightarrow{\alpha} P' \quad b \notin n(\alpha)}{(\nu b:T)P \xrightarrow{\alpha} (\nu b:T)P'}$$

(ST-REPL)
$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

(ST-EQ)
$$\frac{P \xrightarrow{\alpha} P'}{[u=u]P; Q \xrightarrow{\alpha} P'}$$

(ST-NEQ)
$$\frac{Q \xrightarrow{\alpha} Q' \quad u \neq v}{[u=v]P; Q \xrightarrow{\alpha} Q'}$$

B Encoding aPi@ into aPi

The aim of this section is to prove Theorem 1. First we prove some auxiliary results.

Lemma 1 (Subtyping Preservation). *The subtyping lattice of* aPi@ *is preserved by the recursive types of Table 2, that is:* $\mathbb{T}_{rw} <: \{\mathbb{T}_r, \mathbb{T}_w\} <: \mathbb{T}_\top$.

Corollary 1. *If* $\Gamma <: \Gamma'$ *in* aPi@, *then* $\{\!|\Gamma|\!\} <: \{\!|\Gamma'|\!\}$ *in* aPi.

We define an auxiliary encoding for type environments, based on the new class of recursive types depicted Table 11. The tuples $\mathbb{R}$ and $\mathbb{W}$ are the same as in §4, and are defined by using the types of Table 2.

Table 11 Auxiliary recursive types.

$$\begin{array}{ll} \boldsymbol{S}_{\mathsf{r@rw}} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle\rangle & \boldsymbol{S}_{\mathsf{w@rw}} \stackrel{def}{=} \mathsf{rw}\langle \mathbb{R}, \mathbb{W}\rangle \\ \boldsymbol{S}_{\mathsf{r@r}} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{w}\langle \mathbb{R}, \mathsf{T}\rangle\rangle & \boldsymbol{S}_{\mathsf{w@r}} \stackrel{def}{=} \mathsf{rw}\langle \mathbb{R}, \mathsf{T}\rangle \\ \boldsymbol{S}_{\mathsf{r@w}} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle\rangle & \boldsymbol{S}_{\mathsf{w@w}} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{T}, \mathbb{W}\rangle \\ \boldsymbol{S}_{\mathsf{r@}\top} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{w}\langle \mathsf{T}, \mathsf{T}\rangle\rangle & \boldsymbol{S}_{\mathsf{w@}\top} \stackrel{def}{=} \mathsf{rw}\langle \mathsf{T}, \mathsf{T}\rangle \end{array}$$

The definition implies that $\boldsymbol{S}_{\mathsf{r}@A} <: \boldsymbol{T}_{\mathsf{r}@A}$ and $\boldsymbol{S}_{\mathsf{w}@A} <: \boldsymbol{T}_{\mathsf{w}@A}$ for every type A in API@. The auxiliary encoding $\{\!|\ |\!\}_{\mathbb{S}}$ is based on the type $\mathbb{S} \stackrel{def}{=} (\mathbb{R}_{\mathbb{S}}, \mathbb{W}_{\mathbb{S}})$, where $\mathbb{R}_{\mathbb{S}} \stackrel{def}{=} (\boldsymbol{S}_{\mathsf{r@rw}}, \boldsymbol{S}_{\mathsf{r@r}}, \boldsymbol{S}_{\mathsf{r@w}}, \boldsymbol{S}_{\mathsf{r@}\top})$ and $\mathbb{W}_{\mathbb{S}} \stackrel{def}{=} (\boldsymbol{S}_{\mathsf{w@rw}}, \boldsymbol{S}_{\mathsf{w@r}}, \boldsymbol{S}_{\mathsf{w@w}}, \boldsymbol{S}_{\mathsf{w@}\top})$. Clearly $\mathbb{R}_{\mathbb{S}} <: \mathbb{R}$ and $\mathbb{W}_{\mathbb{S}} <: \mathbb{W}$. Then $\mathbb{S} <: \mathbb{T}_{\mathsf{rw}}$ and thus Lemma 1 says that

$$\mathbb{S} <: T \text{ for every } T \in \{\mathbb{T}_{\mathsf{rw}}, \mathbb{T}_{\mathsf{r}}, \mathbb{T}_{\mathsf{w}}, \mathbb{T}_{\top}\}. \tag{1}$$

The encoding $\{\!|\ |\!\}_{\mathbb{S}}$ is defined as

$$\begin{array}{rl} \{\!|\emptyset|\!\}_{\mathbb{S}} & \stackrel{def}{=} \mathsf{t} : \top, \mathsf{f} : \top \\ \{\!|\Gamma, v : A|\!\}_{\mathbb{S}} & \stackrel{def}{=} \{\!|\Gamma|\!\}, (\boldsymbol{v}) : \mathbb{S} \end{array}$$

The observation (1) implies the following fact.

Fact 5 *If Γ is a typing environment in* API@, *then* $\{\!|\Gamma|\!\}_{\mathbb{S}} <: \{\!|\Gamma|\!\}$.

Moreover $\{\!|\Gamma|\!\}_{\mathbb{S}}$ is a good typing environment to type-check the channel manager associated to a type environment Γ, as will be clear from the following lemma.

Lemma 2. *Given the typing environment Γ in* API@, *then* $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Chan}(a)$ *in* API *for every $a \in dom(\Gamma)$.*

Proof. Let $a \in dom(\Gamma)$ and consider $\textsc{Chan}(a) \stackrel{def}{=} \Pi_A\ !\,u_{\mathsf{r}@A}(h).\textsc{Choose}(u, A, h)$. In order to prove that $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Chan}(a)$, we show that $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash u_{\mathsf{r}@A}(h).\textsc{Choose}(u, A, h)$ for every type A in API@. The proof follows a common pattern for every $A \in \{\mathsf{rw}, \mathsf{r}, \mathsf{w}, \top\}$.

For instance, let $A = \mathsf{w}$ and check that $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash u_{\mathsf{r}@\mathsf{w}}(h).\textsc{Choose}(u, \mathsf{w}, h)$. Essentially this amounts to derive

1. $\{\!|\Gamma|\!\}, h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle, l : \mathsf{rw}\langle\top\rangle, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Read}_l \left\langle u_{\mathsf{w}@\mathsf{w}}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle \right\rangle$,
2. $\{\!|\Gamma|\!\}, h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle, l : \mathsf{rw}\langle\top\rangle, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Read}_l \left\langle u_{\mathsf{w}@\mathsf{rw}}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle \right\rangle$.

Consider (i). Let $\Delta_1 = \{\!|\Gamma|\!\}_{\mathbb{S}}, h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle, l : \mathsf{rw}\langle\top\rangle, (z_{\mathbb{R}}, z_{\mathbb{W}}) : (\mathsf{T}, \mathbb{W}), \langle(\boldsymbol{a}) : \mathbb{S}\rangle$. Since $\Delta_1 \vdash u_{\mathsf{w}@\mathsf{w}} : \mathsf{rw}\langle \mathsf{T}, \mathbb{W}\rangle <: \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle$ and $\Delta_1 \vdash h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle$, the rules of Table 6 derive $\Delta_1 \vdash \textsc{Test}_l \left\langle u_{\mathsf{w}@\mathsf{w}}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle \right\rangle$. Moreover $\Delta_1 \vdash u_{\mathsf{w}@\mathsf{w}} : \mathsf{rw}\langle \mathsf{T}, \mathbb{W}\rangle <: \mathsf{r}\langle \mathsf{T}, \mathbb{W}\rangle$, hence (i) follows by (ST-INP). The point *(ii)* is analogous. Let $\Delta_2 = \{\!|\Gamma|\!\}_{\mathbb{S}}, h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle, l : \mathsf{rw}\langle\top\rangle, (z_{\mathbb{R}}, z_{\mathbb{W}}) : (\mathbb{R}, \mathbb{W}), \langle(\boldsymbol{a}) : \mathbb{S}\rangle$. Since $\Delta_2 \vdash u_{\mathsf{w}@\mathsf{rw}} : \mathsf{rw}\langle \mathbb{R}, \mathbb{W}\rangle <: \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle$ and $\Delta_2 \vdash h : \mathsf{w}\langle \mathsf{T}, \mathbb{W}\rangle <: \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle$, the rules of Table 6 derive $\Delta_2 \vdash \textsc{Test}_l \left\langle u_{\mathsf{w}@\mathsf{rw}}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle \right\rangle$. Moreover $\Delta_2 \vdash u_{\mathsf{w}@\mathsf{rw}} : \mathsf{rw}\langle \mathbb{R}, \mathbb{W}\rangle <: \mathsf{r}\langle \mathbb{R}, \mathbb{W}\rangle$, hence (ii) follows by (ST-INP). Now, points (i) and (ii) imply $\{\!|\Gamma|\!\}, h : \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Choose}(u, \mathsf{w}, h)$ by (ST-REPL), (ST-PAR) and (ST-NEW). Then, since $\{\!|\Gamma|\!\}, h : \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash u_{\mathsf{r}@\mathsf{w}} : \mathsf{rw}\langle \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle\rangle <: \mathsf{r}\langle \mathsf{w}\langle \mathbb{R}, \mathbb{W}\rangle\rangle$, an application of the rule (ST-INP) concludes that $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash u_{\mathsf{r}@\mathsf{w}}(h).\textsc{Choose}(u, \mathsf{w}, h)$.

As already said, the typing judgement $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash u_{\mathsf{r}@A}(h).\textsc{Choose}(u, A, h)$ can be derived in a similar way also when A is rw, r, or $\top$. Thus $\{\!|\Gamma|\!\}, \langle(\boldsymbol{a}) : \mathbb{S}\rangle \vdash \textsc{Chan}(a)$ follows by (ST-REPL) and (ST-PAR).

The following corollary is a forthright consequence of this lemma and of the fact that $dom(\Gamma) \subseteq dom(\mathcal{I})$ whenever $\Gamma <: \mathcal{I}$.

Corollary 2. *For every Γ in* aPi@ *it holds $\{\!|\Gamma|\!\}_{\mathbb{S}} \vdash \textsc{Manager}_{\Gamma}$ in* aPi.

Another useful result for the type preservation of the encoding is the following lemma.

Lemma 3. *If $\Gamma \vdash P$ in* aPi@, *then $\{\!|\Gamma|\!\} \vdash \{\!|P|\!\}$ in* aPi.

Proof. First check that $\Gamma \vdash \diamond$ implies $\{\!|\Gamma|\!\} \vdash \diamond$. Then, by means of Lemma 1, show that $\Gamma \vdash v : A$ implies $\{\!|\Gamma|\!\} \vdash (\boldsymbol{v}) : \mathbb{T}_A$. Finally prove the lemma by induction on the derivation of $\Gamma \vdash P$ in aPi@. The interesting cases are the inductive steps for (DYN-INP), (DYN-OUT) and (DYN-NEW).

When (DYN-IN) is the last rule of the type derivation then

$$\frac{\Gamma \vdash u : \mathsf{r} \quad \Gamma, \langle x : A\rangle \vdash P}{\Gamma \vdash u(x@A).P}$$

Assume for instance that $A = \mathsf{w}$. The induction hypotheses say that (i) $\{\!|\Gamma|\!\} \vdash (\boldsymbol{u}) : \mathbb{T}_{\mathsf{r}}$, hence $\{\!|\Gamma|\!\} \vdash u_{\mathsf{r}@\mathsf{w}} : \mathsf{w}\langle\mathsf{w}\langle\mathbb{T}_{\mathsf{w}}\rangle\rangle$; and (ii) $\{\!|\Gamma|\!\}, \langle\{\!| x : \mathsf{w} |\!\}\rangle \vdash \{\!|P|\!\}$, namely $\{\!|\Gamma|\!\}, \langle(\boldsymbol{x}) : \mathbb{T}_{\mathsf{w}}\rangle \vdash \{\!|P|\!\}$. Now the typing rules for aPi (in Tab. 6), (ST-IN) and (ST-IN) in particular, allow the derivation of the typing judgment $\{\!|\Gamma|\!\} \vdash (\nu h : \mathsf{rw}\langle\mathbb{T}_{\mathsf{w}}\rangle)\,(\,\overline{u_{\mathsf{r}@\mathsf{w}}}\langle h\rangle \,|\, h(\boldsymbol{x}).\{\!|P|\!\}\,)$, that is $\{\!|\Gamma|\!\} \vdash \{\!|P|\!\}$. Cases $A = \mathsf{rw}, \mathsf{r}, \top$ are similar.

When (DYN-OUT) is the last rule of the type derivation

$$\frac{\Gamma \vdash u : \mathsf{w} \quad \Gamma \vdash v : A}{\Gamma \vdash \overline{u}\langle v@A\rangle}$$

then (i) $\{\!|\Gamma|\!\} \vdash (\boldsymbol{u}) : \mathbb{T}_A$, hence $\{\!|\Gamma|\!\} \vdash u_{\mathsf{w}@A} : \mathsf{w}\langle\mathbb{T}_A\rangle$; and (ii) $\{\!|\Gamma|\!\} \vdash (v_{\mathbb{R}}, v_{\mathbb{W}}) : \mathbb{T}_A$. Hence $\{\!|\Gamma|\!\} \vdash \overline{u_{\mathsf{w}@A}}\langle\boldsymbol{v}\rangle$, by (DYN-OUT), and so $\{\!|\Gamma|\!\} \vdash \{\!|\overline{u}\langle v@A\rangle|\!\}$.

When (DYN-NEW) is the last rule of the type derivation then

$$\frac{\Gamma, a : A \vdash P}{\Gamma \vdash (\nu a{:}A)P}$$

The induction hypothesis says that $\{\!|\Gamma, a : A|\!\} \vdash \{\!|P|\!\}$ in aPi, that is $\{\!|\Gamma|\!\}, (\boldsymbol{a}) : \boldsymbol{T}_A \vdash \{\!|P|\!\}$, hence $\{\!|\Gamma|\!\}, (\boldsymbol{a}) : \mathbb{S} \vdash \{\!|P|\!\}$. Moreover $\{\!|\Gamma|\!\}, (\boldsymbol{a}) : \mathbb{S} \vdash \textsc{Chan}(a)$ by Lemma 2, then $\{\!|\Gamma|\!\}, (\boldsymbol{a}) : \mathbb{S} \vdash \{\!|P|\!\} \,|\, \textsc{Chan}(a)$ by (ST-PAR). Finally $\{\!|\Gamma|\!\} \vdash \{\!|(\nu a{:}A)P|\!\}$ by (ST-NEW).

Proof. *Proof of Theorem 1. Let $\Gamma \vdash P$ in aPi@. Lemma 3 says that $\{\!|\Gamma|\!\} \vdash \{\!|P|\!\}$ in aPi, hence $\{\!|\Gamma|\!\}_{\mathbb{S}} \vdash \{\!|P|\!\}$ by Fact 5. Moreover $\{\!|\Gamma|\!\}_{\mathbb{S}} \vdash \textsc{Manager}_{\Gamma}$ by Corollary 2. Thus $\{\!|\Gamma|\!\}_{\mathbb{S}} \vdash \{\!|P|\!\}_{\Gamma}$ by (ST-PAR). The thesis of the theorem is obtained with $\Gamma' = \{\!|\Gamma|\!\}_{\mathbb{S}}$.

Then we can prove that the encoding actually maps configurations into configurations.

Corollary 3. *If $\mathcal{I} \triangleright P$ is a configuration in* AΠ@*, with $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$, then $\{\!|\mathcal{I}|\!\} \triangleright \{\!|P|\!\}_\Gamma$ is a configuration in* AΠ.

Proof. Let $\mathcal{I} \triangleright P$ be a configuration, with $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$ in AΠ@. Theorem 1 says that there exists $\Gamma' <: \{\!|\Gamma|\!\}$ such that $\Gamma' \vdash \{\!|P|\!\}_\Gamma$ in AΠ. Lemma 1 says that $\{\!|\Gamma|\!\} <: \{\!|\mathcal{I}|\!\}$, hence $\Gamma' <: \{\!|\mathcal{I}|\!\}$. Thus $\{\!|\mathcal{I}|\!\} \triangleright \{\!|P|\!\}_\Gamma$ is a configuration in AΠ.

Proof of Proposition 2 (sketch). Show (i) by induction on the derivation $P \xrightarrow{\mu} P'$ in AΠ@. Point (ii) is proved in two steps. If $Q \xrightarrow{\Lambda} Q'$ with an administrative transition, then $Q' \approx_A Q$, as $\approx_A$ is closed under administrative reductions. Conclude by choosing $\mu = \tau$. If $Q \xrightarrow{\lambda} Q'$ with a effective transition, then by definition there are R and R' such that $\{\!|P|\!\}_\Gamma \xrightarrow{A_\Lambda} R \xrightarrow{\lambda} \xrightarrow{A_\Lambda} R'$ and $Q' \approx_A R'$. The proof shows that there is a 'canonical' sequence of administrative reductions, noted as $\lceil\Lambda\rceil$, only including the administrative steps from $\{\!|P|\!\}_\Gamma$ required to enable the action in R, and such that $\{\!|P|\!\}_\Gamma \stackrel{\lceil\Lambda\rceil}{\Longrightarrow} \xrightarrow{\lambda} \approx_A R'$. This last property implies that there are a process P' and an action μ in AΠ@ such that $P \xrightarrow{\mu} P'$ and $\{\!|P|\!\}_\Gamma \approx_A R'$. This is done by case analysis on $\xrightarrow{\lambda}$ and by observing that the administrative actions arise either when there is an available output in P or when there is an input ready to synchronise or both (the last case produces a τ) action in AΠ@. Finally, transitivity says that $\{\!|P'|\!\}_\Gamma \approx_A R'$.

C Proof of Soundness

Let the functions ρ, σ range over renaming functions, with $P\rho$ we mean the name substitution applied to the names of the process P

Definition 5 (Derivative process). *We say that an* AΠ *process H is a $(\!|\ |\!)_\Gamma$-derivative if there exist an* AΠ *process K, an* AΠ@ *process P such that $\Gamma \vdash P$ and two renaming functions ρ and σ such that $H = K\sigma$ and $(\!|\ P\ |\!)^*_\Gamma \Longrightarrow K\rho$.*

In the encoding, we resort to a two sorted set of names: 'standard' noted by $m, n \ldots$ and 'signed' noted by $\mathfrak{h}, \mathfrak{m}, \ldots$ In the encoding, signed names are those sent on the channels $u_{\mathfrak{r}@A}$, which are used for the private communication between a process willing for an input on (the encoding of) channel u and the channel manager. In formally, the encoding of an input becomes:

$$(\!|\ u(y@A).P\ |\!)_\Gamma \stackrel{def}{=} \textsc{link}_\Gamma\ (u, x)\ \textsc{in}\ (\nu\mathfrak{h}\!:\!\mathsf{rw}\langle\mathbb{T}_A\rangle)\left(\overline{x_{\mathfrak{r}@A}}\langle\mathfrak{h}\rangle \mid \mathfrak{h}(y).(\!|\ P\ |\!)_{\Gamma,y:A}\right)$$

We call *administrative* every internal communication of the encodings that is not a synchronisation along a signed channel.

Definition 6 (Administrative Reduction). *We say that $P \xrightarrow{\tau} P'$ is an $\mathcal{I}$-administrative reduction, noted $P \xrightarrow{A_\mathcal{I}} P'$, when $\mathcal{I} \triangleright P\downarrow_a$ iff $\mathcal{I} \triangleright P'\downarrow_a$ and $P = C[H]$, $P' = C[H']$ with H, H' $(\!|\ |\!)_\Gamma$-derivatives such that $H \xrightarrow{\tau} H'$ is synchronisation on a standard channel.*

We define $\overset{A_I}{\Longrightarrow}$ to be the reflexive and transitive closure of $\overset{A_I}{\longrightarrow}$. Moreover we mark as $\overset{\tau_I}{\longrightarrow}$ a τ-transition which is not $\mathcal{I}$-administrative.

Definition 7 (Administrative Equivalence). *The administrative equivalence $\approx_A$ is the largest symmetric, contextual and term indexed relation $\mathcal{R}$ such that $\mathcal{I} \models P \mathcal{R} Q$ implies*

1. *if $\mathcal{I} \rhd P\downarrow_a$, then $\mathcal{I} \rhd P'\downarrow_a$*
2. *if $P \overset{A_I}{\longrightarrow} P'$, then $Q \overset{A_I}{\Longrightarrow} Q'$ for some Q' such that $\mathcal{I} \models P' \mathcal{R} Q'$*
3. *if $P \overset{\tau_I}{\longrightarrow} P'$, then $Q \overset{A_I}{\Longrightarrow}\overset{\tau_I}{\longrightarrow}\overset{A_I}{\Longrightarrow} Q'$ for some Q' such that $\mathcal{I} \models P' \mathcal{R} Q'$*

It is easy to see that the above definition implies the following property, useful to prove that the administrative equivalence is indeed an equivalence relation.

Lemma 4. *Given $\mathcal{I} \models P \approx_A Q$ then:*

- *if $P \overset{A_I}{\Longrightarrow} P'$ then $Q \overset{A_I}{\Longrightarrow} Q'$ for some Q' such that $\mathcal{I} \models P' \approx_A Q'$*
- *if $P \overset{A_I}{\Longrightarrow}\overset{\tau_I}{\longrightarrow}\overset{A_I}{\Longrightarrow} P'$ then $Q \overset{A_I}{\Longrightarrow}\overset{\tau_I}{\longrightarrow}\overset{A_I}{\Longrightarrow} Q'$ for some Q' such that $\mathcal{I} \models P' \approx_A Q'$*

Lemma 5. *The relation $\approx_A$ is an equivalence over* API *processes.*

Proof. Reflexivity can be checked by showing that the identity relation *Id* satisfies the properties required by Definition 7. Symmetry holds by definition. Finally, Lemma 5 is useful to show that $\approx_A \approx_A$ satisfies Definition 7 and thus to prove the transitivity of $\approx_A$.

An important observation is that $\approx_A$ is finer than $\approx_s$.

Lemma 6. *If $\mathcal{I} \models P \approx_A Q$ then $\mathcal{I} \models P \approx_s Q$.*

Proof. Show that the relation $\approx_A$ satisfies the properties of Definition 4. The contextuality of $\approx_A$ is a consequence of Definition 7. To see that $\approx_A$ preserves barbs, assume that $\mathcal{I} \models P \approx_A Q$ and $\mathcal{I} \rhd P\downarrow_a$, then $\mathcal{I} \rhd Q\downarrow_a$ by Definition 7, and thus $\mathcal{I} \rhd Q\Downarrow_a$. To see that $\approx_A$ is reduction closed, assume that $\mathcal{I} \models P \approx_A Q$ and $P \overset{\tau}{\longrightarrow} P'$. In case $P \overset{A_I}{\longrightarrow} P'$, then there exists Q' such that $\mathcal{I} \models P' \approx_A Q'$ and $Q \overset{A_I}{\Longrightarrow} Q'$, hence $Q \Longrightarrow Q'$. In case $P \overset{\tau_I}{\longrightarrow} P'$, then there exists Q' such that $\mathcal{I} \models P' \approx_A Q'$ and $Q \overset{A_I}{\Longrightarrow}\overset{\tau_I}{\longrightarrow}\overset{A_I}{\Longrightarrow} Q'$, hence $Q \Longrightarrow Q'$.

The usual congruence of pi-calculus, denoted by $\equiv$, will simplify the proofs in the following of the paper. Basically, it identifies processes with the same internal structure, and it is defined as the smallest congruence that is closed under α-conversion and that satisfies the following axioms:

$$P|Q \equiv Q|P \qquad P|(\nu a:T)Q \equiv (\nu a:T)(P|Q) \qquad !(P|Q) \equiv !P|!Q$$
$$P|\mathbf{0} \equiv P \qquad (\nu a_1:T_1)(\nu a_2:T_2)P \equiv (\nu a_2:T_2)(\nu a_1:T_1)P \qquad !P \equiv !P|P$$
$$[a=a]P;Q \equiv P \qquad [a=b]P;Q \equiv P \quad (\text{if } a \neq b) \qquad !!P \equiv !P$$
$$(\nu a:T)\mathbf{0} \equiv \mathbf{0} \qquad (\nu a:T)\overline{a}\langle\tilde{b}\rangle.P \equiv \mathbf{0} \qquad (\nu a:T)a(\tilde{x}).P \equiv \mathbf{0}$$

It is easy to see that congruence is sound with respect the operational semantics given in Table 7. In fact, if $P \equiv Q$ and $P \overset{\alpha}{\longrightarrow} P'$ then there exists Q' such that $Q \overset{\alpha}{\longrightarrow} Q'$ and $P' \equiv Q'$. As expected, congruence is finer that $\approx_A$, as stated by the following lemma, whose proof is fairly standard.

Lemma 7. *If $P \equiv Q$ then $\mathcal{I} \models P \approx_A Q$ for every $\mathcal{I}$.*

It is straightforward to see how the congruence relation provides a characterisation of administrative reductions. The details are outlined in Table 12 at the end of the Appendix.

Lemma 8 (Administrative Reduction – Characterisation). *The reduction $P \xrightarrow{\tau} P'$ is $\mathcal{I}$-administrative when $\mathcal{I} \rhd P\downarrow_a$ iff $\mathcal{I} \rhd P'\downarrow_a$ and $P = C[H]$, $P' = C[H']$ with H, H' satisfying one of the eight cases depicted in Table 12.*

C.1 Closure Properties of Administrative Equivalence

The key property of administrative reductions is that they are closed under administrative equivalence, as stated by Lemma 10. The notion of non-overlapping transitions and the property stated in Lemma 9 will be useful to prove Lemma 10. We say that the reductions $P \xrightarrow{\tau} Q$ and $P \xrightarrow{\tau} R$ are *non-overlapping* if there are $C[]$, P' and P'' such that $P \equiv C[P' \mid P'']$, $Q \equiv C[Q' \mid P'']$ with $P' \xrightarrow{\tau} Q'$, and $R \equiv C[P' \mid R'']$ with $P'' \xrightarrow{\tau} R''$.

Lemma 9. *If $P \xrightarrow{\tau} Q$ and $P \xrightarrow{\tau} R$ are two non-overlapping transitions, then there exists a process H such that $Q \xrightarrow{\tau} H$ and $R \xrightarrow{\tau} H$.*

Proof. The definition says that there are $C[]$, P' and P'' such that $P \equiv C[P' \mid P'']$, $Q \equiv C[Q' \mid P'']$ with $P' \xrightarrow{\tau} Q'$, and $R \equiv C[P' \mid R'']$ with $P'' \xrightarrow{\tau} R''$. Then define H to be $C[Q' \mid R'']$ and check that $Q \xrightarrow{\tau} H$ and $R \xrightarrow{\tau} H$.

Lemma 10. *If $P \xrightarrow{A_I} P'$ then $\mathcal{I} \models P \approx_A P'$.*

Proof. We consider the type indexed relation $\mathcal{R} \stackrel{def}{=} (\mathcal{R}_1 \cup \mathcal{R}_2 \cup \equiv)$, where $\mathcal{I} \models C[P] \mathcal{R}_1 C[Q]$ and $\mathcal{I} \models C[Q] \mathcal{R}_1 C[P]$ whenever $P \xrightarrow{A_I} Q$. Then we show that $\mathcal{R}$ is symmetric, barb preserving, reduction closed and contextual.

Symmetry. It holds by definition

Barb Preservation. Assume that $\mathcal{I} \models C[P]\ \mathcal{R}_1\ C[Q]$. If $\mathcal{I} \rhd C[]\downarrow_a$ then $\mathcal{I} \rhd C[P]\downarrow_a$ if and only if $\mathcal{I} \rhd C[Q]\downarrow_a$. Otherwise $\mathcal{I} \rhd P\downarrow_a$ if and only if $\mathcal{I} \rhd Q\downarrow_a$, and again $\mathcal{I} \rhd C[P]\downarrow_a$ if and only if $\mathcal{I} \rhd C[Q]\downarrow_a$. The case for $\mathcal{R}_2$ is analogous and the case for $\equiv$ follows from Lemma 7.

Reduction Closure. The relation $\mathcal{R}_1$ is the only significative one. Then we assume that $\mathcal{I} \models C[P]\ \mathcal{R}_1\ C[Q]$, meaning that $P \xrightarrow{A_I} Q$. We need to prove that *(a)* if $C[P] \xrightarrow{A_I} H$ then $C[Q] \stackrel{A_I}{\Longrightarrow} K$ with $H\ \mathcal{R}\ K$, and that *(b)* if $C[P] \xrightarrow{\tau_I} H$ then $C[Q] \stackrel{A_I}{\Longrightarrow} \xrightarrow{\tau_I} \stackrel{A_I}{\Longrightarrow} K$ with $H\ \mathcal{R}\ K$. In the following we discuss the two cases.

Case *(a)* $C[P] \xrightarrow{A_I} H$. This administrative reduction has been generated according to Table 12. Then we analyse each of the six items of Table 12 and in every case we show that there exists K such that $C[Q] \stackrel{A_I}{\Longrightarrow} K$ and $H \xrightarrow{A_I} \equiv K$ or $H \equiv K$. In both the cases we can conclude that $H\ \mathcal{R}\ K$.

1. $C[P] \xrightarrow{A_I} H$ according to item 1 in Table 12. If the administrative reduction $P \xrightarrow{A_I} Q$ has been inferred from the items 2–6 of Table 12, then the reductions $C[P] \xrightarrow{A_I} C[Q]$ and $C[P] \xrightarrow{A_I} H$ are non overlapping, hence we conclude that there exists K such that $C[Q] \xrightarrow{A_I}\equiv K$ and $H \xrightarrow{A_I}\equiv K$ by Lemma 9. On the other hand, if the administrative reduction $P \xrightarrow{A_I} Q$ has been inferred from the item 1 of Table 12, then it must be a synchronisation on a channel $u_{\mathsf{r}@A}$. Also $C[P] \xrightarrow{A_I} H$ is a synchronisation on a channel $u'_{\mathsf{r}@A'}$. Now, if $u \neq u'$ and $A \neq A'$ the two transitions are non-overlapping, then we conclude as above. If $u = u'$ and $A = A'$ then we have two possibilities. On the one hand, if $H = C[Q]$ then we conclude that $C[Q] \xRightarrow{A_I} H$. On the other hand, if there are two synchronisation on two different outputs on $u_{\mathsf{r}@A}$ it must be the case that

$$\begin{aligned} C[P] \equiv C'[\ & (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(\overline{u_{\mathsf{r}@A}}\langle h\rangle \,|\, h(\boldsymbol{x}).Q_1) \\ & (\nu k{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(\overline{u_{\mathsf{r}@A}}\langle k\rangle \,|\, k(\boldsymbol{x}).Q_2) \,|\, !\, u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)\ ]. \end{aligned}$$

In this case it is easy to see that $C[P] \xrightarrow{A_I} H \xrightarrow{A_I} K_1$ and $C[Q] \xrightarrow{A_I} K_2$ with

$$\begin{aligned} C[Q] &\equiv C'[\ (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q_1 \,|\, \textsc{Choose}(\boldsymbol{u}, A, h)) \,| \\ &\qquad (\nu k{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(\overline{u_{\mathsf{r}@A}}\langle k\rangle \,|\, k(\boldsymbol{x}).Q_2) \,|\, !\, u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)\ ] \\ H &\equiv C'[\ (\nu k{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(k(\boldsymbol{x}).Q_2 \,|\, \textsc{Choose}(\boldsymbol{u}, A, k)) \,| \\ &\qquad (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(\overline{u_{\mathsf{r}@A}}\langle h\rangle \,|\, h(\boldsymbol{x}).Q_1) \,|\, !\, u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)\ ] \\ K_1 \equiv K_2 &\equiv C'[\ (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q_1 \,|\, \textsc{Choose}(\boldsymbol{u}, A, h)) \,| \\ &\qquad (\nu k{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(k(\boldsymbol{x}).Q_2 \,|\, \textsc{Choose}(\boldsymbol{u}, A, k)) \,|\, !\, u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)\ ]. \end{aligned}$$

2. $C[P] \xrightarrow{A_I} H$ according to item 2 in Table 12. If the administrative reduction $P \xrightarrow{A_I} Q$ has been inferred from the items 1 and 3–6 of Table 12 then we conclude as done above thanks to Lemma 9. On the other hand if the administrative reduction $P \xrightarrow{A_I} Q$ has been inferred from the item 2 of Table 12, then it must be a synchronisation on a channel $u_{\mathsf{w}@A}$. Also $C[P] \xrightarrow{A_I} H$ is a synchronisation on a channel $u'_{\mathsf{w}@A}$. Again, if $H = C[Q]$ then we conclude that $C[Q] \xRightarrow{A_I} H$. Moreover if they are non-overlapping synchronisation then we conclude as above. The only overlapping case is when $u = u'$ and

$$\begin{aligned} C[P] \equiv C'[\ & (\nu \tilde{n}{:}\tilde{T})(\overline{u_{\mathsf{w}@B_1}}\langle \boldsymbol{v}_1\rangle \,|\, Q'_1) \,| \\ & (\nu \tilde{n}{:}\tilde{T})(\overline{u_{\mathsf{w}@B_2}}\langle \boldsymbol{v}_2\rangle \,|\, Q'_2) \,| \\ & (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \,|\, (\nu l{:}\mathsf{rw}\langle\top\rangle)(\,\bar{l}\langle \mathsf{t}\rangle \,|\, R_A\langle h, l\rangle)\,)\ ] \\ C[Q] \equiv C'[\ & (\nu \tilde{n}{:}\tilde{T})(\overline{u_{\mathsf{w}@B_1}}\langle \boldsymbol{v}_1\rangle \,|\, Q'_1) \,| \\ & (\nu \tilde{n}{:}\tilde{T})(\overline{u_{\mathsf{w}@B_2}}\langle \boldsymbol{v}_2\rangle \,|\, Q'_2) \,| \\ & (\nu h{:}\mathsf{rw}\langle\mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \,|\, (\nu l{:}\mathsf{rw}\langle\top\rangle)(\,\bar{l}\langle \mathsf{t}\rangle \,|\, R_A\langle h, l\rangle)\,)\ ] \end{aligned}$$

.... it is easy to conclude as before as thanks to the replicated input channel $\prod_{C<:A} !\, u_{\mathsf{w}@C}(\boldsymbol{z}).\textsc{Test}_l\langle u_{\mathsf{w}@C}(\boldsymbol{z}).\bar{h}\langle \boldsymbol{z}\rangle\rangle$

Contextuality. To be done...

The result of Lemma 10 can be extended to an arbitrary number of administrative reductions.

Corollary 4. *If* $P \stackrel{A_{\mathcal{I}}}{\Longrightarrow} Q$ *then* $\mathcal{I} \models P \approx_A Q$.

Proof. Extend Lemma 10 by induction on the number of reductions $P \stackrel{A_{\mathcal{I}}}{\Longrightarrow} Q$.

C.2 Operational Correspondence

The soundness result relies on the operational correspondence between the source process p and its encoding $\{\!|\, P \,|\!\}_{\Gamma}$. The 'preservation' direction (Lemma 11) is fairly standard, whereas the 'reflection' (Lemma 14) direction needs more care as the encoding is not 'prompt' [9]: in fact, the encoding $\{\!|\, P \,|\!\}_{\Gamma}$ takes several steps to commit a synchronisation of the source process. In this section we show that those steps have the following properties: each reduction leading to a commit *(i)* does not preclude any other reduction and *(ii)* it is administrative, thus not visible to the context.

Lemma 11 (Operational Preservation). *Let* $\mathcal{I} \rhd P$ *a configuration in* aPi@ *and* $\Gamma <: \mathcal{I}$ *such that* $\Gamma \vdash P$. *If* $P \stackrel{\tau}{\longrightarrow} P'$ *then* $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{A_{\mathcal{J}}}{\Longrightarrow}\stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K$, *with* $\mathcal{J} = (\!|\, \mathcal{I} \,|\!)$ *and* $K \equiv (\!|\, P' \,|\!)^*_{\Gamma}$.

Proof. By induction on the derivation of $P \stackrel{\tau}{\longrightarrow} P'$ in aPi@, we prove that there exists a reduction $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{A_{\mathcal{J}}}{\Longrightarrow} H$ which follows the steps in (the same order of) Table 12 with $H \stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K \equiv (\!|\, P' \,|\!)^*_{\Gamma}$

The reverse direction of the operational correspondence is based on the administrative equivalence $\approx_A$. First we introduce an auxiliary notion. Consider the reductions $P \stackrel{A_{\mathcal{I}}}{\Longrightarrow} P' \stackrel{\tau_{\mathcal{I}}}{\longrightarrow} Q$, we say that the sequence $\stackrel{A_{\mathcal{I}}}{\Longrightarrow}$ is *canonical*, and we denote it as $P \stackrel{\lceil A \rceil_{\mathcal{I}}}{\Longrightarrow} P'$ if it includes just the administrative steps required to enable the non administrative synchronisation in P'. More formally, we can say that the canonical sequence has the minimum length among all the administrative sequences $P \stackrel{A_{\mathcal{I}}}{\Longrightarrow} P'$. Thanks to this definition we ca prove a preparatory lemma.

Lemma 12. *Let* $\mathcal{I} \rhd P$ *a configuration in* aPi@ *and* $\Gamma <: \mathcal{I}$ *such that* $\Gamma \vdash P$. *Define* $\mathcal{J} = (\!|\, \mathcal{I} \,|\!)$. *If* $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{A_{\mathcal{J}}}{\Longrightarrow} H \stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K$, *then there exists* H' *such that* $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{\lceil A \rceil_{\mathcal{J}}}{\Longrightarrow} H' \stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K'$ *with* $K \approx_A K'$.

Proof. Let $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{A_{\mathcal{J}}}{\Longrightarrow} H \stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K$ then the fist synchronisation has to be on a channel p_A. Note that every synchronisation on a channel p_A can lead at most to one non administrative synchronisation. If it is the first one, the canonical reduction.

Lemma 13. *Let* $\mathcal{I} \rhd P$ *a configuration in* aPi@ *and* $\Gamma <: \mathcal{I}$ *such that* $\Gamma \vdash P$. *Define* $\mathcal{J} = (\!|\, \mathcal{I} \,|\!)$. *If* $(\!|\, P \,|\!)^*_{\Gamma} \stackrel{\lceil A \rceil_{\mathcal{J}}}{\Longrightarrow} H \stackrel{\tau_{\mathcal{J}}}{\longrightarrow} K$ *then there exists* P' *such that* $P \stackrel{\tau}{\longrightarrow} P'$ *and* $\mathcal{J} \models K \approx_A (\!|\, P' \,|\!)^*_{\Gamma}$.

Lemma 14 (Operational Reflection). *Let $\mathcal{I} \triangleright P$ a configuration in* aPi@ *and $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$. Define $\mathcal{J} = (\!|\, \mathcal{I} \,|\!)$. Assume that $\mathcal{J} \models H \approx_A (\!|\, P \,|\!)^*_\Gamma$ and $H \xrightarrow{\tau} K$. Then either $\mathcal{J} \models H \approx_A K$ or there exists P' such that $P \xrightarrow{\tau} P'$ and $\mathcal{J} \models K \approx_A (\!|\, P' \,|\!)^*_\Gamma$.*

Lemma 15 (Operational Correspondence). *Let $\mathcal{I} \triangleright P$ a configuration in* aPi@ *and $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$. Define $\mathcal{J} = (\!|\, \mathcal{I} \,|\!)$. The following hold:*

1. *If $P \xrightarrow{\tau} P'$ then $(\!|\, P \,|\!)^*_\Gamma \overset{A_\mathcal{J}}{\Longrightarrow} \xrightarrow{\tau_\mathcal{J}} H$ with $\mathcal{J} \models H \approx_A (\!|\, P' \,|\!)^*_\Gamma$.*
2. *If $\mathcal{J} \models H \approx_A (\!|\, P \,|\!)^*_\Gamma$ and $H \xrightarrow{\tau} K$ then there exists P' such that $P \xrightarrow{\tau} P'$ and $\mathcal{J} \models K \approx_A (\!|\, P' \,|\!)^*_\Gamma$.*

The last result we need for soundness it the barb preservation of the encoding.

Lemma 16 (Barb Correspondence). *Let $\mathcal{I} \triangleright P$ a configuration in* aPi@ *and $\Gamma <: \mathcal{I}$ such that $\Gamma \vdash P$. Then the followings hold.*

1. *If $\mathcal{I} \triangleright P{\downarrow_a}$ then $(\!|\, \mathcal{I} \,|\!), h : rw\langle \mathbb{T}_\top \rangle \triangleright \text{LINK}_\Gamma\ (\boldsymbol{u}, \boldsymbol{x})$ IN $\overline{x_{r@A}}\langle h \rangle \mid (\!|\, P \,|\!)^*_\Gamma \Downarrow_h$.*
2. *If $(\!|\, \mathcal{I} \,|\!), h : rw\langle \mathbb{T}_\top \rangle \triangleright \text{LINK}_\Gamma\ (\boldsymbol{u}, \boldsymbol{x})$ IN $\overline{x_{r@A}}\langle h \rangle \mid (\!|\, P \,|\!)^*_\Gamma \Downarrow_h$ then $\mathcal{I} \triangleright P{\downarrow_a}$.*

Theorem 6 (Soundness). *Let $\Gamma <: \mathcal{I}$ and $\Gamma' <: \mathcal{I}$ be such that $\Gamma \vdash P$ and $\Gamma' \vdash Q$. Then $(\!|\, \mathcal{I} \,|\!) \models (\!|\, P \,|\!)^*_\Gamma \approx_\text{S} (\!|\, Q \,|\!)^*_{\Gamma'}$ implies $\mathcal{I} \models P \approx_\text{D} Q$.*

Proof. The candidate is what we expect.

D Proof of Completeness

Theorem 7 (Completeness). *Let $\Gamma <: \mathcal{I}$ and $\Gamma' <: \mathcal{I}$ be such that $\Gamma \vdash P$ and $\Gamma' \vdash Q$. Then $\mathcal{I} \models P \approx_\text{D} Q$ implies $(\!|\, \mathcal{I} \,|\!) \models (\!|\, P \,|\!)^*_\Gamma \approx_\text{S} (\!|\, Q \,|\!)^*_{\Gamma'}$.*

Proof. We recall that $(\!|\, P \,|\!)^*_\Gamma \overset{def}{=} (\!|\, P \,|\!)_\Gamma \mid (\nu t : \mathsf{tbl})(\text{SERVER} \mid t[\,])$ and this represents the initial configuration without pre-synchronisation between the client $(\!|\, P \,|\!)_\Gamma$ and the proxy server. Due to the communication protocol, the proxy server will fulfill the requesters from the client by generating new entries in the table t and the relatives channel managers. The system will then evolve to a configuration of the form

$$(\nu \boldsymbol{k}_1 : \mathbb{S}) \ldots (\nu \boldsymbol{k}_p : \mathbb{S}) \left(Q \mid \prod_{i=1 \ldots p} \text{CHAN}(\boldsymbol{k}_i) \mid (\nu t : \mathsf{tbl})(\,\text{SERVER} \mid t[(\boldsymbol{n}_1, \boldsymbol{k}_1) :: \ldots :: (\boldsymbol{n}_p, \boldsymbol{k}_p)]\,) \right)$$

where Q is such that

$$\Delta \vdash Q \text{ with } \Delta = (\!|\, \Gamma \,|\!), \boldsymbol{k}_1 : \mathbb{T}_{\mathsf{rw}}, \ldots \boldsymbol{k}_p : \mathbb{T}_{\mathsf{rw}}. \tag{2}$$

In order to build the candidate relation for the behavioural equivalence we need to account the observation above and so, given the typing environment Γ in aPi@, we define the following computing environment

$$\begin{aligned} \mathsf{CE}_\Gamma[-] \overset{def}{=}\ & (\nu \boldsymbol{k}_1 : \mathbb{S}) \ldots (\nu \boldsymbol{k}_p : \mathbb{S}) \\ & \left(- \mid Q \mid \textstyle\prod_{i=1 \ldots p} \text{CHAN}(\boldsymbol{k}_i) \mid (\nu t : \mathsf{tbl})(\,\text{SERVER} \mid t[(\boldsymbol{n}_1, \boldsymbol{k}_1) :: \ldots :: (\boldsymbol{n}_p, \boldsymbol{k}_p)]\,) \right) \end{aligned}$$

with the process Q that satisfies the property (2). Intuitively, Q is meant to represents the process generated by the interaction between the proxy server and the context.

Observe that when $Q = \mathbf{0}$ and $p = 0$ we have $\mathsf{CE}_\Gamma[(\!| P |\!)] \equiv (\!| P |\!)^*_\Gamma$, thus the candidate relation $\mathcal{R}$ can be defined as:

$$(\!| \mathcal{I} |\!), a_1 : T_1, \ldots a_n : T_n \models C[\,\mathsf{CE}_\Gamma[(\!| P |\!)]\,] \;\mathcal{R}\; C[\,\mathsf{CE}_{\Gamma'}[(\!| Q |\!)]\,]$$

whenever

1. $\mathcal{I} \models P \approx_{\mathrm{D}} Q$
2. $\Gamma <: \mathcal{I}$ and $\Gamma \vdash P$
3. $\Gamma' <: \mathcal{I}$ and $\Gamma' \vdash Q$
4. $a_i \notin dom((\!| \mathcal{I} |\!))$
5. $C[\,]$ is a context as previously defined

Now it is straightforward to prove that $\mathcal{R} \cup \approx_A$ satisfies the requirements of Definition 7, thus $\mathcal{R} \subseteq \approx_A$ and hence $\mathcal{I} \models P \approx_{\mathrm{D}} Q$ implies $(\!| \mathcal{I} |\!) \models (\!| P |\!)^*_\Gamma \approx_{\mathrm{S}} (\!| Q |\!)^*_{\Gamma'}$ as required by completeness.

Table 12 Administrative Reductions.

Case 1:
$H \equiv \textsc{link}_{\Gamma}\ (\boldsymbol{a}, \boldsymbol{x})\ \textsc{in}\ P \mid \textsc{Server}$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)\textsc{Send}(\boldsymbol{a}, h, r)\ ) \mid \textsc{Server}$
Case 2(a):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)\textsc{Send}(\boldsymbol{a}, h, r)\ ) \mid t[\mathcal{T}]$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)(\ \overline{r}\langle \mathtt{t}, \boldsymbol{k}\rangle \mid r(x, \boldsymbol{y})[x = \mathtt{t}]\overline{h}\langle \boldsymbol{y}\rangle; (\overline{h}\langle \boldsymbol{y}\rangle \mid \textsc{Chan}(\boldsymbol{y}))\ )\ ) \mid t[\mathcal{T}]$
Case 2(b):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)\textsc{Send}(\boldsymbol{a}, h, r)\ ) \mid t[\mathcal{T}]$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\nu \boldsymbol{k}{:}\mathbb{S})(\ h(\boldsymbol{x}).P \mid$
$\qquad \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)(\ \overline{r}\langle \mathtt{f}, \boldsymbol{k}\rangle \mid r(x, \boldsymbol{y})[x = \mathtt{t}]\overline{h}\langle \boldsymbol{y}\rangle; (\overline{h}\langle \boldsymbol{y}\rangle \mid \textsc{Chan}(\boldsymbol{y}))\ )\ ) \mid t[(\boldsymbol{a}, \boldsymbol{k}) :: \mathcal{T}]$
Case 3(a):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)(\ \overline{r}\langle \mathtt{t}, \boldsymbol{k}\rangle \mid r(x, \boldsymbol{y})[x = \mathtt{t}]\overline{h}\langle \boldsymbol{y}\rangle; (\overline{h}\langle \boldsymbol{y}\rangle \mid \textsc{Chan}(\boldsymbol{y}))\ )\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid [\mathtt{t} = \mathtt{t}]\overline{h}\langle \boldsymbol{k}\rangle; (\overline{h}\langle \boldsymbol{k}\rangle \mid \textsc{Chan}(\boldsymbol{k}))\ )$
Case 3(b):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid (\nu r{:}\mathsf{rw}\langle \top, \mathbb{S}\rangle)(\ \overline{r}\langle \mathtt{f}, \boldsymbol{k}\rangle \mid r(x, \boldsymbol{y})[x = \mathtt{t}]\overline{h}\langle \boldsymbol{y}\rangle; (\overline{h}\langle \boldsymbol{y}\rangle \mid \textsc{Chan}(\boldsymbol{y}))\ )\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid [\mathtt{f} = \mathtt{t}]\overline{h}\langle \boldsymbol{k}\rangle; (\overline{h}\langle \boldsymbol{k}\rangle \mid \textsc{Chan}(\boldsymbol{k}))\ )$
Case 4(a):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid [\mathtt{t} = \mathtt{t}]\overline{h}\langle \boldsymbol{k}\rangle; (\overline{h}\langle \boldsymbol{k}\rangle \mid \textsc{Chan}(\boldsymbol{k}))\ )$
$H' \equiv P\{\boldsymbol{k}/\boldsymbol{x}\}$
Case 4(b):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\ h(\boldsymbol{x}).P \mid [\mathtt{f} = \mathtt{t}]\overline{h}\langle \boldsymbol{k}\rangle; (\overline{h}\langle \boldsymbol{k}\rangle \mid \textsc{Chan}(\boldsymbol{k}))\ )$
$H' \equiv P\{\boldsymbol{k}/\boldsymbol{x}\} \mid \textsc{Chan}(\boldsymbol{k})$
Case 5:
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(\overline{u_{\mathsf{r}@A}}\langle h\rangle \mid h(\boldsymbol{x}).Q) \mid !\,u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid \textsc{Choose}(\boldsymbol{u}, A, h)) \mid !\,u_{\mathsf{r}@A}(h).\textsc{Choose}(\boldsymbol{u}, A, h)$
Case 6:
$H \equiv (\nu \tilde{n}{:}\tilde{T})(\overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle \mid Q') \mid (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(\ \overline{l}\langle \mathtt{t}\rangle \mid R_A\langle h, l\rangle)\ )$
$H' \equiv (\nu \tilde{n}_1{:}\tilde{T}_1)(\ (\nu \tilde{n}_2{:}\tilde{T}_2)Q' \mid (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(\ \overline{l}\langle n\rangle \mid R_A\langle h, l\rangle \mid$
$\qquad \mid l(z).[z = \mathtt{t}](\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle) \oplus (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle); (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle)\ )\ )\ )$
Case 7:
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(\ \overline{l}\langle n\rangle \mid Q' \mid R_A\langle h, l\rangle \mid$
$\qquad \mid l(z).[z = \mathtt{t}]\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle \oplus \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle; \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle)\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid [n = \mathtt{t}]\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle \oplus \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle; (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle))\ )$
Case 8(a):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid [\mathtt{t} = \mathtt{t}]\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle \oplus \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle; (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle))\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid \overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle)\ )$
Case 8(b):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid [\mathtt{t} = \mathtt{t}]\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle \oplus \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle; (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle))\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle)\ )$
Case 8(c):
$H \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid [\mathtt{f} = \mathtt{t}]\overline{l}\langle \mathtt{f}\rangle \mid \overline{h}\langle \boldsymbol{v}\rangle \oplus \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle; (\overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle))\ )$
$H' \equiv (\nu h{:}\mathsf{rw}\langle \mathbb{T}_A\rangle)(h(\boldsymbol{x}).Q \mid (\nu l{:}\mathsf{rw}\langle \top\rangle)(Q' \mid R_A\langle h, l\rangle \mid \overline{l}\langle \mathtt{t}\rangle \mid \overline{u_{\mathsf{w}@B}}\langle \boldsymbol{v}\rangle)\ )$
With $R_A\langle h, l\rangle = \prod_{C<:A} !\,u_{\mathsf{w}@C}(\boldsymbol{z}).\textsc{Test}_l\langle u_{\mathsf{w}@C}(\boldsymbol{z}).\overline{h}\langle \boldsymbol{z}\rangle\rangle$