Hard life with weak binders

Massimo Bartoletti, Pierpaolo Degano, Gian Luigi Ferrari

Dipartimento di Informatica, Università di Pisa, Italy

Roberto Zunino

Dipartimento di Informatica e Telecomunicazioni, Università di Trento, Italy

Abstract

We introduce *weak binders*, a lightweight construct to deal with fresh names in nominal calculi. Weak binders do not define the scope of names as precisely as the standard ν-binders, yet they enjoy strong semantic properties. We provide them with a denotational semantics, an equational theory, and a trace inclusion preorder. Furthermore, we present a trace-preserving mapping between weak binders and ν-binders.

Keywords: Nominal calculi, variable binding, alpha-conversion, freshness

1 Introduction

Over the last few years *naming* has been envisaged as a suitable abstraction for capturing and handling a variety of computational concepts, like distributed objects, cryptographic keys, session identifiers. Also, the dynamicity issues usually arising in distributed computing (e.g., network reconfiguration, module versioning) may be usefully explained in terms of naming disciplines such as fresh name generation, binding and scoping rules. The π-calculus [12,18] is probably the most illustrative example of nominal calculi, in which many of the concepts outlined above have been formally modelled and explained. Nominal calculi manipulate names via explicit binders that define their scope. The standard example is the π-calculus restriction operator νn. A ν-binder also declares that a fresh name has to be created. A broad variety of formal theories [8,9,20,17,13,14,4] developed in the last few years shows the intrinsic difficulties of handling naming and freshness.

This paper aims at contributing to this line of research. Our motivating starting point is to understand what is the actual gain in using ν-binders to deal with fresh names. Indeed, the equational theory of ν-binders allows for freely moving them

* This research has been partially supported by EU-FETPI Global Computing Project IST-2005-16004 SENSORIA (Software Engineering for Service-Oriented Overlay Computers)

almost anywhere in a process (except escaping from a recursion). So, one might wonder whether ν-binders can be omitted in a process, and replaced by a more primitive construct, e.g. an atomic action to be interpreted as a *gensym()* that explicitly creates a fresh name.

We introduce a nominal calculus with *weak binders*, a construct for generating fresh names as an atomic action, without explicit ν-binders. Our calculus slightly extends Bergstra and Klop's Basic Process Algebras [3], by allowing parametrized atomic actions $\alpha(r)$, that abstract from dispatching the action α to the object r. Objects can be freshly created through the special action $\mathit{new}(n)$, our "weak binder".

We study under which conditions a weakly bound process can be treated coherently with a process with ν-binders. For instance, in the weakly bound process $p = \mathit{new}(n) \cdot \alpha(n) + \mathit{new}(m) \cdot \alpha(m)$ there is no confusion between the scopes of the "bound" names n and m, and so p is equivalent to the "strongly bound" process $P = \nu n.\nu m.(\mathit{new}(n) \cdot \alpha(n) + \mathit{new}(m) \cdot \alpha(m))$. We shall then say that p is *well-bound*, and that P is its *bindification*. This transformation makes precise the scopes of names in weakly bound processes, by inserting the ν-binders at the right points. This is not always possible, however, e.g. in the process $\mathit{new}(n) \cdot (\varepsilon + \mathit{new}(n)) \cdot \alpha(n)$ there is an inherent ambiguity, because we cannot tell whether the action α has to be done on the object created by the first or by the second *new*. When bindification is possible, we prove that the semantics of the weakly bound and of the bindified processes are trace equivalent.

A further contribution is a trace inclusion preorder $\precsim$ for weakly bound processes: when $p \precsim q$, the traces of p are included in those of q. We compare this preorder with a trace inclusion preorder for strongly bound processes. Preorders of processes are a relevant and non-trivial aspect of subtyping/subeffecting for type and effect systems [1]. Also, thay can be udes to study the compliance of contracts with implementations and subcontract relations in calculi for Web services [5,6].

We envisage the impact of our approach as follows. Our main result is the formal definition of a methodology for handling the freshness of names without resorting to explicit binders. The overall outcome of our semantical investigation consists in the full characterization of weak binders. We have proved that weak binders still enjoy interesting semantic properties, comparably to what can be obtained through ν-binders. We have exploited weak binders to develop the static machinery (a type and effect system and a model checker) of a linguistic framework for resource usage control [1]. As a downside, we have found that weak binders, having a weaker structure than ν-binders, may make the life hard when going into the proofs.

The paper is organized as follows. We first introduce a calculus with explicit ν-binders, we give its operational and denotational semantics, and we show them fully abstract. We then remove ν-binders, and define a denotational semantics and an equational theory of weakly bound processes. Then, we define the bindify transformation, and we state its correctness: the bindification of a weakly bound process p is trace equivalent to p. After that, we compare the equational theories and the trace inclusion preorders of strongly bound and weakly bound processes. We conclude by reporting our experience about using weak binders, and by discussing some related work. Because of space limitations, here we shall omit the proofs of our statements. All the proofs are available in the companion technical report [2].

2 Strongly bound processes

We now introduce a process calculus with name binders, building upon Basic Process Algebras (BPAs, [3]). Our calculus shares with BPAs the primitives for sequential composition, for non-deterministic choice, and for recursion (though with a slightly different syntax). Quite differently from BPAs, our atomic actions (called *events*) have a parameter, which indicates the *resource* upon which the action is performed. Resources $r, r', \ldots \in \mathsf{Res}$ are system objects that can either be already available in the environment or be created at run-time. Resources can be accessed through a given finite set of *actions* $\alpha, \alpha', \mathit{new}, \ldots \in \mathsf{Act}$. The special action new represents the creation of a fresh resource: this means that for each dynamically created resource r, the event $\mathit{new}(r)$ must precede any other $\alpha(r)$. [1] An *event* $\alpha(r) \in \mathsf{Ev}$ abstracts from accessing the resource r through the action α. We also have events the target of which is a *name* $n, n', \ldots \in \mathsf{Nam}$, to be bound by an outer ν. Since the name binders are explicit in this calculus, we call its processes *strongly bound*, whose abstract syntax is given in Def. 2.1.

Definition 2.1 Syntax of strongly bound processes

P, Q	$::=$	ε	empty process
		h	variable
		$\alpha(\rho)$	event ($\rho \in \mathsf{Res} \cup \mathsf{Nam}$)
		$\nu n.P$	resource binding
		$P \cdot Q$	sequential composition
		$P + Q$	choice
		$\mu h.P$	recursion

In a recursion $\mu h.P$, the free occurrences of h in P are bound by μ. In the construct $\nu n.\, P$, the ν acts as a binder for the free occurrences of the name n in P. The intended meaning is to keep track of the binding between n and a freshly created resource. A process is *closed* when it has no free names and variables.

The behaviour of a strongly bound process is described by the set of sequential traces (typically denoted by $\eta, \eta', \ldots \in \mathsf{Ev}^*$) of its events. As usual, ε denotes the empty trace, and $\varepsilon\eta = \eta = \eta\varepsilon$. The trace semantics $[\![P]\!]^{op}$ of a closed, strongly bound process P, is a function from finite set of resources to sets of traces (Def. 2.2). We first introduce an auxiliary labelled transition relation $P, \mathcal{R} \xrightarrow{a} P', \mathcal{R}'$ (where $a \in \mathsf{Ev} \cup \{\varepsilon\}$ and $\mathcal{R}, \mathcal{R}' \subset \mathsf{Res}$). The set $\mathcal{R}$ in configurations accumulates the resources created at run-time, so that no resource can be created twice, e.g.

$$\begin{aligned}(\nu n.\, \mathit{new}(n)) \cdot (\nu n.\, \mathit{new}(n)), \emptyset &\xrightarrow{\varepsilon} \mathit{new}(r_0) \cdot (\nu n.\, \mathit{new}(n)), \{r_0\} \\ &\xrightarrow{\mathit{new}(r_0)} \nu n.\, \mathit{new}(n), \{r_0\} \\ &\not\longrightarrow \mathit{new}(r_0), \{r_0\}\end{aligned}$$

[1] We conjecture this is a decidable property, e.g. suitably adapting the techniques of [10] should enable us to identify and discard those P that produce ill-formed traces where an $\alpha(r)$ comes before a $\mathit{new}(r)$.

The labelled transition relation is then exploited in the definition of $[\![P]\!]^{op}$, which contains two kinds of traces. First, we include in $[\![P]\!]^{op}$ all the traces for *terminating* executions, i.e. those leading to ε. Then, we add all the prefixes of *all* executions, and mark these truncated traces with a trailing ! symbol. Here, we just let ! be a distinguished event not in Ev. Including these $\eta!$ prefixes in $[\![P]\!]^{op}$ is useful, since they allow us to observe non-terminating computations.

Definition 2.2 Trace semantics of strongly bound processes

$$\alpha(r),\, \mathcal{R} \xrightarrow{\alpha(r)} \varepsilon,\, \mathcal{R}\cup\{r\} \qquad \nu n.\, P,\, \mathcal{R} \xrightarrow{\varepsilon} P\{r/n\},\, \mathcal{R}\cup\{r\} \quad \text{if } r \notin \mathcal{R}$$

$$\varepsilon\cdot P,\, \mathcal{R} \xrightarrow{\varepsilon} P,\, \mathcal{R} \qquad P\cdot Q,\, \mathcal{R} \xrightarrow{a} P'\cdot Q,\, \mathcal{R}' \text{ if } P,\, \mathcal{R} \xrightarrow{a} P',\, \mathcal{R}'$$

$$P+Q,\, \mathcal{R} \xrightarrow{\varepsilon} P,\, \mathcal{R} \qquad P+Q,\, \mathcal{R} \xrightarrow{\varepsilon} Q,\, \mathcal{R} \qquad \mu h.\, P,\, \mathcal{R} \xrightarrow{\varepsilon} P\{\mu h.\, P/h\},\, \mathcal{R}$$

The trace semantics $[\![P]\!]^{op}(\mathcal{R})$ is then defined as

$$[\![P]\!]^{op}(\mathcal{R}) = \{\, \eta \mid P,\, \mathcal{R} \xrightarrow{\eta} \varepsilon,\, \mathcal{R}' \,\} \cup \{\, \eta! \mid P,\, \mathcal{R} \xrightarrow{\eta} P',\, \mathcal{R}' \,\}$$

Example 2.3 Consider the following strongly bound processes:

$$P_0 = \mu h.\, \alpha(r)\cdot h \qquad P_1 = \mu h.\, h\cdot\alpha(r) \qquad P_2 = \mu h.\, \nu n.\, (\varepsilon + \alpha(n)\cdot h)$$

Then, $[\![P_0]\!]^{op}(\emptyset) = \alpha(r)^*!$, i.e. P_0 generates traces with an arbitrary, finite number of $\alpha(r)$. Note that all the traces of P_0 are non-terminating (as indicated by the !) since there is no way to exit from the recursion. Instead, $[\![P_1]\!]^{op}(\emptyset) = \{!\}$, i.e. P_1 loops forever, without generating any events. The semantics of $[\![P_2]\!]^{op}(\emptyset)$ consists of all the traces of the form $\alpha(r_1)\cdots\alpha(r_k)$ or $\alpha(r_1)\cdots\alpha(r_k)!$, for all $k \geq 0$ and pairwise distinct resources r_i. □

The denotational semantics $[\![P]\!]^s_\theta$ of a strongly bound process P is given below (Def. 2.5) as a function Y in a cpo D_s, which we define now. We first let D_0 be $\{\, X \subseteq \mathsf{Ev}^* \cup \mathsf{Ev}^*! \mid\, ! \in X \,\}$, that is the cpo of sets X of traces such that $! \in X$. Then we let D_h be the cpo $\mathcal{P}_{\mathit{fin}}(\mathsf{Res}) \rightharpoonup D_0$ (where $\rightharpoonup$ denotes partiality). Finally, D_s is the cpo $(\mathsf{Nam} \to \mathsf{Res}) \to D_h$ Intuitively, $[\![P]\!]^s_\theta(\chi)(\mathcal{R})$ contains all the possible traces of P. The first argument $\chi \in \mathsf{Nam} \to \mathsf{Res}$ records the bindings between names and resources. The second argument $\mathcal{R} \in \mathcal{P}_{\mathit{fin}}(\mathsf{Res})$ is a finite set of resources which indicates those already used, so to make them unavailable for future creations. As usual, the parameter θ binds the free variables of P (in our case, to values in D_h).

Before giving the semantics, it is convenient to introduce some auxiliary definitions that help in composing traces sequentially (see Def. 2.4 below).

The operator $\odot$ ensures that all the events after a ! are discarded. For instance, the process $P = (\mu h.\, h)\cdot\alpha(r)$ will never fire the event $\alpha(r)$, because of the infinite loop that precedes the event. The composition of the semantics of the first component $\mu h.\, h$ is $\{\,!\,\}$, while the semantics of $\alpha(r)$ is $\{\,!, \alpha(r), \alpha(r)\,!\,\}$. Combining the two semantics results in $\{\,!\,\} \odot \{\,!, \alpha(r), \alpha(r)\,!\,\} = \{\,!\,\}$.

The operator $\boxdot$ takes two semantics and combines their traces sequentially. While doing that, it records the resources created, so to avoid that a resource is generated twice. For instance, let $P = (\nu n.\, new(n)) \cdot (\nu n'.\, new(n'))$. The traces of the right-hand side $\nu n'.\, new(n')$ must not generate the same resources used in the left-hand side $\nu n.\, new(n)$, e.g. $new(r_0)new(r_0)$ is *not* a possible trace of P.

The definition of $\boxdot$ exploits the auxiliary operator R, that computes the set of resources occurring in a trace η. Also, $\downarrow \in \mathsf{R}(\eta)$ indicates that η is terminating, i.e. it does not contain any $!$s.

Definition 2.4 Let $X \in D_0$, and $x \in \mathsf{Ev} \cup \{\,!\,\}$. We define $x \odot X$ and $\eta \odot X$ as:

$$x \odot X = \begin{cases} \{\, x\,\eta \mid \eta \in X \,\} & \text{if } x \neq ! \\ \{x\} & \text{if } x = ! \end{cases} \qquad (a_1 \cdots a_n) \odot X = a_1 \odot \cdots \odot a_n \odot X$$

Given $Y_0, Y_1 \in D_s$, their composition $Y_0 \boxdot Y_1$ is:

$$Y_0 \boxdot Y_1 = \lambda\chi, \mathcal{R}.\ \bigcup \{\, \eta_0 \odot Y_1(\chi)(\mathcal{R} \cup \mathsf{R}(\eta_0)) \mid \eta_0 \in Y_0(\chi)(\mathcal{R}) \,\}$$

where $\mathsf{R}(\eta)$ is defined inductively as follows:

$$\mathsf{R}(\varepsilon) = \{\downarrow\} \qquad \mathsf{R}(\eta\, \alpha(r)) = \mathsf{R}(\eta) \cup \{r\} \ \text{ if } ! \notin \eta \qquad \mathsf{R}(\eta\,!\eta') = \mathsf{R}(\eta) \setminus \{\downarrow\}$$

Definition 2.5 Denotational semantics of strongly bound processes

$$[\![\varepsilon]\!]^s_\theta = \lambda\chi, \mathcal{R}.\ \{\,!, \varepsilon\} \qquad [\![h]\!]^s_\theta = \lambda\chi, \mathcal{R}.\ \theta(h)(\mathcal{R})$$

$$[\![\alpha(\rho)]\!]^s_\theta = \lambda\chi, \mathcal{R}.\ \begin{cases} \{\,!, \alpha(\rho), \alpha(\rho)\,!\} & \text{if } \rho = r \\ \{\,!, \alpha(\chi(n)), \alpha(\chi(n))\,!\} & \text{if } \rho = n \end{cases} \qquad [\![P \cdot Q]\!]^s_\theta = [\![P]\!]^s_\theta \boxdot [\![Q]\!]^s_\theta$$

$$[\![\nu n.\, P]\!]^s_\theta = \lambda\chi, \mathcal{R}.\ \textstyle\bigcup_{r \notin \mathcal{R}} [\![P]\!]^s_\theta(\chi\{r/n\})(\mathcal{R} \cup \{r\}) \qquad [\![P + Q]\!]^s_\theta = [\![P]\!]^s_\theta \sqcup [\![Q]\!]^s_\theta$$

$$[\![\mu h.P]\!]^s_\theta = \lambda\chi, \mathcal{R}.\ \textstyle\bigcup_{i \geq 0} \left(\lambda Z.\, \lambda\bar{\mathcal{R}}.\ [\![P]\!]^s_{\theta\{Z/h\}}(\chi)(\bar{\mathcal{R}})\right)^i (\lambda\mathcal{R}.\{\,!\}) \ (\mathcal{R})$$

The semantics of an event $\alpha(r)$ comprises the possible "truncations" of $\{\alpha(r)\}$, i.e. $!, \alpha(r)\,!$ and $\alpha(r)$ (notice that $!$ is always included in the semantics of all P, coherently with the definition of the trace semantics). The semantics of $\alpha(n)$ is similar, but it looks in χ for the resource associated with n. The semantics of $\nu n.\, P$ joins the semantics of P, where the parameters $\mathcal{R}$ and χ are updated to record the binding of n with r, for *all* the resources r not yet used in $\mathcal{R}$. The semantics of $P \cdot Q$ combines the semantics of P and Q with the operator $\boxdot$. The semantics of $P + Q$ is the least upper bound of the semantics of P and Q. The semantics of a recursion $\mu h.\, P$ is the least upper bound of $f^i(\lambda\mathcal{R}.\{\,!\})$, where $f(Z) = \lambda\bar{\mathcal{R}}.\ [\![P]\!]^s_{\theta\{Z/h\}}(\chi)(\bar{\mathcal{R}})$. Since f is continuous and $\lambda\mathcal{R}.\{\,!\}$ is the bottom element of the cpo D_h, then $f^i(\lambda\mathcal{R}.\{\,!\})$ is an ω-chain, and its least upper bound is the least fixed point of f.

The following theorem states that the denotational semantics of strongly bound processes is fully abstract with respect to their operational semantics.

Theorem 2.6 (Full abstraction) *Let $\mathcal{R}$ be a finite sets of resources, and let $\emptyset$ be the empty mapping. Then, for all closed strongly bound processes P:*

$$[\![P]\!]^{op}(\mathcal{R}) \ = \ [\![P]\!]^s_\emptyset(\emptyset)(\mathcal{R})$$

3 Weakly bound processes

In strongly bound processes, the ν-binders precisely define the scope of names. However, classical equational theories [11] for these processes usually allow binders to be floated out, towards the top-level, e.g. in $P_0 + \nu n.\, P_1 = \nu n.\, P_0 + P_1$, under the usual conditions. Indeed, the binder can always be brought outside a context, provided that 1) no recursion boundary is crossed, i.e. in $\mu h.\, \nu n.P$ the binder cannot be moved outside, and 2) no name in the context is captured. Because of this, it is often convenient to define a *normal form* for processes, where all the binders are placed at their top-most position, i.e. at the top-level or just under a recursion. These are standard and well-known facts about process algebras.

One might wonder what information is actually carried by the presence of the ν-binders. From an operational point of view, we can see them as the points where resources are created. In our setting, this information is also carried by the *new* events. Therefore, it is interesting to study whether, under this assumption, we can neglect placing binders in our processes, and let the *new* events to define, at least in some loose way, the scope of names.

To this purpose, we now introduce *weakly bound* processes, which have no ν-binders (Def. 3.1). For instance, let $p = \mathit{new}(n) \cdot \alpha(n) + \mathit{new}(m) \cdot \alpha'(m)$. Here, the event $\mathit{new}(n)$ binds the name n, while $\mathit{new}(m)$ binds m. We shall later on define a semantics of weakly bound processes such that p is equivalent to the strongly bound process $(\nu n.\mathit{new}(n) \cdot \alpha(n)) + (\nu m.\mathit{new}(m) \cdot \alpha'(m))$, as the intuition suggests.

While weakly bound processes may make our reasoning more agile, we must not neglect that, unlike in the strongly bound case, weakly bound processes are possible where names have no clear scope. E.g., in $\mathit{new}(n) \cdot (\mathit{new}(n) + \varepsilon) \cdot \alpha(n)$ it is not clear what binds the last occurrence of n. Roughly, these troublesome processes are those that can be derived from a strongly bound process by neglecting to α-convert some name while enlarging the scope of a ν-binder, yielding to unwanted name captures. We shall return to this point in Sect. 4.

Definition 3.1 Syntax of weakly bound processes

p, q	$::=$	ε	empty process
		h	variable
		$\alpha(\rho)$	event ($\rho \in \mathsf{Res} \cup \mathsf{Nam}$)
		$\mathit{new}(n)$	resource creation
		$p \cdot q$	sequential composition
		$p + q$	choice
		$\mu h.p$	recursion

Free names in weakly bound processes have to be dealt with quite peculiarly, because of the absence of ν-binders. Consider e.g. $p = p' \cdot \alpha(n)$. To tell whether n is free in p we have to inspect p'. For example if $p' = \mathit{new}(n)$, we shall consider n as non-free; instead, if $p' = \varepsilon$, the name n is obviously free. Given p', we define which names are *bound* by p', so to extend the scope of the names of p' when it occurs at the left of another process, as in $p' \cdot p''$. Non-determinism complicates matters: it might happen than a process p' binds a name to a resource only in some, but not all, of its execution, e.g. $p' = \mathit{new}(n) + \varepsilon$. So, we define two sets of names, the *must-bound* names $bn^{\Box}(p)$ and the *may-bound* names $bn^{\Diamond}(p)$, for the names that are bound in *every* execution of p, and the names that are bound in *some* execution of p, respectively (see Def. 3.2). So, if $p' = \mathit{new}(n) + \varepsilon$, we have $bn^{\Box}(p') = \emptyset$ and $bn^{\Diamond}(p') = \{n\}$. Note that the sets $bn^{\Box}(p')$ and $bn^{\Diamond}(p')$ can be seen as static approximations for the actual run-time bindings created by the process p'. Of course, $bn^{\Box}(p) \subseteq bn^{\Diamond}(p)$. Note that no "weak" binding can escape a recursion, as real ν-binders cannot cross recursive contexts. So, in $(\mu h.\, \mathit{new}(n) \cdot h + \varepsilon) \cdot \alpha(n)$ the last n is free, and is unrelated to the $\mathit{new}(n)$ event under the μh. Therefore, the bound names (both *must* and *may*) of a recursion are empty.

Definition 3.2 Must-bound names $\mathbf{bn}^{\Box}(\mathbf{p})$ and may-bound names $\mathbf{bn}^{\Diamond}(\mathbf{p})$

$$bn^{\Box}(\varepsilon) = bn^{\Box}(h) = \emptyset \qquad bn^{\Box}(\alpha(\rho)) = \begin{cases} \{n\} & \text{if } \alpha = \mathit{new} \text{ and } \rho = n \\ \emptyset & \text{otherwise} \end{cases}$$

$$bn^{\Box}(p \cdot q) = bn^{\Box}(p) \cup bn^{\Box}(q) \qquad bn^{\Box}(p + q) = bn^{\Box}(p) \cap bn^{\Box}(q) \qquad bn^{\Box}(\mu h.\, p) = \emptyset$$

$$bn^{\Diamond}(p) = \begin{cases} bn^{\Box}(p) & \text{if } p = \varepsilon,\ h,\ \alpha(\rho),\ \mu h.\, p' \\ bn^{\Diamond}(p') \cup bn^{\Diamond}(p'') & \text{if } p = p' + p'' \text{ or } p = p' \cdot p'' \end{cases}$$

We can now define the free names $\mathit{fn}(p)$ of a weakly bound process p. This is mostly standard, except that must-bound names are checked to single out captured names. The choice of using must-bound names instead of may-bound names is done so that, e.g. in $p = (\mathit{new}(n) + \varepsilon) \cdot \alpha(n)$ we consider n as free. This has the nice property that, whenever $\mathit{fn}(p) = \emptyset$, in no execution of p we will attempt to fire an event $\alpha(n)$ without a proper binding for n.

Definition 3.3 Free names $\mathbf{fn}(\mathbf{p})$

$$\mathit{fn}(h) = \emptyset \qquad \mathit{fn}(\alpha(\rho)) = \begin{cases} \{n\} & \text{if } \rho = n \text{ and } \alpha \neq \mathit{new} \\ \emptyset & \text{otherwise} \end{cases} \qquad \mathit{fn}(\mu h.\, p) = \mathit{fn}(p)$$

$$\mathit{fn}(\varepsilon) = \emptyset \qquad \mathit{fn}(p \cdot q) = \mathit{fn}(p) \cup (\mathit{fn}(q) \setminus bn^{\Box}(p)) \qquad \mathit{fn}(p + q) = \mathit{fn}(p) \cup \mathit{fn}(q)$$

We now define a denotational semantics of weakly bound processes. Unlike in the case of strongly bound processes, where the result of the semantics was a set of event traces, here we also need to keep track of the bindings generated by the *new* events. We shall then use sets of pairs (η, χ) instead of sets of traces η. Note that this difference – the extra χ – between the semantic domains for the strongly/weakly bound processes is exactly the same difference between the classic domains for programming languages with static/dynamic scoping.

As we did with strongly bound processes in Def. 2.4, we introduce the auxiliary operators $\odot$ and $\boxdot$ to handle sequential composition.

The operator $\odot$ merges two pairs (η, χ), so ensuring that all the events after a ! are discarded, as well as the bindings created after the !. For example, $(\eta!, \chi) \odot (\eta', \chi') = (\eta!, \chi)$, discarding both η' and χ'. Here we also use two cpos, D_1 and D_w, to play the role of D_0 and D_s used for strongly bound processes. We let D_1 be the cpo of sets X of pairs (η', χ') such that there exists a pair in X with $\eta' = !$. Formally, D_1 is the cpo $\{\, X \subseteq (\mathsf{Ev}^* \cup \mathsf{Ev}^*\, !) \times (\mathsf{Nam} \to \mathsf{Res}) \mid \exists \chi'.\, (\,!, \chi') \in X \,\}$.

Definition 3.4 Let $a \in \mathsf{Ev} \cup \{\,!\,\}$, $X \in D_1$, $(\eta, \chi), (\eta', \chi') \in X$. We define $\odot$ as:

$$(a, \chi) \odot (\eta', \chi') = \begin{cases} (a, \chi) & \text{if } a = ! \\ (a\eta', \chi') & \text{otherwise} \end{cases}$$

$$(\eta, \chi) \odot (\eta', \chi') = (a_1, \chi) \odot \cdots \odot (a_k, \chi) \odot (\eta', \chi') \;\text{ if } \eta = a_1 \cdots a_k$$

$$(\eta, \chi) \odot X = \{\, (\eta, \chi) \odot (\bar\eta, \bar\chi) \mid (\bar\eta, \bar\chi) \in X \,\}$$

The operator $\boxdot$ takes two semantics Y_0 and Y_1 and combines their traces sequentially. In $Y_0 \boxdot Y_1$ the bindings (i.e. the χ) generated by Y_0 are passed to Y_1, so that e.g. $\mathit{new}(n) \cdot \alpha(n)$ works as expected.

Definition 3.5 Let $D_w = (\mathsf{Nam} \to \mathsf{Res}) \to \mathcal{P}_{\mathit{fin}}(\mathsf{Res}) \rightharpoonup D_1$ be the cpo of functions from functions from names to resources, to the finite subsets of Res to D_1 (where $\rightharpoonup$ denotes partiality). Given $Y_0, Y_1 \in D_w$, their composition $Y_0 \boxdot Y_1$ is:

$$Y_0 \boxdot Y_1 = \lambda\chi, \mathcal{R}.\ \bigcup \{\, (\eta_0, \chi_0) \odot Y_1(\chi_0)(\mathcal{R} \cup \mathsf{R}(\eta_0)) \mid (\eta_0, \chi_0) \in Y_0(\chi)(\mathcal{R}) \,\}$$

The denotational semantics $[\![p]\!]^w_\theta$ of a weakly bound process p is defined as a function $Y \in D_w$, where we assume that $Y(\chi)(\mathcal{R})$ is defined only if $\mathcal{R} \supseteq \mathit{ran}(\chi)$. The parameter θ is a mapping from the free variables h of p to D_h.

Definition 3.6 Denotational semantics of weakly bound processes

Below, we let $\mathit{set}_\chi I = \{\, (\eta, \chi) \mid \eta \in I \,\}$.

$$[\![\varepsilon]\!]^w_\theta = \lambda\chi, \mathcal{R}.\ \mathit{set}_\chi\{\,!, \varepsilon\} \qquad [\![h]\!]^w_\theta = \lambda\chi, \mathcal{R}.\ \mathit{set}_\chi \theta(h)(\mathcal{R})$$

$$[\![\alpha(\rho)]\!]^w_\theta = \lambda\chi, \mathcal{R}. \begin{cases} \mathit{set}_\chi\{\,!, \alpha(\rho), \alpha(\rho)\,!\} & \text{if } \rho = r \\ \mathit{set}_\chi\{\,!, \alpha(\chi(n)), \alpha(\chi(n))\,!\} & \text{if } \rho = n \in \mathit{dom}(\chi) \\ \{(\,!, \chi)\} \cup \bigcup_{r \notin \mathcal{R}} \mathit{set}_{\chi\{r/n\}}\{\alpha(r), \alpha(r)\,!\} & \text{if } \begin{array}{l} \rho = n \notin \mathit{dom}(\chi) \\ \text{and } \alpha = \mathit{new} \end{array} \end{cases}$$

$$[\![p \cdot q]\!]^w_\theta = [\![p]\!]^w_\theta \boxdot [\![q]\!]^w_\theta \qquad [\![p + q]\!]^w_\theta = [\![p]\!]^w_\theta \sqcup [\![q]\!]^w_\theta$$

$$[\![\mu h.p]\!]^w_\theta = \lambda\chi, \mathcal{R}.\ \mathit{set}_\chi \bigcup_{i \geq 0} \left(\lambda Z.\lambda\bar{\mathcal{R}}.\, \mathit{fst}([\![p]\!]^w_{\theta\{Z/h\}}(\chi|_{\mathit{dom}(\chi)\setminus \mathit{bn}^\diamond(p)})(\bar{\mathcal{R}}))\right)^i (\lambda\mathcal{R}.\{\,!\}) \;(\mathcal{R})$$

The semantics above is similar to the one for strongly bound processes, so we just comment on the differences. First, each trace η has now been bundled with its generated bindings χ. Related to this, now the $\mathit{new}(n)$ event creates the actual binding, which augments the χ at hand. Note that we assume the operators $\cup$ and $\sqcup$ to be undefined when one of the arguments is undefined. This must hold also for $\odot$ and $\boxdot$, so making e.g. the semantics of $(new(n)+\varepsilon)\cdot\alpha(n)$ undefined when $n \notin dom(\chi)$, since in one branch $\alpha(n)$ is evaluated without a proper binding for n.

The semantics of recursion variables h is peculiar. First, note that we chose the semantics parameter θ so that $\theta(h)$ is an element of D_h and not of D_w. This is because, when recursion is involved, the bindings of names must not be propagated: this is strictly related to the fact that ν-binders cannot cross a recursive context in strongly bound processes. For example, in the strongly bound process $\mu h.\, \nu n.\, P{\cdot}h{\cdot}P'$ there is no way for the resource bound to n to be "passed" to the inner "call" to h; similarly, if the inner "call" generates a binding, it cannot be "returned" so to interfere with P'. Of course, this would change if we allowed a more complex form of recursion where h can take a resource as an argument. Returning to the semantics of h, since $\theta(h) \in D_h$ needs no χ, then it suffices to pass it an $\mathcal{R}$, and then augment the returned set of traces with χ. This is accomplished by the set_χ function.

The semantics of the recursion $\mu h.\, p$ is quite similar to the one for strongly bound processes. For the reasons explained above, we compute a fixed point over D_h and not D_w. This means that we have to adapt the semantics of p, which is in D_w, to a function in D_h. More concretely, we just need to provide χ to $[\![p]\!]^w$ and ignore the χ returned by it. The latter is done by a trivial left projection, the *fst* in the actual formula. The χ we pass, instead, is the top-level χ – the one provided to the whole recursive process – after the bindings which affect $bn^\diamond(p)$ have been filtered out. This filtering is needed to prevent name confusion e.g. in $new(n)\cdot(\mu h.\, new(n)\cdot p)$, where the outer n is unrelated to the inner one. Aside from this, the fixed point is computed exactly as for the strongly bound processes, exploiting the continuity of $f(Z) = \lambda\bar{\mathcal{R}}.\, fst([\![p]\!]^w_{\theta\{Z/h\}}(\chi')(\bar{\mathcal{R}}))$.

4 Bindifying weakly bound processes

To make precise the scope of names in weakly bound processes, we shall translate them into strongly bound processes, through the transformation *bindify* (Def. 4.3). This transformation will insert the ν-binders at the right points, provided that the introduced scopes of names do not interfere dangerously. We shall call *well-bound* those weakly bound processes that can be safely translated into strongly bound ones. To help the intuition, we shall first give some examples.

Example 4.1 Consider the weakly bound processes:

$$p_1 = new(n)\cdot new(n)\cdot\alpha(n) \qquad p_2 = \alpha(n)\cdot new(n) \qquad p_3 = new(n)+\alpha(n)$$

$$p_4 = (\varepsilon + new(n))\cdot\alpha(n) \quad p_5 = \mu h.new(n).h \quad p_6 = new(n)\cdot(\mu h.\,(\varepsilon + new(n)\cdot h))\cdot\alpha(n)$$

The processes p_1, p_2, p_3, p_4 are not well-bound. If p_1 were such, its bindification would either be $\nu n.\, new(n){\cdot}(\nu n.\, new(n)){\cdot}\alpha(n)$ – where α is performed on the resource generated by the outer ν-binder – or $\nu n.\, new(n)\cdot(\nu n.\, new(n)\cdot\alpha(n))$ – where α acts on the resource of the inner binder. Because of this possible ambiguity, we treat p_1 as not well-bound. The process p_2 is not well-bound, too, because it would produce an ill-formed trace $\alpha(r)new(r)$ where the event $\alpha(r)$ is fired *before* the event $new(r)$ that signals the creation of r. Similarly, the process p_3 is not well-bound, because its bindification would give rise to the ill-formed trace $\alpha(r)$. The process p_4 is not well-bound, because choosing the branch ε would lead to a similar situation. Observe that the denotation of p_1 contains the non-sense trace $new(r)new(r)\alpha(r)$, while the semantics of p_2, p_3 and p_4 are undefined, because $\boxdot$ and $\sqcup$ are strict. Finally, the process p_5 is well-bound: it loops over $new(n)$, generating a fresh resource at each iteration. Also, p_6 is well-bound, because the μ-binder clearly separates the scope of the outer $new(n)$ from that of the inner one. □

The following definition formalizes when a process is well-bound. The empty process, variables and events are well-bound. A recursion is well-bound when its body is such. A choice $p + q$ is well-bound when both p and q are well-bound. Additionally, we require that the may-bound names of p are disjoint from the free names of q, and *viceversa* (e.g. $new(n) + \alpha(n)$ is not well-bound). A sequence $p \cdot q$ is well-bound when both p and q are well-bound, and furthermore (i) the may-bound names of q are disjoint from the names of p (e.g. $\alpha(n) \cdot new(n)$ and $new(n) \cdot new(n)$ are not well-bound), and (ii) the free names of q are either must-bound in p, or they are not may-bound in p (e.g. $(\varepsilon + new(n)) \cdot \alpha(n)$ is not well-bound).

Definition 4.2 Well-bound processes

A weakly bound process p is *well-bound* when $wb(p)$, defined inductively as:

$$wb(\varepsilon) = wb(h) = wb(\alpha(\rho)) = true \qquad wb(\mu h.\, p) \text{ if } wb(p)$$

$$wb(p + q) \text{ if } wb(p), wb(q), bn^{\Diamond}(p) \cap fn(q) = bn^{\Diamond}(q) \cap fn(p) = \emptyset$$

$$wb(p \cdot q) \text{ if } wb(p), wb(q), bn^{\Diamond}(q) \cap (bn^{\Diamond}(p) \cup fn(p)) = (bn^{\Diamond}(p) \setminus bn^{\Box}(p)) \cap fn(q) = \emptyset$$

We now introduce the *bindify* transformation, which is defined on well-bound processes only. The may-bound names are lifted to the leftmost position of the bindified process, and they are placed within the scope of a ν-binder. In the case of a recursion $\mu h.\, p$, the may-bound names of p are lifted to the leftmost position *within* the recursion, i.e. they do not escape the scope of the μh.

Definition 4.3 Bindification

If $wb(p)$, the bindification $bindify(p)$ of p is a strongly bound process, defined as:

$$bindify(p) = \nu\, bn^{\Diamond}(p).\, \beta(p)$$

where the auxiliary operator β is defined inductively as follows:

$$\beta(\varepsilon) = \varepsilon \qquad \beta(\alpha(\rho)) = \alpha(\rho) \qquad \beta(p + q) = \beta(p) + \beta(q)$$

$$\beta(h) = h \qquad \beta(\mu h.\, p) = \mu h.\, bindify(p) \qquad \beta(p \cdot q) = \beta(p) \cdot \beta(q)$$

Example 4.4 Recall from Sect. 1 the process $p = new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m)$. It is easy to check that p is well-bound, and that its may-bound names are:

$$bn^{\Diamond}(p) = bn^{\Diamond}(new(n) \cdot \alpha(n)) \cup bn^{\Diamond}(new(m) \cdot \alpha(m)) = \{n, m\}$$

Then the bindification of p is the strongly bound process:

$$bindify(p) \;=\; \nu n.\nu m.(new(n) \cdot \alpha(n) + new(m) \cdot \alpha(m))$$

Example 4.5 Recall the process $p_5 = \mathit{new}(n) \cdot (\mu h.\, (\varepsilon + \mathit{new}(n) \cdot h)) \cdot \alpha(n)$ from Ex. 4.1. It is easy to check that p_5 is well-bound. Its may-bound names are:

$$bn^\diamond(p_5) = bn^\diamond(\mathit{new}(n)) \cup bn^\diamond(\mu h.\, (\varepsilon + \mathit{new}(n) \cdot h)) \cup bn^\diamond(\alpha(n)) = \{n\} \cup \emptyset = \{n\}$$

The bindification of p_5 is then computed as follows:

$$\begin{aligned} \mathit{bindify}(p_5) &= \nu n.\, \beta\big(\mathit{new}(n) \cdot (\mu h.\, (\varepsilon + \mathit{new}(n) \cdot h)) \cdot \alpha(n)\big) \\ &= \nu n.\, \big(\beta(\mathit{new}(n)) \cdot \mu h.\, \mathit{bindify}(\varepsilon + \mathit{new}(n) \cdot h) \cdot \beta(\alpha(n))\big) \\ &= \nu n.\, \mathit{new}(n) \cdot (\mu h.\, \nu n.\, \beta(\varepsilon + \mathit{new}(n) \cdot h)) \cdot \alpha(n) \\ &= \nu n.\, \mathit{new}(n) \cdot (\mu h.\, \nu n.\, (\varepsilon + \mathit{new}(n) \cdot h)) \cdot \alpha(n) \end{aligned}$$

We now state the correctness of bindification (Theorem 4.6). The "strong" semantics of $\mathit{bindify}(p)$ contains exactly the traces of the "weak" semantics of p.

Theorem 4.6 *For all closed, weakly bound processes p such that $wb(p)$, $[\![p]\!]^w_\emptyset(\emptyset)(\emptyset)$ is defined, and:*

$$[\![\mathit{bindify}(p)]\!]^s_\emptyset(\emptyset)(\emptyset) \quad = \quad \mathit{fst}([\![p]\!]^w_\emptyset(\emptyset)(\emptyset))$$

5 Equational theories and trace inclusion

In this section we provide strongly bound and weakly bound processes with an equational theory and a trace inclusion preorder. We shall state their correctness, i.e. the equational theory preserves the set of traces, while the preorder preserves their inclusion. Finally, we shall highlight some differences between the two calculi.

We first give in Def. 5.1 an equational theory of strongly bound processes.

Definition 5.1 An equational theory of strongly bound processes

The relation $=$ over strongly bound processes is the least congruence including α-conversion of names and variables such that:

$$P + P = P \qquad (P + P') + P'' = P + (P' + P'') \qquad P + P' = P' + P$$

$$(P \cdot P') \cdot P'' = P \cdot (P' \cdot P'') \qquad \varepsilon \cdot P = P = P \cdot \varepsilon$$

$$(P + P') \cdot P'' = P \cdot P'' + P' \cdot P'' \qquad P \cdot (P' + P'') = P \cdot P' + P \cdot P''$$

$$\mu h.\mu h'.P = \mu h'.\mu h.P \qquad \mu h.P = P\{\mu h.\, P/h\} \qquad \nu n.\varepsilon = \varepsilon$$

$$\nu n.\nu n'.P = \nu n'.\nu n.P \qquad \nu n.(P + P') = (\nu n.P) + P' \ \text{ if } n \notin \mathit{fn}(P')$$

$$\nu n.(P \cdot P') = P \cdot (\nu n.P') \ \text{ if } n \notin \mathit{fn}(P) \quad \nu n.(P \cdot P') = (\nu n.P) \cdot P' \ \text{ if } n \notin \mathit{fn}(P')$$

The operation $+$ is associative, commutative and idempotent; $\cdot$ is associative, has identity ε, and distributes over $+$. The binders μ and ν allow for α-conversion of bound names and variables, and can be rearranged. A μh can be introduced/eliminated when h does not occur free. A νn can be extruded when it does not bind a free occurrence of n. A $\mu h.\, P$ can be folded/unfolded as usual.

As expected, the equational theory above is not complete, e.g. $[\![P]\!]^s = [\![P']\!]^s$ does not imply $P = P'$. E.g., $\mu h.\, \alpha(r) \cdot h$ cannot be equated to $\mu h.\, \alpha(r) \cdot \alpha(r) \cdot h$, yet they have the same traces $\alpha(r)^* !$. However, the equational theory is sound w.r.t. our semantics, as established by the first item Theorem 5.3 below.

We then define a preorder $P \preceq Q$ betweeen strongly bound processes. The preorder $\preceq$ includes equivalence, and it is closed under contexts. Also, a strongly bound process P can be arbitrarily "weakened" to $P + Q$.

Definition 5.2 A trace inclusion preorder of strongly bound processes

The relation $\preceq$ over strongly bound processes is the least precongruence such that:

$$P \preceq Q \quad \text{if } P = Q \qquad\qquad P \preceq P + Q$$

The following theorem states that the equational theory $=$ and the preorder $\preceq$ agree with the semantics of strongly bound processes.

Theorem 5.3 *For all closed, strongly bound processes P and Q:*

- *if $P = Q$, then $[\![P]\!]^s_\emptyset = [\![Q]\!]^s_\emptyset$.*
- *if $P \preceq Q$ then $[\![P]\!]^s_\emptyset(\chi)(\mathcal{R}) \subseteq [\![Q]\!]^s_\emptyset(\chi)(\mathcal{R})$, for all $\mathcal{R}$ and χ.*

We now consider how to express an equational theory and a trace inclusion preorder for weak binders, in the same spirit of Def. 5.1 and Def. 5.2. In spite of their weaker structure, weakly bound processes still share many semantic-preserving equational properties with strongly bound processes, as summarized in Def. 5.4. Notably, the equations involving $+$ and $\cdot$ are identical with respect to Def. 5.1. The recursions μh can be rearranged, as before. Of course, here we do not have ν-binders, so the α-conversion of bound names can not be done, in general. As an important exception, we know that bound names inside a recursion can not escape, so their scope is completely known. In this case, we allow for α-conversion. Note that unfolding recursions is not allowed, otherwise we would have $\mu h.new(n) \cdot h \approx new(n) \cdot (\mu h.new(n) \cdot h) \approx new(n) \cdot new(n) \cdot (\mu h.new(n) \cdot h)$, so causing name confusion — indeed, the first two processes are well-bound, while the last one is not. As with strong binders, the equational theory below is not complete, yet it is sound w.r.t. the $[\![-]\!]^w$ semantics, as established by the first item of Theorem 5.7.

Definition 5.4 An equational theory of weakly bound processes

The relation $\approx$ over weakly bound processes is the least congruence including α-conversion of variables such that:

$$p + p \approx p \qquad (p + p') + p'' \approx p + (p' + p'') \qquad p + p' \approx p' + p \qquad \varepsilon \cdot p \approx p \approx p \cdot \varepsilon$$

$$(p \cdot p') \cdot p'' \approx p \cdot (p' \cdot p'') \quad (p + p') \cdot p'' \approx p \cdot p'' + p' \cdot p'' \quad p \cdot (p' + p'') \approx p \cdot p' + p \cdot p''$$

$$\mu h.\mu h'.p \approx \mu h'.\mu h.p \qquad \mu h.p \approx \mu h.(p\{m/n\}) \ \text{ if } n \in bn^\diamond(p) \text{ and } m \notin p$$

Example 5.5 The equational theories shown above offer an opportunity to compare strong ν-binders with weak *new* binders. Consider the following equation: $\mathit{new}(n) \cdot p + \mathit{new}(n) \cdot q \approx \mathit{new}(n) \cdot (p + q)$. This is a trivial fact, since it directly follows from the distributive law. Its equivalent for strongly bound processes, $(\nu n.\, P) + (\nu n.\, Q) = \nu n.\, (P + Q)$, appears instead to be non trivial. Indeed, although Def. 5.1 comprises all the classic equations for ν-binders, the mentioned equation can not be derived from them, since we can not identify the two binders. Yet, in most process algebras, we expect the equation to be sound w.r.t. any reasonable process equivalence relation. So, in this case weak binders offer a simpler view.

We shall now introduce a preorder $p \precsim_{\mathcal{N}} q$ on weakly bound processes. Here, we use a set of names $\mathcal{N}$ as an index to the preorder relation. This index is needed to avoid name confusion, as we shall see below. When $p \precsim_{\mathcal{N}} q$ holds, then the semantics of p is included in that of q (second item of Theorem. 5.7).

Definition 5.6 A trace inclusion preorder of weakly bound processes

The relation $\precsim_{\mathcal{N}}$ over weakly bound processes is the least preorder such that:

$$p \precsim_{\emptyset} q \quad \text{if } p \approx q \qquad p \precsim_{\emptyset} p + q \qquad p \precsim_{\mathcal{N} \cup \mathcal{N}'} p'' \quad \text{if } p \precsim_{\mathcal{N}} p' \text{ and } p' \precsim_{\mathcal{N}'} p''$$

$$\mathcal{C}(p) \precsim_{\mathcal{N}} \mathcal{C}(q) \quad \text{if } p \precsim_{\mathcal{N}} q \text{ and } \mathcal{N} \cap (\mathit{bn}^{\diamond}(\mathcal{C}) \cup \mathit{fn}(\mathcal{C})) = \emptyset$$

$$p\sigma\{\mu h.\, p/h\} \precsim_{\mathit{ran}(\sigma)} \mu h.\, p \quad \text{if } \mathit{ran}(\sigma) \cap \mathit{fn}(p) = \emptyset$$

where $\mathcal{C} = p \cdot \bullet \mid \bullet \cdot p \mid p + \bullet \mid \bullet + p$ is a *context*, $\sigma : \mathsf{Nam} \to \mathsf{Nam}$ is an injective function with $\mathit{dom}(\sigma) = \mathit{bn}^{\diamond}(p)$, and $p\sigma\{\mu h.\, p/h\}$ is capture-avoiding.

The preorder $\precsim_{\mathcal{N}}$ includes $\approx$-equivalence (Def. 5.4). A process p can be arbitrarily "weakened" to $p + q$. The relation is closed under contexts, provided that the names in $\mathcal{N}$ are disjoint from those in the context. Note that, because of this side condition, $\precsim_{\mathcal{N}}$ is not a precongruence, unlike $\preceq$ for strongly bound processes. Folding/unfolding is possible, but in a weaker form than in Def. 5.1. To avoid name confusion and preserve well-boundness, the unfolded names must be fresh. For instance, if $p = \mu h.\, \mathit{new}(n) \cdot \alpha(n) \cdot h$, then we shall have $\mathit{new}(n') \cdot \alpha(n') \cdot p \precsim_{\{n'\}} p$. The name n' in $\precsim_{\{n'\}}$ is needed to avoid name clashes. For instance, it prevents from using the previous unfolding in the context $\mathcal{C} = \bullet \cdot \alpha'(n')$, since the extruded $\mathit{new}(n')$ would bind the name n' in $\alpha'(n')$, as checked by the context rule above. The side condition on the rule for folding/unfolding is needed to ensure that all the processes smaller (w.r.t. $\precsim$) than a well-bound process are well-bound (Theorem 5.8). Omitting the disjointness condition between $\mathit{fn}(p)$ and the range of the substitution σ would lead to situations like $\alpha(n') \cdot \mathit{new}(n') \precsim_{\{n'\}} \mu h.\, \alpha(n') \cdot \mathit{new}(n)$, where the right-hand side is well-bound, while the left-hand side is not. Substitutions of names must be coherent with bindification, i.e. they must not affect names that would be put under a ν-binder by $\beta(-)$, e.g. $(\mathit{new}(n) \cdot \mu h.\, \mathit{new}(n))\{m/n\} = \mathit{new}(m) \cdot \mu h.\, \mathit{new}(n)$. Similarly, substitutions can trigger α-conversions to avoid name captures, e.g. $(\mu h.\, \mathit{new}(m) \cdot \alpha(n))\{m/n\} = \mu h.\, \mathit{new}(m') \cdot \alpha(m)$.

We now formally state that our syntactic preorder agrees with the semantics of weakly bound processes, as it yields trace inclusion. Note that trace inclusion requires the two semantics to be defined. Otherwise we have $\mathit{new}(n) \cdot \mu h.\, \alpha(n) \precsim_\emptyset \mathit{new}(n) \cdot \mu h.\, (\mathit{new}(n) + \varepsilon) \cdot \alpha(n)$: when the branch ε is chosen in the right-hand side, we find $\chi'(n) = \chi|_{\mathit{dom}(\chi)\setminus\{n\}}(n)$, so $\alpha(n)$ cannot be evaluated, and the whole semantics is undefined (while the semantics of the left-hand side is always defined). Note however that is q is well-bound, then also p is such (Theorem 5.8), and so by Theorem 4.6 both the semantics are defined.

Theorem 5.7 *For all closed, weakly bound processes p and q:*

- *if $p \approx q$, then $[\![p]\!]^w_\emptyset = [\![q]\!]^w_\emptyset$.*
- *if $p \precsim_\mathcal{N} q$ and, then $\mathit{fst}([\![p]\!]^w_\emptyset(\chi)(\mathcal{R})) \subseteq \mathit{fst}([\![q]\!]^w_\emptyset(\chi)(\mathcal{R}))$, for all $\mathcal{R}$ and χ such that $\mathit{dom}(\chi) \cap \mathcal{N} = \emptyset$ and both the semantics are defined.*

The projection *fst* in the statement above is necessary. Consider e.g. $p = \mathit{new}(n) \precsim_{\{n\}} \mu h.\, \mathit{new}(m) = q$. Here, the semantics of p and q agree on the η components, i.e. the truncations of $\mathit{new}(r)$ with $r \notin \mathcal{R}$, but p will augment χ with the new binding $\{r/n\}$, unlike q which does not affect χ.

The next theorem guarantees that *bindify* is well-defined, i.e. it maps $\approx$-equivalent weakly bound processes to $=$-equivalent strongly bound processes. Moreover, processes smaller (w.r.t. $\precsim_\mathcal{N}$) than well-bound processes are well-bound.

Theorem 5.8 *For all weakly bound processes p and q:*

- *if $p \approx q$, then $\mathit{wb}(p)$ if and only if $\mathit{wb}(q)$.*
- *if $p \approx q$ and $\mathit{wb}(p)$, then $\mathit{bindify}(p) = \mathit{bindify}(q)$.*
- *if $p \precsim_\mathcal{N} q$ and $\mathit{wb}(q)$, then $\mathit{wb}(p)$.*

6 Conclusions

We have investigated weak binders – a construct for fresh name generation – as an alternative for ν-binders in nominal calculi. Weak binders allow for a looser reasoning, while still admitting a trace-preserving translation into strong binders. However, this comes at a cost: often, useful properties, e.g. trace inclusion (Th. 5.7), require more side conditions to be checked for ensuring sanity. Also, α-conversion of names can only be applied inside μ-binders. This is possible through the last rule of the equational theory in Def. 5.4. An alternative would be to always consider weakly bound processes modulo α-conversion within the μ-binders, at the cost of making some proofs (e.g. those that do not depend on $\approx$) more complex. A further downside of weak binders is that compositionality is reduced, since e.g. $\mathit{wb}(p)$ and $\mathit{wb}(q)$ do not automatically imply $\mathit{wb}(p \cdot q)$ which – if needed – must be established by exploiting further assumptions on the names of p and q. Future work would address the use of weak binders in other process calculi. Indeed, we expect that weak binders enjoy stronger properties in calculi without sequential composition (e.g. CCS [11]). Moreover, studying some relaxed variants of well-boundness could improve the applicability of weak binders.

In our experiments with weak binders, we also found they sometimes lead to intricate proofs, since particular care must be exercised with corner cases. For instance, handling recursion in an operational semantics for weakly bound processes seems to be quite complex. Indeed, naïve unfolding causes name confusion, so one has to resort to either renaming all bound names so that they are indeed globally fresh, or to record the “call frames” (entering/leaving the body of a recursion) in a stack. Since we need to keep track of this, run-time configurations become more complex, and we found our operational semantics (not presented in this paper) to be too inconvenient to be used in proofs. Even when using the denotational semantics (Def. 3.6), we felt that writing inductive statements for weak binders required more trial-and-error steps, w.r.t. strong binders. However, in some occasions weak binders may become a more agile tool. For instance, they can be exploited to implement a type and effect inference algorithm for a calculus with side effects and explicit name binders (like [1]), on top of an existing algorithm for a calculus without binders. Each time a ν-binder is encountered, a fresh name is generated, similarly to fresh type variables in Hindley-Milner type inference. After solving the obtained type and effect constraints through unification, the resulting effect is bindified. Of course, this is not always possible, e.g. when the effect is not well-bound. Possible counter-measures consist in suitably extending let-polymorphism to ν-binders.

Related work. A number of formal techniques have been developed to handle binding and freshness of names. The permutation model of sets introduced by Fraenkel-Mostowski has led to an elegant and powerful mathematical theory of naming [8]. The key observation of this approach is that α-conversion, binding and freshness can be defined through name permutations (or swappings). For instance, the freshness axiom for a name of a computational entity (i.e. an object, a process, a context, *etc.*) is expressed by saying that the fresh name does not belong to the support of the computational entity. Notably, in the permutation model the support of computational entities is *finite*. This mathematical theory has been used to model early and late semantics of the π-calculus [9]. Also, it has driven the design of a functional language, FreshML [20], which includes primitive mechanisms for handling fresh bindable names. In FreshML freshness is managed by a *gensym()* primitive to dynamically generate names, and a primitive for permuting names. Our notion of weakly bound processes exploits the *gensym()* primitive without resorting to α-conversion. Indeed, the bindify trasformation singles out the names in the finite support of a weakly bound process. A monadic denotational semantics for FreshML has been used to handle freshness through a continuation monad on FM-sets [19]. This semantics allows for translating the usual domain-theoretic results in the context of FM-sets, and to use them to prove freshness-related properties. There is also a cost associated to α-converting names [7,15] which could be reduced e.g. by compiling strong binders into weak binders.

The $\lambda\nu$-calculus presented in [16] extends the pure λ-calculus with names. In contrast to λ-bound variables, nothing can be substituted for a name, yet names can be tested for equality. Reduction in $\lambda\nu$ is confluent, and it allows for deterministic evaluation. Furthermore, all the observational equivalences that hold in the pure λ-calculus still hold in $\lambda\nu$. This has the practical consequence that all the equational techniques for transforming and verifying pure functional programs are also applicable to programs with local names. Nominal techniques have been implicitly used for reasoning about the semantics of functional languages with local state in [17], to prove when two functional programs are equivalent in every evaluation context.

Binding and freshness of names have been a main concern in process calculi. History-Dependent automata [13,14] provide an automata-based model where states are equipped with name permutations to manage freshness and garbage collections of names. They automatically manage the creation and deallocation of names, while allowing for a compact representation of the system behaviour, by collapsing the states that only differ for the renaming of local names. The π-calculus is extended in [4] with an operational model where names are localized to their owners; each sequential process has its logical space on names and a local manager generates fresh names whenever necessary.

References

[1] M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Types and effects for resource usage analysis. In *Foundations of Software Science and Computation Structures (FOSSACS)*, 2007.

[2] M. Bartoletti, P. Degano, G. L. Ferrari, and R. Zunino. Hard life with weak binders. Technical Report TR-08-13, Dip. Informatica, Univ. Pisa, 2008.

[3] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37:77–121, 1985.

[4] C. Bodei, P. Degano, and C. Priami. Names of the π calculus agents handled locally. *Theoretical Computer Science*, 253(2):155–184, 2001.

[5] M. Bravetti and G. Zavattaro. Towards a unifying theory for choreography conformance and contract compliance. In *Software Composition*, 2007.

[6] G. Castagna, N. Gesbert, and L. Padovani. A theory of contracts for web services. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 2008.

[7] M. Fernández, I. Mackie, and F.-R. Sinot. Closed reduction: explicit substitutions without α-conversion. *Math. Structures Comput. Sci.*, 15(2):343–381, 2005.

[8] M. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

[9] M. J. Gabbay. The pi-calculus in FM. In F. Kamareddine, editor, *Thirty-five years of Automath*, volume 28 of *Applied Logic Series*, pages 247–269. Kluwer, 2003.

[10] A. Igarashi and N. Kobayashi. Type reconstruction for linear -calculus with i/o subtyping. *Inf. Comput.*, 161(1):1–44, 2000.

[11] R. Milner. *Communication and concurrency*. Prentice-Hall, Inc., 1989.

[12] R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, I and II. *Information and Computation*, 100(1):1–40,41–77, September 1992.

[13] U. Montanari and M. Pistore. An introduction to history dependent automata. *Electr. Notes Theor. Comput. Sci.*, 10, 1997.

[14] U. Montanari and M. Pistore. Structured coalgebras and minimal hd-automata for the π-calculus. *Theor. Comput. Sci.*, 340(3):539–576, 2005.

[15] M. Norrish and R. Vestergaard. Proof pearl: De bruijn terms really do work. In *TPHOLs*, pages 207–222, 2007.

[16] M. Odersky. A functional theory of local names. In *ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, 1994.

[17] A. M. Pitts and I. D. B. Stark. Operational reasoning for functions with local state. *Higher order operational techniques in semantics*, pages 227–274, 1998.

[18] D. Sangiorgi and D. Walker. *The π-Calculus: a Theory of Mobile Processes.* Cambridge University Press, 2002.

[19] M. R. Shinwell and A. M. Pitts. On a monadic semantics for freshness. *Theoretical Computer Science*, 342(1):28–55, 2005.

[20] M. R. Shinwell, A. M. Pitts, and M. Gabbay. FreshML: programming with binders made simple. In *International Conference on Functional Programming (ICFP)*, 2003.

A Appendix: strongly bound processes

This appendix contains a number of intermediate definitions and lemmata that are necessary to prove Theorem 2.6. All the detailed proofs are in [2].

Remark A.1 To simplify the proof of full abstraction, hereafter we shall extend strongly bound processes with the process $!$, that models a non-terminated computation. The labelled transition system in Def. 2.2 is enriched with the following rule, that allows for observing the finite prefixes of non-terminating computations:

$$P, \mathcal{R} \xrightarrow{!} !, \mathcal{R}$$

The trace semantics $[\![P]\!]^{op}(\mathcal{R})$ is then defined as follows:

$$[\![P]\!]^{op}(\mathcal{R}) \;=\; \{\, \eta \mid P, \mathcal{R} \xrightarrow{\eta} \varepsilon, \mathcal{R}' \,\} \cup \{\, \eta ! \mid P, \mathcal{R} \xrightarrow{\eta !} Q, \mathcal{R}' \text{ and } ! \notin \eta \,\}$$

Lemma A.2 *For all strongly bound processes P and for all $\mathcal{R}$, if $\eta \in [\![P]\!]^{op}(\mathcal{R})$ and $! \in \eta$, then $\eta = \eta' !$ for some η' such that $! \notin \eta'$.*

We now define the function $\mathsf{R}(P)$, that computes the set of resources mentioned in P and reachable in some computations of P. To do that, $\mathsf{R}(P)$ performs a sort of reachability analysis, e.g. $\mathsf{R}(\alpha(r) \cdot (\mu h.\, h) \cdot \alpha(r'))$ contains r but not r', since the non-terminaing loop $\mu h.\, h$ makes $\alpha(r')$ unreachable. Having $\downarrow \in \mathsf{R}(P)$, means that P allows for some terminating computations. The function $\mathsf{T}(P)$ defined below exploits this fact to characterize the processes that may terminate.

Definition A.3 For all strongly bound processes P and for all functions Θ from variables h to $\mathsf{Res} \cup \{\downarrow\}$, we define $\mathsf{R}_\Theta(P)$ inductively as follows:

$$\mathsf{R}_\Theta(\varepsilon) = \{\downarrow\} \qquad \mathsf{R}_\Theta(!) = \emptyset \qquad \mathsf{R}_\Theta(h) = \Theta(h) \qquad \mathsf{R}_\Theta(\nu n.\, P) = \mathsf{R}_\Theta(P)$$

$$\mathsf{R}_\Theta(\alpha(\rho)) = \begin{cases} \{r, \downarrow\} & \text{if } \rho = r \\ \{\downarrow\} & \text{otherwise} \end{cases} \qquad \mathsf{R}_\Theta(P \cdot Q) = \begin{cases} (\mathsf{R}_\Theta(P) \setminus \{\downarrow\}) \cup \mathsf{R}_\Theta(Q) & \text{if } \downarrow \in \mathsf{R}_\Theta(P) \\ \mathsf{R}_\Theta(P) & \text{otherwise} \end{cases}$$

$$\mathsf{R}_\Theta(P + Q) = \mathsf{R}_\Theta(P) \cup \mathsf{R}_\Theta(Q) \qquad \mathsf{R}_\Theta(\mu h.\, P) = \mathsf{R}_{\Theta\{\{\downarrow\} \cap \mathsf{R}_{\Theta\{\emptyset/h\}}(P)/h\}}(P)$$

Also, we define $\mathsf{T}_\Theta(P)$ as follows:

$$\mathsf{T}_\Theta(P) \quad = \quad \{\downarrow\} \cap \mathsf{R}_\Theta(P)$$

Example A.4 Let $P = \mu h.\, \nu n.\, h \cdot \alpha(r) + \alpha(n)$. Since $\mathsf{T}_{\{\emptyset/h\}}(\nu n.\, h \cdot \alpha(r) + \alpha(n)) = \mathsf{T}_{\{\emptyset/h\}}(h \cdot \alpha(r)) \cup \mathsf{T}_{\{\emptyset/h\}}(\alpha(n)) = \{\downarrow\}$ and $\downarrow \in \mathsf{R}_{\{\{\downarrow\}/h\}}(h) = \{\downarrow\}$, we have that:

$$\begin{aligned} \mathsf{R}_\emptyset(P) &= \mathsf{R}_{\{\mathsf{T}_{\{\emptyset/h\}}(\nu n.\, h \cdot \alpha + \alpha(n))/h\}}(\nu n.\, h \cdot \alpha(r) + \alpha(n)) \\ &= \mathsf{R}_{\{\{\downarrow\}/h\}}(\nu n.\, h \cdot \alpha(r) + \alpha(n)) \\ &= \mathsf{R}_{\{\{\downarrow\}/h\}}(h \cdot \alpha(r)) \cup \mathsf{R}_{\{\{\downarrow\}/h\}}(\alpha(n)) \\ &= \mathsf{R}_{\{\{\downarrow\}/h\}}(h) \cup \mathsf{R}_{\{\{\downarrow\}/h\}}(\alpha(r)) \cup \{\downarrow\} \\ &= \{\downarrow, r\} \end{aligned}$$

Lemma A.5 *For all P and Θ, we have that $\mathsf{T}_\Theta(P)$ equals to:*

$\{\downarrow\}$	*if* $P = \varepsilon$ *or* $P = \alpha(\rho)$
$\{\downarrow\} \cap \Theta(h)$	*if* $P = h$
$\emptyset$	*if* $P = !$
$\mathsf{T}_\Theta(Q)$	*if* $P = \nu n.\, Q$
$\mathsf{T}_\Theta(P_0) \cap \mathsf{T}_\Theta(P_1)$	*if* $P = P_0 \cdot P_1$
$\mathsf{T}_\Theta(P_0) \cup \mathsf{T}_\Theta(P_1)$	*if* $P = P_0 + P_1$
$\mathsf{T}_{\Theta\{\emptyset/h\}}(Q)$	*if* $P = \mu h.\, Q$

The following lemma proves some basic facts about R and T.

Lemma A.6 *For all P, h, $\mathcal{R}$, $\mathcal{R}'$, Θ, and for all $\chi : \mathsf{Nam} \to \mathsf{Res}$:*

(A.6a) $\mathcal{R} \subseteq \mathcal{R}' \implies \mathsf{T}_{\Theta\{\mathcal{R}/h\}}(P) \subseteq \mathsf{T}_{\Theta\{\mathcal{R}'/h\}}(P)$

(A.6b) $\mathsf{T}_\Theta(P) = \mathsf{T}_{\Theta\{\mathsf{T}_\Theta(h)/h\}}(P)$

(A.6c) $\mathsf{T}_{\Theta\{\mathsf{T}_{\Theta\{\emptyset/h\}}(P)/h\}}(P) = \mathsf{T}_{\Theta\{\emptyset/h\}}(P)$

(A.6d) $\mathcal{R} \subseteq \mathcal{R}' \implies \mathsf{R}_{\Theta\{\mathcal{R}/h\}}(P) \subseteq \mathsf{R}_{\Theta\{\mathcal{R}'/h\}}(P)$

(A.6e) $\mathsf{R}_\Theta(P) \subseteq \mathsf{R}_\Theta(P\chi) \subseteq \mathsf{R}_\Theta(P) \cup \mathit{ran}(\chi)$

(A.6f) $\mathsf{R}_\Theta(P) \subseteq \mathsf{R}_{\Theta\{\mathsf{T}_\Theta(h)/h\}}(P) \cup \Theta(h)$

(A.6g) $\mathsf{R}_\Theta(\mu h.\, P) = \mathsf{R}_{\Theta\{\mathsf{R}_\Theta(\mu h.\, P)/h\}}(P)$

The following lemma states two basic properties of the trace semantics. The first item guarantees that the freshly created resources are disjoint from $\mathcal{R}$. The second item shows that the trace semantics is antimonotonic w.r.t. R.

Lemma A.7 *For all closed strongly bound processes P:*

(A.7a) $\eta \in [\![P]\!]^{op}(\mathcal{R}) \implies (\mathcal{R} \setminus \mathsf{R}_\emptyset(P)) \cap \mathsf{R}(\eta) = \emptyset$

(A.7b) $\mathcal{R} \subseteq \mathcal{R}' \implies [\![P]\!]^{op}(\mathcal{R}) \supseteq [\![P]\!]^{op}(\mathcal{R}')$

We study below how the trace semantics is affected by adding/removing resources from $\mathcal{R}$, and by substituting a resource for a name.

Lemma A.8 *For all strongly bound processes P and P':*

(A.8a) $P, \mathcal{R} \xrightarrow{\eta} P', \mathcal{R}' \implies P, \mathcal{R} \setminus \{r\} \xrightarrow{\eta} P', \mathcal{R}' \setminus \{r\}$ *if* $r \in \mathcal{R} \setminus \mathsf{R}(\eta)$

(A.8b) $P, \mathcal{R} \xrightarrow{\eta} P', \mathcal{R}' \implies P, \mathcal{R} \cup \{r\} \xrightarrow{\eta} P', \mathcal{R}' \cup \{r\}$ *if* $r \notin \mathcal{R}'$

(A.8c) $P\{r/n\}, \mathcal{R} \xrightarrow{\eta} \varepsilon, \mathcal{R}' \implies P\{r'/n\}, \mathcal{R} \xrightarrow{\eta} \varepsilon, \mathcal{R}'$ *if* $r \notin \mathsf{R}(\eta)$

(A.8d) $P\{r/n\}, \mathcal{R} \xrightarrow{\eta} \varepsilon, \mathcal{R}' \implies P\{r'/n\}, \mathcal{R}\{r'/r\} \xrightarrow{\eta} \varepsilon, \mathcal{R}'\{r'/r\}$ *if* $\begin{array}{l} r \notin \mathsf{R}(\eta), \\ r' \notin \mathcal{R}' \end{array}$

The following lemma relates the labelled transition relation with the trace semantics.

Lemma A.9 *For all strongly bound processes P, P':*

$$P, \mathcal{R} \xrightarrow{a} P', \mathcal{R}' \implies [\![P]\!]^{op}(\mathcal{R}) \supseteq a \odot [\![P']\!]^{op}(\mathcal{R}')$$

We now introduce a further denotational semantics $[\![-]\!]^{sub}$ of strongly bound processes. The main difference from $[\![-]\!]^{s}$ of Def. 2.5 is the way the two semantics handle the case $\nu n.\, P$. In $[\![\nu n.\, P]\!]^{s}$, the freshly created resource r is used to extend the environment χ with the binding $\{r/n\}$. Instead, in $[\![\nu n.\, P]\!]^{sub}$ the substitution $\{r/n\}$ is performed directly on P — hence the environment χ can be omitted. Since substitutions are also used by $[\![-]\!]^{op}$, in the proof of full abstraction we shall conveniently use $[\![-]\!]^{sub}$ as a bridge between $[\![-]\!]^{op}$ and $[\![-]\!]^{s}$.

Definition A.10 Substitution semantics of strongly bound processes

The substitution semantics $[\![P]\!]^{sub}_{\theta}$ of a strongly bound process P such that $fn(P) = \emptyset$ is defined below. Let D_0 be the following cpo of sets of traces ordered by set inclusion: $D_0 = \{\, X \subseteq (\mathsf{Ev} \cup \{\,!\,\})^* \mid\, ! \in X \,\wedge\, \forall \eta \in X : \eta\,! \in X \,\}$. The set $\{\,!\,\}$ is the bottom element of D_0. Then, let $D_{sub} = \mathcal{P}_{\mathit{fin}}(\mathsf{Res}) \to D_0$ be the cpo of functions from the finite subsets of Res to D_0. Note that the bottom element $\bot$ of D_{sub} is $\lambda \mathcal{R}.\, \{\,!\,\}$. Then, the semantics of P (parametrized by θ) is a function in D_{sub}. The parameter θ is a function that maps each variable h to a function in D_{sub}. We require $dom(\theta) \supseteq fv(P)$. The semantics $[\![P]\!]^{sub}_{\theta}$ is inductively defined through the following equations.

$$
\begin{aligned}
[\![\varepsilon]\!]^{sub}_{\theta} &= \lambda \mathcal{R}.\ \{\,!, \varepsilon\} \\
[\![\alpha(\rho)]\!]^{sub}_{\theta} &= \lambda \mathcal{R}.\ \{\,!, \alpha(\rho), \alpha(\rho)\,!\,\} \quad \text{if } \rho \in \mathsf{Res} \\
[\![\nu n.\, P]\!]^{sub}_{\theta} &= \lambda \mathcal{R}.\ \textstyle\bigcup_{r \notin \mathcal{R}}\ [\![P\{r/n\}]\!]^{sub}_{\theta}(\mathcal{R} \cup \{r\}) \\
[\![P \cdot P']\!]^{sub}_{\theta} &= [\![P]\!]^{sub}_{\theta} \boxdot [\![P']\!]^{sub}_{\theta} \\
[\![P + P']\!]^{sub}_{\theta} &= [\![P]\!]^{sub}_{\theta} \sqcup [\![P']\!]^{sub}_{\theta} \\
[\![!]\!]^{sub}_{\theta} &= \bot \\
[\![h]\!]^{sub}_{\theta} &= \theta(h) \\
[\![\mu h.P]\!]^{sub}_{\theta} &= \bigsqcup_{i \geq 0} f^i(\bot) \quad \text{where} \quad f(Y) = [\![P]\!]^{sub}_{\theta\{Y/h\}}
\end{aligned}
$$

We first check that the above semantics is well-defined. Lemma A.11 proves that the image of the semantics function is indeed D_0. Lemma A.14 guarantees that the least upper bound in the last equation exists (since f is monotone). Also, since f is continuous, by the Knaster-Tarski theorem the semantics of $\mu h.\, P$ is the least fixed point of f.

Lemma A.11 *For all strongly bound processes P, for all θ and $\mathcal{R}$, $! \in [\![P]\!]^{sub}_{\theta}(\mathcal{R})$.*

Lemma A.12 *The structure $(D_{sub}, \sqcup, \boxdot, id_{\sqcup}, id_{\boxdot})$, where $id_{\sqcup} = \bot$ and $id_{\boxdot} = \lambda \mathcal{R}.\{\,!, \varepsilon\}$ is a semi-ring.*

Lemma A.13 *Let $\{Y_i\}_i$ and $\{Z_i\}_i$ be subsets of D_{sub}. Then:*

$$
\bigsqcup_i (Y_i \boxdot Z_i) \;=\; \Big(\bigsqcup_i Y_i\Big) \boxdot \Big(\bigsqcup_i Z_i\Big)
$$

Lemma A.14 *For all strongly bound processes P such that $fn(P) = \emptyset$, and for all θ such that $dom(\theta) \cup \{h\} \supseteq fv(P)$, the function $f_P(Y) = [\![P]\!]^{sub}_{\theta\{Y/h\}}$ is continuous.*

Lemma A.15 *We say $Y \in D_{sub}$* anti-monotone *when $\mathcal{R} \subseteq \mathcal{R}'$ implies $Y(\mathcal{R}) \supseteq Y(\mathcal{R}')$ for all $\mathcal{R}, \mathcal{R}'$. For all P and anti-monotone θ, $[\![P]\!]^{sub}_{\theta}$ is anti-monotone.*

Definition A.16 For all $Y \in D_{sub}$, we define $\mathsf{R}(Y)$ and $\mathsf{T}(Y)$ as follows:

$$\mathsf{R}(Y) = \bigcap_{\mathcal{R}} \mathsf{R}(Y(\mathcal{R})) \qquad\qquad \mathsf{T}(Y) = \{\downarrow\} \cap \mathsf{R}(Y)$$

Here we state that T correctly characterizes termination.

Lemma A.17 *For all $Y \in D_{sub}$:*

$$\mathsf{T}(Y) = \begin{cases} \emptyset & \text{if } \forall \mathcal{R} : \eta \in Y(\mathcal{R}) \implies ! \in \eta \\ \{\downarrow\} & \text{otherwise} \end{cases}$$

Remark A.18 The function R is not continuous. For instance, take a set of distinct resources $\{r_i\}_{i\in\omega}$. Let $\{Y_i\}_{i\in\omega}$ a family of functions in D_{sub}, defined as $Y_i = \lambda\mathcal{R}.\{\, \alpha(r_k) \mid k > |\mathcal{R}| - i \,\}$ where $|\mathcal{R}|$ denotes the cardinality of the finite set $\mathcal{R}$. This family is actually an ω-chain, since $k > |\mathcal{R}| - i$ implies $k > |\mathcal{R}| - (i+1)$. However, it is easy to check that $\mathsf{R}(\bigsqcup_i Y_i) = \{\, r_k \mid k \in \omega \,\}$ while $\bigcup_i \mathsf{R}(Y_i) = \emptyset$.

The following lemma provides an alternative definition of $\mathsf{R}(Y)$, in terms of $\mathsf{R}(\eta)$.

Lemma A.19 *For all monotone non-increasing Y, and for all ω-chain $\{\mathcal{R}_i\}_i$ such that $\bigcup_i \mathcal{R}_i = \mathsf{Res}$:*

$$\mathsf{R}(Y) \;=\; \bigcap_i \bigcup \{\, \mathsf{R}(\eta) \mid \eta \in Y(\mathcal{R}_i) \,\}$$

Termination is not affected by the particular choice of $\mathcal{R}$.

Lemma A.20 *For all $Y \in D_{sub}$ and for all $\mathcal{R}$, $\mathsf{T}(Y) = \mathsf{T}(Y(\mathcal{R}))$.*

We now state the correspondence between the syntactic and semantic variants of R.

Lemma A.21 *For all strongly bound processes P with $fn(P) = \emptyset$:*

$$\mathsf{R}([\![P]\!]^{sub}_{\theta}) = \mathsf{R}_{\mathsf{R}(\theta)}(P)$$

for all θ such that: $\forall \mathcal{R}\; \forall h \in fv(P) : \mathsf{R}(\theta(h)(\mathcal{R})) \subseteq \mathsf{R}(\theta(h)) \cup (\mathsf{Res} \setminus \mathcal{R})$.

This is the analogous of Lemma A.7a, adapted to the substitution semantics.

Lemma A.22 *For all P, θ, $\mathcal{R}$, we have that:*

$$\eta \in [\![P]\!]^{sub}_{\theta}(\mathcal{R}) \implies (\mathcal{R} \setminus \mathsf{R}_{\mathsf{R}(\theta)}(P)) \cap \mathsf{R}(\eta) = \emptyset$$

We now state that the substitution semantics is included in the trace semantics.

Lemma A.23 *Let P be a strongly bound processes with $fn(P) = \emptyset$, let θ be a function such that $dom(\theta) = \{h_1, \ldots, h_k\} \supseteq fv(P)$ and $\theta(h_i)(\bar{\mathcal{R}}) \subseteq [\![P_i]\!]^{op}(\bar{\mathcal{R}})$ for all*

$i \in 1..k$ *and for all* $\bar{\mathcal{R}} \supseteq \mathcal{R}$, *and let* $\mathcal{R} \supseteq \mathsf{R}_{\mathsf{R}(\theta)}(P)$ *be a finite set of resources. Then:*

$$[\![P]\!]^{sub}_{\theta}(\mathcal{R}) \subseteq [\![P\{P_1/h_1, \cdots, P_k/h_k\}]\!]^{op}(\mathcal{R})$$

The next two lemmata allow for substitution of h-variables and for unfolding of recursions in strongly bound processes.

Lemma A.24 *For all strongly bound processes* P, P' *with* $fn(P) = fn(P') = \emptyset$, *for all* θ *such that* $dom(\theta) \supseteq fv(P) \cup fv(P')$, *and for all* $h \notin fv(P')$*:*

$$[\![P\{P'/h\}]\!]^{sub}_{\theta} = [\![P]\!]^{sub}_{\theta\{[\![P']\!]^{sub}_{\theta}/h\}}$$

Lemma A.25 (Unfolding) *For all strongly bound processes* P, *and for all* θ*:*

$$[\![\mu h.\, P]\!]^{sub}_{\theta} \;=\; [\![P]\!]^{sub}_{\theta\{[\![\mu h.\, P]\!]^{sub}_{\theta}/h\}}$$

We now state the opposite direction of Lemma A.23, i.e. that the labelled transition relation is included in the substitution semantics.

Lemma A.26 *Let* P, P' *be closed, strongly bound process, let* $\mathcal{R}, \mathcal{R}'$ *be finite sets of resources, and let* $\theta_0 = \emptyset$. *Then:*

$$P,\, \mathcal{R} \xrightarrow{a} P',\, \mathcal{R}' \implies [\![P]\!]^{sub}_{\theta_0}(\mathcal{R}) \supseteq a \odot [\![P']\!]^{sub}_{\theta_0}(\mathcal{R}')$$

The next two lemmata relate the operational, the denotational and the substitution semantics. Summed up, they imply the full abstraction result (Theorem 2.6).

Lemma A.27 *Let* P *be a closed, strongly bound process, and let* $\mathcal{R}$ *be a finite set of resources. Then:*

$$[\![P]\!]^{op}(\mathcal{R}) \;=\; [\![P]\!]^{sub}_{\emptyset}(\mathcal{R})$$

Lemma A.28 *For all strongly bound* P, *for all* $\mathcal{R}$, θ *and* χ *such that* $fn(P\chi) = \emptyset$*:*

$$[\![P\chi]\!]^{sub}_{\theta}(\mathcal{R}) = [\![P]\!]^{s}_{\theta}(\chi)(\mathcal{R})$$

B Appendix: weakly bound processes

In this Appendix we prove some intermediate results about weakly bound processes. These will be exploited in App. C and in App. D to show the correctness of the bindify transformation and of the trace inclusion preorder, respectively.

Lemma B.1 *For all weakly bound processes* p*:*

(B.1a) $fn(p) \cap bn^{\Box}(p) = \emptyset$

(B.1b) $Fn(p) \supseteq fn(p) \cup bn^{\diamond}(p)$

The following lemma ensures that the condition $\mathcal{R} \supseteq ran(\chi)$ is always respected by the intermediate results of Def. 3.6. Therefore, in what follows we shall always omit to explicitly check this condition.

Lemma B.2 *For all weakly bound processes p, and for all $\mathcal{R}$, χ:*

$$(\eta, \chi') \in [\![p]\!]^w_\theta(\chi)(\mathcal{R}) \land \mathcal{R} \supseteq \mathit{ran}(\chi) \implies \mathcal{R} \cup \mathsf{R}(\eta) \supseteq \mathit{ran}(\chi')$$

The weakly-bound analogous of Def. A.16 and Lemma A.17 follow.

Definition B.3 For all $Y \in D_w$, we define $\mathsf{T}(Y)$ and $\mathsf{R}(Y)$ as follows:

$$\mathsf{R}(Y) = \bigcap_{\mathcal{R},\chi} \mathsf{R}(\mathit{fst}(Y(\chi)(\mathcal{R}))) \qquad\qquad \mathsf{T}(Y) = \{\downarrow\} \cap \mathsf{R}(Y)$$

Lemma B.4 *For all $Y \in D_w$:*

$$\mathsf{T}(Y) = \begin{cases} \emptyset & \text{if } \forall \mathcal{R}, \chi.\, (\eta, \chi') \in Y(\chi)(\mathcal{R}) \implies ! \in \eta \\ \{\downarrow\} & \text{otherwise} \end{cases}$$

The following definition enumerates some sanity conditions that any reasonable semantics of weakly bound process must adhere to. Lemma B.6 below ensures that the semantics of Def. 3.6 fulfills all these conditions.

Definition B.5 We say that $Y \in D_w$ is *sane* if and only if, for all $\mathcal{R}, \chi$:

(B.5a) $Y(\chi)(\mathcal{R}) \neq \emptyset$

(B.5b) $Y(\chi)(\mathcal{R}) \supseteq Y(\chi)(\mathcal{R}')$ if $\mathcal{R} \subseteq \mathcal{R}'$

(B.5c) $(\eta, \chi') \in Y(\bar{\chi})(\mathcal{R}) \implies \chi' \supseteq \bar{\chi}$

(B.5d) $(\eta, \chi'\{r/n\}) \in Y(\chi\{r/n\})(\mathcal{R}) \;\land\; r \notin \eta \implies (\eta, \chi') \in Y(\chi)(\mathcal{R})$

(B.5e) if $r \notin \mathsf{R}(\eta) \land n \notin \mathit{dom}(\chi')$:

$(\eta, \chi') \in Y(\chi)(\mathcal{R}) \implies (\eta, \chi'\{r/n\}) \in Y(\chi\{r/n\})(\mathcal{R} \cup \{r\})$

(B.5f) $(\eta, \chi') \in Y(\chi)(\mathcal{R}) \implies \mathsf{R}(Y) \subseteq \mathsf{R}(\eta) \subseteq \mathcal{R}(Y) \cup \mathit{ran}(\chi) \cup (\mathsf{Res} \setminus \mathcal{R})$

We say that $Z \in D_{\mathit{sub}}$ is *sane* if $\lambda\mathcal{R}, \chi.\mathit{set}_\chi Z(\mathcal{R})$ is sane.
We say that θ is *sane* if $\theta(h)$ is sane for all $h \in \mathit{dom}(\theta)$.

Lemma B.6 *For all weakly bound processes p and sane θ:*

(B.6a) $[\![p]\!]^w_\theta$ *is sane*

(B.6b) $f(Z) = \lambda\bar{\mathcal{R}}.\, \mathit{fst}([\![p]\!]^w_{\theta\{Z/h\}}(\chi)(\bar{\mathcal{R}}))$ *is sane, for all χ and sane $Z \in D_{\mathit{sub}}$*

Lemma B.7 *For all weakly bound processes p and sane θ:*

$$[\![p]\!]^w_\theta(\chi)(\mathcal{R}) \subseteq \bigcup_{r \notin \mathcal{R}} [\![p]\!]^w_\theta(\chi)(\mathcal{R} \cup \{r\})$$

The following lemma is the weakly-bound analogous of Lemma A.21.

Lemma B.8 *For all weakly bound processes p and sane θ, $\mathsf{R}([\![p]\!]^w_\theta) = \mathsf{R}_{\mathsf{R}(\theta)}(p)$.*

C Appendix: correctness of bindification

In this Appendix we shall establish in Theorem 4.6 the correctness of bindification, i.e. that $[\![p]\!]^w_\emptyset = [\![\mathit{bindify}(p)]\!]^s_\emptyset$ for each well-bound p. Some intermediate results and definitions precede the proof of the main theorem.

We start by defining the action of filtering a name n from a pair (η, χ). If $\chi(n)$ does not occur in η, then the filter removes from χ the binding for n (Def. C.1). This is used in the following Def. C.2. A semantic function Y is "anticipating on n" when $Y(\chi)(R)$ can be computed by joining the possible instantiations $Y(\chi\{r/n\})(R \cup \{r\})$ for all $r \notin \mathcal{R}$ – modulo filtering of n. This is the weakly-bound analogous of lifting a νn binder to the top-level, as done in strongly-bound processes. Lemma C.3 below states that the semantics of weakly bound processes in Def. 3.6 is anticipating, for all names n.

Definition C.1 For all η, χ and $n \in \mathsf{Nam}$, we define:

$$\mathit{flt}_n((\eta,\chi)) = \begin{cases} (\eta,\chi) & \text{if } \chi(n) \in \mathsf{R}(\eta) \\ (\eta, \chi|_{\mathit{dom}(\chi)\setminus\{n\}}) & \text{otherwise} \end{cases}$$

Definition C.2 We say $Y \in D_w$ is *anticipating on* n if, for all χ such that $n \notin \mathit{dom}(\chi)$ and $\mathcal{R} \supseteq \mathsf{R}(Y)$ such that $Y(\chi)(\mathcal{R})$ is defined:

$$Y(\chi)(\mathcal{R}) \quad = \quad \mathit{flt}_n(\bigcup_{r \notin \mathcal{R}} Y(\chi\{r/n\})(\mathcal{R} \cup \{r\}))$$

We say that $Y \in D_{sub}$ is *anticipating* if $\lambda\mathcal{R}, \chi.\, \mathit{set}_\chi Y(\mathcal{R})$ is anticipating on Nam.

Lemma C.3 *For all weakly bound processes p, for all anticipating θ such that $\mathit{dom}(\theta) \supseteq \mathit{fv}(p)$, and for all $n \notin \mathit{fn}(p)$, $[\![p]\!]^w_\theta$ is anticipating on n.*

The following lemma relates the free names and the R-function of (well-bound) weakly bound processes with their strongly-bound counterparts. Also, the third item guarantees that the χ component is preserved when it already contains the bindings for all the may-bound names of the process.

Lemma C.4 *For all weakly bound processes p such that $\mathit{wb}(p)$:*

(C.4a) $\mathit{fn}(p) = \mathit{Fn}(\mathit{bindify}(p))$

(C.4b) $\mathsf{R}_\Theta(p) = \mathsf{R}_\Theta(\mathit{bindify}(p))$

(C.4c) $(\eta, \chi') \in [\![p]\!]^w_\theta(\chi)(\mathcal{R}) \ \wedge\ \mathit{dom}(\chi) \supseteq \mathit{bn}^\diamond(p) \implies \chi = \chi'$

The following lemma contains the inductive statements about the bindify transformation that will allow us to prove in Theorem 4.6 its correctness.

Lemma C.5 *For all weakly bound processes p such that $\mathit{wb}(p)$, and for all $\mathcal{R}, \chi, \theta$ such that $\mathit{dom}(\chi) \supseteq \mathit{fn}(p)$, $\mathit{dom}(\theta) \supseteq \mathit{fv}(p)$, and $\mathcal{R} \supseteq \mathsf{R}_{\mathsf{R}(\theta)}(p)$:*

(C.5a) $[\![p]\!]^w_\theta(\mathcal{R})(\chi)$ *is defined*

(C.5b) $[\![\mathit{bindify}(p)]\!]^s_\theta(\chi)(\mathcal{R}) = \mathit{fst}([\![p]\!]^w_\theta(\chi)(\mathcal{R}))$ *if* $\mathit{dom}(\chi) \cap \mathit{bn}^\diamond(p) = \emptyset$

(C.5c) $[\![\beta(p)]\!]^s_\theta(\chi)(\mathcal{R}) = \mathit{fst}([\![p]\!]^w_\theta(\chi)(\mathcal{R}))$ *if* $\mathit{dom}(\chi) \supseteq \mathit{bn}^\diamond(p)$

The following Theorem states the correctness of the bindify transformation. Its proof is direct from Lemma C.5b, the hypotheses of which are trivially satisfied.

Theorem 4.6. For all closed, weakly bound processes p such that $\mathit{wb}(p)$, $[\![p]\!]^w_\emptyset(\emptyset)(\emptyset)$ is defined, and $[\![\mathit{bindify}(p)]\!]^s_\emptyset(\emptyset)(\emptyset) = \mathit{fst}([\![p]\!]^w_\emptyset(\emptyset)(\emptyset))$.

D Appendix: equational theories and trace inclusion

In this Appendix we prove the main results from Sect. 5, i.e. that the preorder for strongly bound processes preserves trace inclusion (Theorem 5.3), and that the same happens for weakly bound processes (Theorem 5.7). Finally, in Theorem 5.8 we show that the equational theory of weakly bound processes preserve well-boundness, and that processes smaller (w.r.t. $\precsim_{\mathcal{N}}$) than well-bound processes are well-bound.

Definition D.1 The *names* $\mathsf{N}(p)$ of a weakly bound process p are defined as:

$$\mathsf{N}(p) = \mathit{fn}(p) \cup \mathit{bn}^{\diamond}(p)$$

The following definition introduced a preorder $\precsim_N$ between semantic functions in D_w. Roughly, $Y \precsim_N Z$ holds when the traces returned by Y are included in those of Z, neglecting the possible extra bindings of names in $\mathcal{N}$ returned by Y.

Definition D.2 Let $Y, Z \in D_w$. We write $Y \precsim_N Z$ when for all $\mathcal{R}, \chi$ such that $\chi \cap \mathcal{N} = \emptyset$:

$$\{\, (\eta, \chi'|_{\mathit{dom}(\chi')\setminus\mathcal{N}}) \mid (\eta, \chi') \in Y(\chi)(\mathcal{R}) \,\} \subseteq Z(\chi)(\mathcal{R})$$

Let $Y, Z \in D_{sub}$, we write $Y \preceq Z$ whenever for all $\mathcal{R}$ we have $Y(R) \subseteq Z(R)$.

The following lemma states that the semantic function $[\![\cdot]\!]^{sub}_{\theta}$ is monotonic on the argument θ. This fact is used in the proof of Theorem 5.3, in the case of recursion.

Lemma D.3 *For all strongly bound processes P, and for all θ, θ':*

$$\theta \preceq \theta' \implies [\![P]\!]^{sub}_{\theta} \preceq [\![P]\!]^{sub}_{\theta'}$$

Theorem 5.3. For all closed, strongly bound processes P and P':

- if $P = P'$, then $[\![P]\!]^{s}_{\emptyset} = [\![P']\!]^{s}_{\emptyset}$.
- if $P \preceq P'$ then $[\![P]\!]^{s}_{\emptyset}(\chi)(\mathcal{R}) \subseteq [\![P']\!]^{s}_{\emptyset}(\chi)(\mathcal{R})$, for all $\mathcal{R}$ and χ.

The following two lemmata about bound and free names are straightforward.

Lemma D.4 *For all weakly bound processes p and q such that $p \approx q$:*

$$\mathit{fn}(p) = \mathit{fn}(q) \qquad \mathit{bn}^{\square}(p) = \mathit{bn}^{\square}(q) \qquad \mathit{bn}^{\diamond}(p) = \mathit{bn}^{\diamond}(q)$$

Lemma D.5 *For all weakly bound processes p and q such that $p \precsim_{\mathcal{N}} q$:*

$$\mathit{fn}(p) \subseteq \mathit{fn}(q) \qquad \mathit{bn}^{\diamond}(p) \subseteq \mathit{bn}^{\diamond}(q) \cup \mathcal{N} \qquad \mathit{bn}^{\square}(p) \supseteq \mathit{bn}^{\square}(q)$$

The following definition formalizes the notion of "captures" and capture-avoidance in substitutions of h-variables in weakly bound processes.

Definition D.6 The *captures* $cpt_{\mathcal{N}}(p,h)$ of h in a weakly bound process p are defined inductively as follows:

$$cpt_{\mathcal{N}}(\varepsilon,h) = cpt_{\mathcal{N}}(\,!\,,h) = cpt_{\mathcal{N}}(\alpha(\rho),h) = \emptyset \qquad cpt_{\mathcal{N}}(h',h) = \begin{cases} \mathcal{N} & \text{if } h = h' \\ \emptyset & \text{otherwise} \end{cases}$$

$$cpt_{\mathcal{N}}(p_0 \cdot p_1, h) = cpt_{\mathcal{N}}(p_0 + p_1, h) = cpt_{\mathcal{N}}(p_0,h) \cup cpt_{\mathcal{N}}(p_1,h)$$

$$cpt_{\mathcal{N}}(\mu h'.p', h) = \begin{cases} cpt_{\mathcal{N} \cup bn^{\diamond}(p')}(p',h) & \text{if } h \neq h' \\ \emptyset & \text{otherwise} \end{cases}$$

We say $p\{p'/h\}$ *capture-avoiding* if $bn^{\diamond}(p') = \emptyset = cpt_{bn^{\diamond}(p)}(p,h) \cap fn(p')$.

The following lemma is the weakly-bound analogous of Lemma A.24. Note that in this case the statement is trickier, since it requires capture-avoidance, and the semantics $[\![p']\!]^w_\theta$ must be suitably projected on D_{sub}.

Lemma D.7 (Substitution) *If $p\{p'/h\}$ is capture-avoiding, then:*

$$[\![p\{p'/h\}]\!]^w_\theta(\chi)(\mathcal{R}) \quad = \quad [\![p]\!]^w_{\theta\{\lambda\bar{\mathcal{R}}.\, fst([\![p']\!]^w_\theta(\chi)(\bar{\mathcal{R}}))/h\}}(\chi)(\mathcal{R})$$

Lemma D.8 *If $X \precsim_{\mathcal{N}} Y$, $Y \precsim_{\mathcal{N}'} Z$, then $X \precsim_{\mathcal{N} \cup \mathcal{N}'} Z$.*

The following lemma states that the semantic function $[\![p]\!]^w_\theta$ is monotonic on the argument p w.r.t. the preorder $\precsim$. This fact is used to prove Theorem 5.7.

Lemma D.9 *For all weakly bound processes p, p' and for all $\mathcal{N}$ and θ:*

(D.9a) $\quad p \approx p' \implies [\![p]\!]^w_\theta \precsim_\emptyset [\![p']\!]^w_\theta$

(D.9b) $\quad p \precsim_{\mathcal{N}} p' \implies [\![p]\!]^w_\theta \precsim_{\mathcal{N}} [\![p']\!]^w_\theta$

The following Theorem relates $\approx$ and $\precsim$ with trace inclusion. The first item follows from Lemma D.9a. The second item follows from Lemma D.9b and by Def. D.2.

Theorem 5.7. For all closed, weakly bound processes p and q:

- if $p \approx q$, then $[\![p]\!]^w_\emptyset = [\![q]\!]^w_\emptyset$.
- if $p \precsim_{\mathcal{N}} q$, then $fst([\![p]\!]^w_\emptyset(\chi)(\mathcal{R})) \subseteq fst([\![q]\!]^w_\emptyset(\chi)(\mathcal{R}))$, for all $\mathcal{R}$ and χ such that $dom(\chi) \cap \mathcal{N} = \emptyset$ and both the semantics are defined.

The following Theorem relates well-boundness with $\approx$ and $\precsim$. The first items is straightforward by Lemma D.4 and by induction on the derivation of $p \approx q$. The second item is by induction on the structure of p. The last item is straightforward by Lemma D.5 and by induction on the derivation of $p \precsim_{\mathcal{N}} q$.

Theorem 5.8. For all weakly bound processes p and q:

- if $p \approx q$, then $wb(p)$ if and only if $wb(q)$.
- if $p \approx q$ and $wb(p)$, then $bindify(p) = bindify(q)$.
- if $p \precsim_{\mathcal{N}} q$ and $wb(q)$, then $wb(p)$.