A fully abstract coalgebraic semantics for the pi-calculus under weak bisimilarity

Andrei Popescu

University of Illinois at Urbana-Champaign and
Institute of Mathematics "Simion Stoilow" of the Romanian Academy

Abstract. Combining traces, coalgebra and lazy-filtering channel configurations for parallel composition, we give a fully-abstract denotational semantics for the π-calculus under weak early bisimilarity.

1 Introduction

Weak bisimilarity (henceforth, as customary, referred to simply as *bisimilarity*) is one of the most reasonable equivalences defined on nondeterministic interactive processes. A nondeterministic process p may take one of several actions a_i and transit to a process p_i; among the possible actions, there is the *silent*, *unobservable* action, τ. Intuitively, a process may take a τ action at any time without being noticed, while any other action is *observable*. Now, the *(behavioral) identity* of two processes p_1 and p_2 can be determined by the following two clauses (and their symmetrics): **(i)** if p_1 may do something internally (i.e., take zero or more τ actions) becoming p_1', then p_2 may as well do something internally becoming p_2', behaviorally identical to p_1'; **(ii)** if p_1 may do something internally, then take an observable action a, then again do something internally eventually becoming p_1', then p_2 may as well go through a similar series of steps (consisting of action a possibly succeeded and/or preceded by some silent steps) eventually becoming p_2', behaviorally identical to p_1'. The notion of bisimilarity, although essentially based on the above idea, may take slightly more complicated forms in calculi which involve scope and substitution/instantiation, where additional phenomena, like that of substituting actual parameters for formal parameters, need to be considered. Moreover, in such calculi (as well as in calculi which are simpler but involve operators not compatible with bisimilarity) the behavioral identity of a process is not given by bisimilarity directly, but rather by *bisimilarity congruence*, which holds for two processes if, when placed in the same contexts, yield bisimilar items. The *strong bisimilarity* of processes is a useful, but often too crude, approximation of bisimilarity, obtained by assuming the internal action τ observable.

The typical route for introducing processes is through some syntactic constructs building terms whose behavior is given by an SOS-specified labeled transition system. The identity of a process can then be taken to be the bisimilarity class of a term. A more direct, and often more insightful way of establishing the identity of the described processes is to show that process terms really describe processes(!), in other words, to assign a denotation from a *domain of processes* (or *behaviors*) to each term. The agreement between bisimilarity defined operationally and the kernel of the denotation map is referred to as *full abstraction*.

Bisimilarity, although extensively studied in various operational settings [12,

14, 17], has not benefited yet, in our opinion, from a suitable compositional denotational-semantics account not even in the name-free calculi such as CCS. (Although a certain ad hoc compositionality for CCS and CSP has been achieved in [3] and could be achieved along the lines of [15], and non-compositional denotational models for CCS, CSP and π under weak bisimilarity have been sketched in [10, 11] – see Section 4 and Appendix C.) To the contrary, *strong* bisimilarity, having a simpler structure, has received compositional fully-abstract semantics, both domain-theoretic [1, 7, 18] and coalgebraic [15, 20], based on hypotheses that hold (or merely using techniques that work) for many process calculi in current use. The same holds true for *may/must testing equivalence*, also simpler than bisimilarity but in a different way (namely, in that establishing it does not involve any alternating game as for strong bisimilarity and bisimilarity) [8, 4].

The difficulty with assigning compositional denotational semantics for bisimilarity seems to emerge from the fundamental divorce between two features: **(I)** the traditional one-action-depth, "branching-time" presentation of a process calculus; **(II)** the "linear-time" consideration of τ^*-sequences in the definition of bisimilarity. Considering (I), one is tempted to transform right away the conditional rules from the SOS presentation of the system into corresponding corecursive or fixpoint recursive definitions, leading to problems with (II). The solution proposed by this paper comes from resisting this temptation, and acknowledging that, due to the possibility of melting observable actions into silent actions via communication, *in order to handle what happens arbitrarily τ-deep into processes, one also needs to deal with what happens arbitrarily deep along any sequence of actions.* This might suggest to abandon coalgebraic semantics altogether and go after some form of pure trace semantics, but this would be a mistake, since the very nature of bisimilarity is coalgebraic – the infinite game that defines bisimilaritiy is a journey into processes with infinitely often occurring stops and restarts. Instead, we shall combine traces with coalgebra, identifying a process with all pairs (traceOfActions,continuationProcess) such that that trace of actions is possible and leads to that continuation.

Technically, we shall regard bisimilarity as strong bisimilarity where the "actions" are now sequences of *observable* actions, and where a chain of zero or more τ actions is represented by the empty sequence ϵ. The monoidal properties of the set of action sequences shall be reflected in modal axioms for the final coalgebraic semantic domain. Of course, defining semantic operations within such an approach requires a preliminary combinatorial study of sequences/traces of actions, that need to be shuffled in consistent ways. As it happens, paying a priori attention to traces has its rewards – operations on processes like parallel composition and restriction, main characters in process algebra, receive concise, uniform and conceptually clear definitions.

This paper applies the above idea to the π-calculus. More precisely, we give a compositional fully abstract denotational semantics to the synchronous π-calculus under (weak) early bisimilarity congruence. Here, implementing the idea poses additional challenges to the situation of a simpler process algebra such as CCS or CSP, since the sequencing of actions might change the channel

topology of the resulted process in quite complicated ways. We handle this dynamic channel topology using what we call *(channel) configurations* – they are data structures held at "run-time" when composing two processes in parallel, having the potential of evolving during (co)recursive calls. A configuration keeps track of the private links (that have been established so far) between two processes π_1 and π_2 running in parallel, as well as of the views that these processes have towards the outside world. Thus, a configuration acts as an interface both between π_1 and π_2 and between each of them and the outside world. All actions (synchronized or independent) of π_1 and/or π_2, and, consequently, the whole behaviors of π_1 and π_2: **(1)** on the one hand are filtered through the current configuration; **(2)** on the other hand may change the configuration.

To illustrate (1), assume π_1 is ready to send j_1 on i_1 and π_2 is ready to receive j_2 on i_2. In our framework, the behavior of the parallel composite $\pi_1|\pi_2$ is not defined unless a configuration is given. So assume we have a configuration that stores the pair (i_1, i_2) as a private link. Then communication from π_1 to π_2 is possible through this link provided j_1 and j_2 are recognized by the configuration to represent the same channel (the latter meaning that either both j_1 and j_2 represent the same public channel or they constitute another private link). To illustrate (2), suppose that π_1 decides, and is able to send i_1, which is currently part of a private link (i_1, i_2) with π_2, to the outside – this corresponds to scope extrusion; in our setting, the effect of π_1 sending i_1 is the *publication* of the link (i_1, i_2), meaning that the configuration changes by establishing a new public channel (which will henceforth be viewed by π_1 as i_1 and by π_2 as i_2) and deleting the private link (i_1, i_2). Private links may not only be deleted, but also created – the latter happens if one of the processes, say π_1, sends a new channel to π_2. The mechanism for communicating new channels is also controlled by configurations (and affects the configurations at the same time): a new channel is simply one that does not exist in the configuration; upon encountering a new channel, say j_1 sent by π_1, the configuration either establishes a new private link between π_1 and π_2 if the sent channel was meant for π_2 (and if π_2 is willing and able to receive it), or it establishes a new public channel (the smallest one available – this exists because configurations are finite), henceforth viewed by π_1 as its own j_1, and sends it outside. Our configurations being held at "run-time" rather than being part of semantic processes allows for full abstraction, in spite of the great deal of concrete operational information contained in configurations.

Here is how the remaining of the paper is structured. The rest of this section is dedicated to general mathematical preliminaries and conventions and to the recollection of the syntax and operational semantics of π-calculus. Section 2 discusses, in coalgebraic terms, semantic models for weak bisimilarity in general, without reference to syntax. Section 3 adapts our approach to weak bisimilarity to the π-calculus, resulting in a fully-abstract denotational semantics. Section 4 draws conclusions and discusses related work. The appendix has three sections, A,B, C, with more details on the topics in Sections 2,3,4, respectively.

General preliminaries and conventions. When we introduce a metavariable (such as x) to range over a certain domain, we implicitly assume that this

is also the case for its version decorated with accents and subscripts (such as x_1, x'). We use λ-abstractions and "let" expressions informally, at the meta-level. Meta-level binding operators such as λ and $\prod$ have the lowest priority (thus, e.g., $\prod_{a\in A} B_a \to C$ is $\prod_{a\in A}(B_a \to C)$), and $\times$ has a higher priority than $\to$. We work within a standard set theory such as Zermelo-Fraenkel with Urelements, where the Axiom of Foundation is replaced by the Anti-Foundation Axiom (AFA), the latter stating that any directed graph (not necessarily acyclic) can have its vertexes labeled with sets in such a way that the edges correspond to membership [2]. (Our usage of non-well-founded sets is not strictly necessary, a set-theoretically well-founded coalgebra doing the same job up to isomorphism – we prefer non-well-founded sets though as an elegant implementation of the final-coalgebra-semantics dogma "to be is to do" (Jan Rutten).) We make the standard distinction between classes and sets. For a class C, $\mathcal{P}(C)$ is the class of all *subsets* (not subclasses!) of C and $\mathcal{P}f(C)$ is the class of all finite subsets of C. By "relation" we mean, by default, "binary relation". For a relation $R \subseteq A \times B$, $R^\smile \subseteq B \times A$ is its converse relation, and $Prj_1(R) \subseteq A$ and $Prj_2(R) \subseteq B$ are its first and second projection, respectively. For two relations $R \subseteq A \times B$ and $S \subseteq B \times C$, $S \circ R$ is the composition of S with R, namely, $\{(a, c).\ \exists b.\ (a, b) \in R \wedge (b, c) \in S\}$. (The function composition notation is a particular case of this.) Given a function $f : A \to B$ and $A' \subseteq A$, $Im(f)$ is the image of f and $f|_{A'}$ is the restriction of f to A'. We usually denote function application by juxtaposition, as in $f\,x$, but sometimes also employ the parenthesized notation $f(x)$. Given $f : A \to B$, $a \in A$ and $b \in B$, $f[a \leftarrow b] : A \to B$ is defined by $(f[a \leftarrow b])a' = b$ if $a' = a$ and $= f\,a'$ otherwise. A^* is the set of finite sequences of items in A, i.e., the set of words over the alphabet A. $\#$ denotes, in an infixed manner, word concatenation, and ϵ the empty word. (I.e., $(A^*, \#, \epsilon)$ is the monoid freely generated by A.) Thus, for two words $w_1, w_2 \in A^*$ (unless otherwise stated), $w_1 \# w_2$, and *not* the simple juxtaposition $w_1 w_2$, denotes their concatenation. However, given the elements $a_1, \ldots, a_n \in A$, we write $a_1 \ldots a_n$ for the word (sequence) built with these letters (in this order). Rather than working with partial functions, we shall consider total functions in $A \to B_\perp$, where $B_\perp = B \cup \{\perp\}$ with $\perp \notin B$, which induce (or can be regarded as) partial functions between A and B in the obvious way. $\mathbf{\bot}$ is the constant function in $A \to B_\perp$ returning $\perp$ for all inputs. For $m, n \in I\!N$, $\overline{m,n}$ is $\{i.\ m \leq i \leq n\}$.

Syntax and operational semantics of the π-calculus recalled. To avoid cluttering the presentation of the ideas with irrelevant technical details, we shall consider a π-calculus without sum, match and mismatch. The way to handle these in our framework should be obvious (guarded sum will be union of the continuations with the guards appended). The next presentation of the calculus follows [14] quite closely. We also refer to [13, 17] for the notion of bisimilarity. We fix a countably infinite set *Var*, of *(channel) variables*, or *names*, ranged over by x, y. The sets *Pterm*, of *process terms* (*pterms* for short), ranged over by P, Q, and *Gact*, of *generic actions*, ranged over by γ, are defined by the grammars:

$P \ ::= \ 0 \mid \overline{x}y.P \mid x(y).P \mid \nu\, y.P \mid \ P|Q \ \mid \ !P \qquad\qquad \gamma \ ::= \ \overline{x}y \mid \overline{x}(y) \mid x(y) \mid \tau$

(The adjective "generic" is meant distinguish these actions, involving variables,

from the more "concrete" actions introduced later for our domain.) $\overline{x}y$ sends a free name y on x, $\overline{x}(y)$ sends a fresh (bound) name y on x, $x(y)$ receives the generic (bound) name y on x, and τ is the silent action. In a pterm of the form $x(y).P$ or $\nu\, y.\, P$, y is bound in P – we identify pterms modulo the induced alpha-equivalence. $FV(P)$ denotes the set of free variables of P. Assume $x_1, \ldots, x_n$ are distinct. Then $z[y_1/x_1, \ldots, y_n/x_n]$ is y_i if $z = x_i$ for some i and z otherwise and $P[y_1/x_1, \ldots, y_n/x_n]$ is the term obtained by simultaneously substituting (in a capture-free way) the y_i's for the x_i's in P. In the actions $\overline{x}(y)$ and $x(y)$, x is free and y bound; in the action $\overline{x}y$, x and y are free. $BV(\gamma)$, $FV(\gamma)$ and $V(\gamma)$ denote the (at most two-element) sets of all the *bound*, *free* and *arbitrary* variables of γ.

The transition system for the calculus, OS (for "the Original System") is given below. (We omit the left-right symmetrics (ParR), (ComOR), (ComNR) of (ParL), (ComOL), (ComNL).) The system uses *late-instantiation input commitments*, i.e., a process commits to an input from outside *generically* (rule (Inp)), instantiation of the actual parameter for the formal parameter happening late, at communication time ("Com...").

$$\frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}(\mathsf{Out}) \qquad \frac{}{x(y).P \xrightarrow{x(y)} P}(\mathsf{Inp}) \qquad \frac{P \,|\, !P \xrightarrow{\gamma} P'}{!P \xrightarrow{\gamma} P'}(\mathsf{Repl}) \qquad \frac{P \xrightarrow{\overline{x}y} P'}{\nu\, y.\, P \xrightarrow{\overline{x}(y)} P'}\ [x \neq y]\ (\mathsf{Open})$$

$$\frac{P \xrightarrow{\gamma} P'}{\nu\, z.\, P \xrightarrow{\gamma} \nu\, z.\, P'}\ [z \notin V(\gamma)]\ (\mathsf{Nu}) \qquad \frac{P \xrightarrow{\gamma} P'}{P \,|\, Q \xrightarrow{\gamma} P' \,|\, Q}\ [BV(\gamma) \cap FV(Q) = \emptyset]\ (\mathsf{ParL})$$

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, (Q'[z/y])}(\mathsf{ComOL}) \qquad \frac{P \xrightarrow{\overline{x}(y)} P' \qquad Q \xrightarrow{x(y)} Q'}{P \,|\, Q \xrightarrow{\tau} \nu\, y.\, P' \,|\, Q'}(\mathsf{ComNL})$$

We write $\vdash_{OS} P \xrightarrow{\gamma} P'$ to indicate that the transition $P \xrightarrow{\gamma} P'$ is derivable in OS. For a set L of words over *Gact* (i.e., a language over the alphabet *Gact*), we write $\vdash_{OS} P \xrightarrow{L} P'$ to indicate that there exist $n \in \mathbb{N}$, $P_0, \ldots, P_n$ and a word $\gamma_0 \ldots \gamma_n \in L$ such that $P_0 = P$, $P_n = P'$, and $\vdash_{OS} P_i \xrightarrow{\gamma_i} P_{i+1}$ for all $i \in \overline{0, n-1}$. Given $\gamma \in \mathit{Gact}$, the expressions $\tau^*\gamma\,\tau^*$, $\tau^*\gamma$, $\gamma\,\tau^*$ and τ^* denote the obvious regular languages (where this time we decided to omit the concatenation operator # in order to keep notation simple). A *(weak) early simulation* is a binary relation θ on *Pterm* such that the following four clauses hold: **(1)** if $P\,\theta\,Q$ and $\vdash_{OS} P \xrightarrow{\tau^*} P'$, then there exists Q' such that $\vdash_{OS} Q \xrightarrow{\tau^*} Q'$ and $P'\,\theta\,Q'$; **(2)** if $P\,\theta\,Q$ and $P \xrightarrow{\tau^*\overline{x}y\,\tau^*} P'$, then there exists Q' such that $\vdash_{OS} Q \xrightarrow{\tau^*\overline{x}y\,\tau^*} Q'$ and $P'\,\theta\,Q'$; **(3)** if $P\,\theta\,Q$ and $\vdash_{OS} P \xrightarrow{\tau^*\overline{x}(y)\,\tau^*} P'$ with $y \notin FV(Q)$, then there exists $Q' \in \mathit{Pterm}$ such that $\vdash_{OS} Q \xrightarrow{\tau^*\overline{x}(y)\,\tau^*} Q'$ and $P'\,\theta\,Q'$; **(4)** if $P\,\theta\,Q$, $\vdash_{OS} P \xrightarrow{\tau^* x(y)} P'$ and $\vdash_{OS} P'[z/y] \xrightarrow{\tau^*} P''$ with $y \notin FV(Q)$, then there exist $Q', Q'' \in \mathit{Pterm}$ such that $\vdash_{OS} Q \xrightarrow{\tau^* x(y)} Q'$, $\vdash_{OS} Q'[z/y] \xrightarrow{\tau^*} Q''$ and $P''\,\theta\,Q''$.

An *early bisimulation* is an early simulation whose converse is also an early simulation. The *early bisimilarity* relation on pterms, denoted $\sim_{OS}$, is the largest early bisimulation. Bisimilarity is often alternatively called *observation equivalence*, because it is based on a notion of experimenting over processes. The interesting clause in the definition of early simulation is the one for input: an experiment allows the process to take zero or more silent steps both before and after receiving the actual input datum. $\sim_{OS}$ is compatible with all the

syntactic constructs except for input. Two pterms P and Q are called *early bisimilarly congruent*, written $P \equiv_{OS} Q$, if the pair (P, Q) belongs to the congruence cogenerated by $\sim_{OS}$, i.e., if $C[P] \sim_{OS} C[Q]$ for all contexts $C[*]$, i.e., if $P[y_1/x_1, \ldots, y_n/x_n] \sim_{OS} Q[y_1/x_1, \ldots, y_n/x_n]$ for all $n \in \mathbb{N}$, $x_1, \ldots, x_n$ distinct variables and $y_1, \ldots, y_n$ variables. (A suitable quantification swapping in the clauses for input yields *late* bisimilarity [14, 13], the difference from early bisimilarity laying in the atomicity of choosing the port and the datum: in one single step for the former, in two steps for the latter.)

2 Semantic domain for weak bisimilarity in general

In this section, we present a semantic domain for weak bisimilarity based on traces of actions, and show its connection with, and its advantage over a more standard domain based on single actions. (Appendix A gives more details.)

Cls denotes the category of classes and functions between classes. In what follows, we shall employ basic concepts and results about coalgebras [16] and non-well-founded sets [2]. We only recall here a couple of definitions from coalgebra. Given a functor $F : Cls \to Cls$, an F-coalgebra is a pair $(A, \alpha : A \to F\,A)$. A *morphism* between two F-coalgebras (A, α) and (B, β) is a map $f : A \to B$ such that $(F\,f) \circ \alpha = \beta \circ f$. A *stable part* of an F-coalgebra (A, α) is a subclass $X \subseteq A$ such that $\forall x \in X.\, \alpha\, x \in F\,X$; a stable part X yields a *subcoalgebra* $(X, \alpha\,|_X)$. Given a fixed set Z, the functor $\mathcal{P}(Z \times \_)$ on *Cls* maps each X to $\mathcal{P}(Z \times X)$ and each $f : X \to Y$ to $\lambda K.\, \{(z, f\,x).\, (z, x) \in K\}$. Fix *Act*, a set of *actions*. We only consider coalgebras for $\mathcal{P}((Act \cup \{\epsilon\}) \times \_)$ and $\mathcal{P}(Act^* \times \_)$, which we call *one-step*, and *multi-step coalgebras*, respectively.

Domains of processes under strong bisimilarity are typically modeled as final one-step coalgebras. However, if the desired process identity is bisimilarity, then (also having in mind that operations like parallel composition need to take a deeper, multi-step look into the argument processes) it is more natural to consider a suitable *multi-step* coalgebra as the domain. But also we would like to keep in sight that bisimilarity is a weakening of strong bisimilarity by internalizing τ, in particular, to be able to infer bisimilarity from strong bisimilarity without any detour, and also to infer bisimilarity as usual, by showing how to simulate single steps only (by multi-steps). These lead to the following constructions.

Let *Preproc*, the class of *preprocesses*, be the largest class X satisfying the equation $X = \mathcal{P}(Act^* \times X)$. Since we assume AFA, $(Preproc, 1_{Preproc})$ is the final multi-step coalgebra [2]. Given a multi-step coalgebra (D, δ), an element $d \in D$ is said to be: *reflexive*, if $(\epsilon, d) \in \delta\, d$; *transitive*, if $\forall w, w', d', d''.\, (w, d') \in \delta\, d \wedge (w', d'') \in \delta\, d' \Rightarrow (w \# w', d'') \in \delta\, d$; *prefix-closed*, if $\forall w, w', d''.\, (w \# w', d'') \in \delta\, d \Rightarrow (\exists d'.\, (w, d') \in \delta\, d \wedge (w', d'') \in \delta\, d')$; *monoidal*, if it is reflexive, transitive and prefix-closed. (D, δ) is said to be *monoidal* if all elements of D are monoidal.

Let *Proc*, the class of *processes*, be the stable part of the multi-step coalgebra $(Preproc, 1_{Preproc})$ cogenerated by the class of all monoidal preprocesses. This notion of process encompasses two ideas. First, processes have a *linear-time* structure which is compatible with the monoid of action sequences (via reflexivity and transitivity) and has no discontinuities (via prefix-closeness). Second, processes have the above linear-time properties preserved by the *branching-*

time, coalgebraic structure – one should think of a process as an *hereditarily* monoidal preprocess, that is, a preprocess π such that π is monoidal and the preprocesses from all its arbitrarily deep unfoldings are so. The properties of the linear-time structure, making sense for any transition system labeled with sequences of actions (i.e., for any multi-step coalgebra), are the essential features of weak bismilarity. Our choice to impose these properties hereditarily deep for elements in the final multi-step coalgebra makes $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$ the *final monoidal multi-step coalgbebra,* featuring the following *corecursive definition* principle, employed in the next section for giving semantics to the π-calculus: to define a (parameterized) operation $f : Param \times Proc^n \rightarrow Proc$, it suffices to organize $Param \times Proc^n$ as a *monoidal* multi-step coalgebra by defining $\delta : Param \times Proc^n \rightarrow \mathcal{P}(Act^* \times (Param \times Proc^n))$ (then take f to be the unique morphism between $(Param \times Proc^n, \delta)$ and $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$).

Moreover, the fact that $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$ is *simple* (being a subcoalgebra of the (absolutely) final coalgebra), yields standardly a proof principle: Assume $\theta \subseteq Proc \times Proc$ is a strong bisimulation on $Proc$ (regarded as an Act^*-labeled transition system), in that the following hold for all $(\pi, \pi') \in \theta$: **(i)** $\forall (w, \pi'') \in \pi.\, \exists \pi'''.\, (w, \pi''') \in \pi' \wedge (\pi'', \pi''') \in \theta$; **(ii)** $\forall (w, \pi'') \in \pi'.\, \exists \pi'''.\, (w, \pi''') \in \pi \wedge (\pi''', \pi'') \in \theta$. Then $\pi = \pi'$ for all $(\pi, \pi') \in \theta$. Thanks to the affinity between $Proc$ and the monoidal structure of Act^*, we also have a simpler (but equally powerful) proof principle which reflects more closely the traditional way of dealing with weak bisimilarity, by showing how to simulate single actions only: Let $\theta \subseteq Proc \times Proc$ be such that the following hold for all $(\pi, \pi') \in \theta$: **(i)** $\forall (w, \pi'') \in \pi.\, |w| \in \{0, 1\} \Rightarrow (\exists \pi'''.\, (w, \pi''') \in \pi' \wedge (\pi'', \pi''') \in \theta)$; **(ii)** $\forall (w, \pi'') \in \pi'.\, |w| \in \{0, 1\} \Rightarrow (\exists \pi'''.\, (w, \pi''') \in \pi \wedge (\pi''', \pi'') \in \theta)$. Then $\pi = \pi'$ for all $(\pi, \pi') \in \theta$. The latter proof principle may appear, at first, as if employing *strong* bisimulation (w.r.t. *single* actions) – but remember that processes absorb the monoidal properties of action sequences; in particular: **(1)** ϵ is identified with (any sequence in the language) ϵ^*, thus meaning "zero or more silent steps"; **(2)** the single action a is identified with $\epsilon^* \# a \# \epsilon^*$, meaning "$a$ preceded and succeeded by zero or more silent steps". Thus, *weak* bisimulation is what we really deal with here, strong bisimulation being only a particular case.

The fact that each process is uniquely identified by its behavior w.r.t. sequences of length 0 or 1 (i.e., elements of $Act \cup \{\epsilon\}$) suggests a more compact representation of the domain of processes as a one-step coalgebra. As already mentioned, the choice of the (absolutely) final one-step coalgebra as the semantic domain for strong bisimilarity typically already yields the desired properties (in particular, full abstraction [15]). However, here, since we are after bismilarity, we shall require that processes, even in this more compact representation with single steps, retain the affinity with the monoid of action sequences – this means here that zero or more ϵ-steps can always be appended and/or prepended "silently", yielding a property similar to monoidality that we shall call ϵ-monoidality. (Although the constructions below are not needed later in the paper, they are useful for placing our domain in a context that clarifies its connection with the traditional view on bisimilarity and its benefit w.r.t. compositionality.)

Let *Cpreproc*, the class of *compact (representations of) preprocesses*, be the largest X with $X = \mathcal{P}((Act \cup \{\epsilon\}) \times X)$, making $(Cpreproc, 1_{Cpreproc})$ the final one-step coalgebra. Given a one-step coalgebra (D, δ), an element $d \in D$ is said to be: *ϵ-reflexive*, if $(\epsilon, d) \in \delta\, d$; *$\epsilon$-transitive*, if $\forall d', d'' \in D.\ (\epsilon, d') \in \delta\, d \wedge (\epsilon, d'') \in \delta\, d' \Rightarrow (\epsilon, d'') \in \delta\, d$; *$\epsilon$-loud-transitive*, if $\forall d', d'', d''' \in D. \forall a \in Act.\ (\epsilon, d') \in \delta\, d \wedge (a, d'') \in \delta\, d' \wedge (\epsilon, d''') \in \delta\, d'' \Rightarrow (a, d''') \in \delta\, d$; *$\epsilon$-monoidal*, if it is ϵ-reflexive, ϵ-transitive and ϵ-loud-transitive. (D, δ) is said to be *ϵ-monoidal* if all elements of D are ϵ-monoidal. Let *Cproc*, the class of *compact (representations of) processes*, be the stable part of the one-step coalgebra $(Cpreproc, 1_{Cpreproc})$ cogenerated by the class of all ϵ-monoidal compact preprocesses. Then $(Cproc, Cproc \hookrightarrow \mathcal{P}((Act \cup \{\epsilon\}) \times Cproc))$ is the final ϵ-monoidal one-step coalgebra. Next, we write π for processes and σ for compact processes. Using the finality of *Cproc* and *Proc*, we define two maps, $pack : Proc \to Cproc$ and $unpack : Cproc \to Proc$, for moving back and forth between a process and its compact representation: $pack\ \pi = \{(w, pack\ \pi').\ (w, \pi') \in \pi \wedge w \in Act \cup \{\epsilon\}\}$ and $unpack\ \sigma = \{(w_1 \# \ldots \# w_n, unpack\ \sigma').\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\ w_i \in Act \cup \{\epsilon\}) \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\ \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\ (w_i, \sigma_{i+1}) \in \sigma_i))\}$. Thus, $pack\ \pi$ retains from π (and from all its (arbitrarily deep) continuations) only the one-step continuations, while $unpack\ \sigma$ expands σ (and all its continuations) along all possible sequences of steps. *pack* and *unpack* are mutually inverse bijections.

Compact processes are coalgebraically only one-step deep, hence correspond more directly to the traditional operational semantics presentation of process calculi. However, in our context of (weak) bisimilarity, these compact one-step representations have a salient disadvantage compared to (multi-step) processes: crucial operations in process calculi, such as parallel composition and iteration, are not definable purely coalgebraically on compact processes, the one-step coalgebra falling short on providing means to describe the composite process. To illustrate this point, assume that Act is endowed with a bijection $^- : Act \to Act$ which is involutive (in that $\overline{\overline{a}} = a$ for all $a \in Act$), and say we would like to define CCS-like parallel composition on processes (thus, we assume that a and $\overline{a}$ are to be interpreted as complementary actions, whose synchronization yields a silent action). The main task in front of us is to show how sequences of actions interact, possibly nondeterministically. Knowing how *single actions* interact, and assuming an *interleaving semantics*, we obtain the following definition of the *parallel composition* (or *synchronized shuffle*) $| : Act^* \times Act^* \to \mathcal{P}(Act^*)$ (where we extend $\#$ to sets of sequences in the obvious way): **(i)** $\epsilon|\epsilon = \{\epsilon\}$; **(ii)** if $(w_1, w_2) \neq (\epsilon, \epsilon)$, then $w_1|w_2 = \{a \# (w_1'|w_2) : w_1 = a \# w_1'\} \cup \{a \# (w_1|w_2') : w_2 = a \# w_2'\} \cup \{w_1|w_2' : \exists a.\ w_1 = a \# w_1' \wedge w_2 = \overline{a} \# w_2'\}$. Now, parallel composition of processes, $| : Proc \times Proc \to Proc$, simply follows coinductively the interaction law prescribed by the action sequence composition: $\pi_1|\pi_2 = \{(w, \pi_1'|\pi_2') : \exists w_1, w_2.\ w \in w_1|w_2 \wedge (w_1, \pi_1') \in \pi_1 \wedge (w_2, \pi_2') \in \pi_2\}$.

Note the separation of concerns in our definition of $|$: first we dealt with action sequence combinatorics, so that afterwards the definition of parallel composition of processes was stated *purely coalgebraically*, composing together the continuations of the components to yield the continuations of the composite. On the other

hand, if trying to define parallel composition on *compact processes*, one finds the colagebraic one-step depth of the latter insufficient for describing even the one-step behavior of the composite – this is because a step of the composite $\sigma_1|\sigma_2$ may result from an *arbitrarily long* sequence of interactions between steps taken by continuations of σ_1 and σ_2. In fact, the reader is invited to try to define parallel composition on *Cproc* and to notice that any reasonable definition would essentially appeal to our multi-step domain *Proc*, unpacking the components σ_1 and σ_2, composing these unpacked versions in *Proc*, and then packing the result (i.e., the operation on *Cproc* would be essentially $pack((unpack\ \sigma_1)|(unpack\ \sigma_2))$). This shows that *Proc*, and not *Cproc*, is the fundamental domain for compositional weak bisimilarity. One may argue that, via the bijections *pack* and *unpack*, *Proc* and *Cproc* are really the same domain, so that considering one or the other are two sides of the same coin. However, *Proc* and *Cproc* take nevertheless two *distinct views* to processes, with the multi-step view brought by *Proc*, as explained above, allowing for a cleaner compositionality and separation of concerns when defining process composition. When the effect of actions becomes more complex, with possible changes of the communication topology, the multi-step view brings an insight into the semantics of a calculus under bisimilarity which is essential for apprehending the "right" semantic operations for full abstraction.

3 Semantic domain and interpretation of the π-calculus

Here we combine our multi-step view on processes with a suitable notion of channel configuration to give semantics to the π-calculus under (weak) bisimilarity.

Modeling the channel topology semantically. In the first part of our semantic endeavor, we focus on modeling the dynamic connections between processes to a large extent independently from the intended process equivalence. Remember that the actions of the original system OS are called *generic actions*. The process terms (pterms) of OS should also be regarded as generic. The attribute "generic" refers in both cases to viewing the channel names x, y, etc. appearing in these items *not as actual channels, but as channel variables*. This view (completely standard in programming languages but less standard in π-calculus) implies that a pterm such as $\nu y.\,\overline{x}y.\overline{y}z.0$ denotes an actual process only via an interpretation of its free variables, here x and z, as actual channels. Thus channels are *distinct* from channel variables – we take $\mathbb{N}$ as the set of channels. (Hence an interpretation will be an environment ρ assigning numbers to channel variables.)

We can now talk about *concrete actions* (simply referred to as *actions*), having a very basic format – we define Act, ranged over by a, to be $Vb \times \mathbb{N} \times \mathbb{N}$, where Vb, the set of *verbs*, ranged over by vb, is $\{\mathsf{sd}, \mathsf{rc}\}$ ("send" and "receive"). In the action (vb, i, j), i is the *subject* channel, or the *port*, and j is the *object* channel, or the *datum*. (In accordance to our discussion in Section 2, we do not consider the silent action as a proper action – rather, zero or more silent actions, i.e., some internal change, will correspond to the empty action sequence ϵ.) For $vb \in Vb$, let its *complement*, $\overline{vb}$, be sd if $vb = \mathsf{rc}$ and rc if $vb = \mathsf{sd}$. The *semantic domain of processes* will be the one from Section 2, i.e., the *final monoidal multi-step coalgebra* $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$. A pterm will thus denote (given an environment, left implicit in the next example) a concrete-action process.

Example. Consider the pterm $P_1|P_2$, where P_1 is $\nu y.\,\overline{x}(y).\,P_1'$, P_1' is $y(z).\,P_1''$, P_2 is $x(y).\,P_2'$, and $P_2' = \overline{y}x.\,P_2''$, and say x denotes a channel i. Syntactically, in the parallel composite $P_1|P_2$, there is already a connection between P_1 and P_2, implicit in the usage of the same free variable x by P_1 and P_2. In our semantics, such connections will be explicit: if π_1 and π_2 are to be the denotations of P_1 and P_2, then, in order to form the denotation of the composite $P_1|P_2$, π_1 and π_2 are composed *through their existing connection* given by the two processes recognizing the same *public* channel i. It will be convenient to relax the notion of public connectivity by not imposing that π_1 and π_2 *share a public channel* like i, but allowing each of them to have *its own specific channel*, say i_1 for π_1 and i_2 for π_2, and asking that i_1 and i_2 be *mapped to the same public channel* i of the composite. Here $i_1 = i$ and $i_2 = i$, but in general i_1 and i_2, which are the *local views* towards the public channel i of the composite, need not coincide with i. Thus, the denotation of $P_1|P_2$ shall be $\pi_1|_c\pi_2$, where c is a *(channel) configuration*, initially containing the following information: **(1)** there is one public channel, i; **(2)** the view of π_1 towards the public channels is a partial map from numbers to $\{i\}$, namely $\{(i,i)\}$, and the view of π_2 towards the public channels is the same partial map $\{(i,i)\}$; **(3)** the set of private links between π_1 and π_2 is currently $\emptyset$. Syntactically, we have the transition $P_1|P_2 \xrightarrow{\tau} \nu y.(P_1'|\,P_2')$, creating a private link between the two processes. Semantically, this means that the process $\pi_1|_c\pi_2$ has the silent continuation $\pi_1'|_{c'}\pi_2'$, where π_1' and π_2' denote P_1' and P_2', c' holds the information of c with (3) updated from $\emptyset$ to $\{(j_1,j_2)\}$, as a result of the private link (j_1,j_2) being created after π_1 sent the fresh channel j_1 on i and π_2 received the fresh channel j_2 on i. Then, we have the transition $\nu y.(P_1'|\,P_2') \xrightarrow{\tau} \nu y.(P_1''[x/z]|P_2'')$, a private communication between the two processes. Semantically, the composite process $\pi_1'|_{c'}\pi_2'$ continues silently, via the communication of i between π_2' and π_1' through the private link (j_1,j_2), with $\pi_1''|_{c'}\pi_2''$, where π_1'' and π_2'' are the denotations of $P_1''[x/z]$ and P_2'' (now, the sent information does not affect the communication topology, i.e., c' stays the same).

We are led to the following concept. *Config*, the set of *configurations*, ranged over by c, consists of tuples (n, f_1, f_2, R) with: **(1)** $n \in I\!N$ (we call n the *level* of c); **(2)** $f_1 : I\!N \to \overline{0,n}_\perp$ and $f_2 : I\!N \to \overline{0,n}_\perp$ two functions that, regarded as partial functions from $I\!N$ to $\overline{0,n}$, are injective, i.e., $f_1\,i_1 = f_1\,j_1 \neq \perp$ implies $i_1 = j_1$ and $f_2\,i_2 = f_2\,j_2 \neq \perp$ implies $i_2 = j_2$; **(3)** $R \subseteq I\!N \times I\!N$ a finite bifunctional relation (i.e., both R and $R^\smile$ are functional relations), with the first projection of R disjoint from the domain of f_1 regarded as a partial function, i.e., $Prj_1(R) \cap \{i_1 \in I\!N : f_1\,i_1 \neq \perp\} = \emptyset$, and similarly for f_2, i.e., $Prj_2(R) \cap \{i_2 \in I\!N : f_2\,i_2 \neq \perp\} = \emptyset$.

As mentioned, a configuration represents the channel topology of two communicating processes π_1 and π_2 at a given time. The known public channels in the composite process are $\overline{0,n}$. The process π_u (with $u \in \{1,2\}$) has access to a (globally) public channel i provided there exists a (necessarily unique) i_u such that $f_u\,i_u = i$; in this case, π_u views i as its own (public) channel i_u. R is the set of private links established between π_1 and π_2 – if $(i_1,i_2) \in R$, then π_u regards that link as its own (private) channel i_u, and, moreover, i_1 and i_2 are

identified in the composite process: what π_1 views as i_1 *coincides* with what π_2 views as i_2. (R is essentially an injective partial function, like f_1 and f_2, but is regarded as a relation to emphasize the symmetry of the concept it represents.) The disjointness conditions for R versus f_1, f_2 state that a channel cannot be used for both public and private communication; however, the channel topology may evolve in time, so that private channels may become public.

Parallel composition in configurations. Parallel composition is central in the π-calculus (and in most process algebras). Because our semantic parallel composition will be *annotated with channel configurations*, it will also cover the restriction operation. This means that our semantic parallel composition in configurations will be the single operation responsible for the core of the π-calculus transition mechanism. It is now time to remember the tenet of Section 2: *parallel composition of processes (under bisimilarity) follows the parallel composition of sequences of traces*. Thanks to our usage of configurations, π-calculus processes are not required to change "eagerly" their internal contexts (in particular, to shift or swap internal channels) during communication in order to "keep up" with the outside changes, but rather all changes are "lazily" reflected in configurations that filter the behavior of the components – as a consequence, parallel composition will be as in Section 2, except that configurations are an extra parameter, varying during corecursive calls. Namely, if we assume a suitable configuration-parameterized parallel composition on traces of actions, $par : Config \times Act^* \times Act^* \to \mathcal{P}f(Config \times Act^*)$, the corresponding process operation, $\mathbf{Parc} : Config \times Proc \times Proc \to Proc$, simply allows all possible reactions-interleavings of sequences of actions between the two process arguments starting from the given configuration and calls itself corecursively for the continuation processes in the resulted configurations: $\mathbf{Parc}(c, \pi_1, \pi_2) = \{(w, \mathbf{Parc}(c', \pi_1', \pi_2')) : \exists w_1, w_2.\ \ (w_1, \pi_1') \in \pi_1 \wedge (w_2, \pi_2') \in \pi_2 \wedge (c', w) \in par(c, w_1, w_2)\}$. Thus, in $\mathbf{Parc}(c, \pi_1, \pi_2)$, the configuration argument $c = (n, f_1, f_2, R)$ acts, according to the laws of trace combinatorics, as an interface both between π_1 and π_2 (via R) and between π_1, π_2 and the rest of the world (via the "views" f_1 and f_2). (In the above example, we wrote $\pi_1|_c\pi_2$ instead of $\mathbf{Parc}(c, \pi_1, \pi_2)$.)

Trace combinatorics. But how to compose traces in configurations? Because composition of processes followed corecursively composition of action traces, we define trace composition having in mind composition of communicating processes! In other words, we consider two presumptive parallel processes, henceforth referred to as π_1 and π_2, and prescribe the effect of interleaving two sequences of actions, w_1 from π_1 and w_2 from π_2, thinking of how the communication topology should evolve as w_1 and w_2 are traversed recursively. The key concept here, that allows us to speak about communication links between π_1 and π_2 *in the absence of* π_1 *and* π_2, is of course that of *configuration*, purposely separated from the processes themselves. The effect that *(one-step) separate actions by* π_1 have on configurations is described by the relation $\_ \tilde{\rightsquigarrow} (\_, \_) \subseteq Config \times Act \times Config \times Act$ (where $c \overset{a}{\tilde{\rightsquigarrow}} (c', a')$ means: starting in configuration c, action a of π_1 changes the configuration into c' and results in action a' of the composite), defined to be the least relation satisfying the following clauses, whose intuitions explained in

parentheses refer to the previously discussed meanings of configuration components (below, "vb"-s stands for "sends" or "receives", depending on what vb is):

(S1) $(n, f_1, f_2, R) \overset{(vb, i_1, j_1)}{\rightsquigarrow} ((n, f_1, f_2, R), (vb, f_1\, i_1, f_1\, j_1))$, if $f_1\, i_1 \neq \bot \neq f_1\, j_1$
(π_1 "vb"-s j_1 on i_1, where both i_1 and j_1 are public channels, viewed outside as $f_1\, j_1$ and $f_1\, i_1$, respectively; the configuration does not change).

(S2) $(n, f_1, f_2, R) \overset{(\mathsf{sd}, i_1, j_1)}{\rightsquigarrow} ((n+1, f_1[j_1 \leftarrow n+1], f_2[j_2 \leftarrow n+1], R \setminus \{(j_1, j_2)\}), (vb, f_1\, i_1, n+1)))$, if $f_1\, i_1 \neq \bot$ and $(j_1, j_2) \in R$
(π_1 "vb"-s j_1 on i_1, where i_1 is public channel, viewed outside as $f_1\, i_1$, and j_1 is part of a private link with π_2, (j_1, j_2), sent outside as a fresh channel $n+1$; the private link (j_1, j_2) disappears from the configuration, and instead both j_1 of π_1 and j_2 of π_2 become public, viewed outside as $n+1$ (scope extrusion)).

(S3) $(n, f_1, f_2, R) \overset{(vb, i_1, j_1)}{\rightsquigarrow} ((n+1, f_1[j_1 \leftarrow n+1], f_2, R), (vb, f_1\, i_1, f_1\, n+1)))$, if $f_1\, i_1 \neq \bot = f_2\, j_2$ and $j_1 \notin Prj_1(R)$
(π_1 "vb"-s on the public channel i_1 (viewed outside as $f_1\, i_1$) the channel j_1 which is fresh (i.e., neither public, nor part of a private link with π_2); j_1 becomes public for π_1 and is henceforth viewed outside as a fresh channel $n+1$).

(S4) $(n, f_1, f_2, R) \overset{(\mathsf{rc}, i_1, j_1)}{\rightsquigarrow} ((n, f_1[j_1 \leftarrow j], f_2, R), (\mathsf{rc}, f_1\, i_1, j)))$, if $f_1\, i_1 \neq \bot = f_2\, j_2$, $j_1 \notin Prj_1(R)$, and $j \in \overline{0, n} \setminus Im(f_1)$.
(π_1 receives on the public channel i_1 (viewed outside as $f_1\, i_1$) the fresh channel j_1; j_1 becomes public for π_1 and is henceforth viewed outside as a channel j which, although previously available in the configuration (i.e., in $\overline{0, n}$), had not been known by π_1; thus, the capability of π_1 to receive fresh is used to bring π_1 "up to date" with some of the information from the configuration).

The effect that *one-step communications between π_1 and π_2, with π_1 sending and π_2 receiving,* have on configurations is described by the relation $\_ \overset{(\_,\_)}{\rightsquigarrow} \_ \subseteq Config \times Act \times Act \times Config$ (where $c \overset{(a_1, a_2)}{\rightsquigarrow} c'$ means: in configuration c, action a_1 of π_1 and action a_2 of π_2 react (silently), changing the configuration to c'), defined to be the least relation satisfying the following clauses, all having the implicit assumption $\bot \neq f_1\, i_1 = f_2\, i_2$ or $(i_1, i_2) \in R$ (meaning that i_1 of π_1 and i_2 of π_2 either are viewed outside as the same public channel, or form a private link):

(C1) $(n, f_1, f_2, R) \overset{((\mathsf{sd}, i_1, j_1), (\mathsf{rc}, i_2, j_2))}{\rightsquigarrow} (n, f_1, f_2, R)$, if $\bot \neq f_1 j_1 = f_2 j_2$ or $(j_1, j_2) \in R$
(communication of an already known (public or private) channel).

(C2) $(n, f_1, f_2, R) \overset{((\mathsf{sd}, i_1, j_1), (\mathsf{rc}, i_2, j_2))}{\rightsquigarrow} (n, f_1, f_2, R \cup \{(j_1, j_2)\})$, if $f_1\, j_1 = \bot = f_2\, j_2$, $j_1 \notin Prj_1(R)$, and $j_2 \notin Prj_2(R)$
(communication of a fresh channel, yielding a new private link).

(C3) $(n, f_1, f_2, R) \overset{((\mathsf{sd}, i_1, j_1), (\mathsf{rc}, i_2, j_2))}{\rightsquigarrow} (n, f_1, f_2[j_2 \leftarrow f_1\, j_1], R)$, if $f_1\, j_1 \neq \bot = f_2\, j_2$, $j_2 \notin Prj_2(R)$, and $f_1\, j_1 \notin Im(f_2)$ (communication of a channel which, although public, had not been known to π_2 – this clause is similar (S4)).

Now, the law for combining sequences cumulates the effect of one-step separate actions and one-step communications through an interleaving semantics. The operations $par, lmerge, com : Config \times Act^* \times Act^* \rightarrow \mathcal{P}f(Config \times Act^*)$ are defined mutually recursively on the structure of the last two arguments. par takes as input an initial configuration c and sequences of actions w_1 and w_2 for

(the presumptive) π_1 and π_2 and returns all pairs (w, c'), where: w is a sequence of actions resulted from well-defined reactions and interleavings of w_1 and w_2 starting in the configuration c (such that both w_1 and w_2 are fully exhausted); c' is the configuration obtained after all these reactions and interleavings. *lmerge* and *com* are two auxiliary operators introduced for convenience, for cumulating the effect of $\_ \bar{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\rightsquigarrow} \_$, respectively. Since *lmerge* and *com* do only "half" of the separate-action and communication jobs, *par* invokes each of them twice, once for the given configuration and once for its symmetric.

• $par((n, f_1, f_2, R), w_1, w_2) = lmerge((n, f_1, f_2, R), w_1, w_2) \cup lmerge((n, f_2, f_1, R^\smile), w_2, w_1) \cup com((n, f_1, f_2, R), w_1, w_2) \cup com((n, f_2, f_1, R^\smile), w_2, w_1)$.

• $lmerge(c, a\#w_1, w_2) = \{(c'', a'\#w).\ \exists c'.\ c \overset{a}{\rightsquigarrow} (c', a') \wedge (c'', w) \in par(c', w_1, w_2)\}$.

• $lmerge(c, \epsilon, w) = \emptyset$. • $com(c, \epsilon, \epsilon) = \{(c, \epsilon)\}$. • $com(c, \epsilon, a\#w) = com(c, a\#w, \epsilon) = \emptyset$.

• $com(c, a_1\#w_1, a_2\#w_2) = \{(c'', w).\ \exists c'.\ c \overset{(a_1,a_2)}{\rightsquigarrow} c' \wedge (c'', w) \in par(c', w_1, w_2)\}$.

Idle process, input and output. These are defined corecursively rather straightforardly: $\mathbf{Zero} \in Proc$, by $\mathbf{Zero} = \{(\epsilon, \mathbf{Zero})\}$; $\mathbf{Out} : \prod_{n \in \mathbb{N}} \overline{0,n} \times \overline{0,n} \times Proc \to Proc$, by $\mathbf{Out}\ n\ (i, j, \pi) = \{(\epsilon, \mathbf{Out}\ n\ (i, j, \pi))\} \cup \{((\mathsf{sd}, i, j)\ \#\ w, \pi') : (w, \pi') \in \pi\}$; $\mathbf{Inp} : \prod_{n \in \mathbb{N}} \overline{0,n} \times (\overline{0,n} \to Proc) \times Proc \to Proc$, by $\mathbf{Inp}\, n\, (i, h, \pi) = \{(\epsilon, \mathbf{Inp}\ n\ (i, h, \pi))\} \cup \{((\mathsf{rc}, i, j)\ \#\ w, \pi') : j \in \overline{0,n},\ (w, \pi') \in h\, j\} \cup \{((\mathsf{rc}, i, n+1)\ \#\ w, \pi') : (w, \pi') \in \pi\}$. **Zero** is the process that may only transit silently to itself. The first parameter, n, of **Out** and **Inp** represents the number of available channels. $\mathbf{Out}\, n\, (i, j, \pi)$ yields a process that, besides the possibility to transit silently to itself, also has the same traces with the same continuations as π, just that the traces have the output action (sd, i, j) prepended. $\mathbf{Inp}\, n$ has three arguments: the port i, a function h giving continuations for the cases of receiving known channels (in $\overline{0,n}$), and a continuation π for the case of receiving a new channel, identified as $n+1$. Save for these specifics, the definition of $\mathbf{Inp}\, n$ proceeds similarly to the one for output.

Replication. Notice that the π-calculus considered here has *unguarded* replication – this is impossible to model domain-theoretically for strong bisimilarity [7, 18], due to the resulted infinite branching. Since weak bisimilarity is infinitely branching anyway, here unguarded replication does not raise new fundamental problems – however, unguardedness does catch coalgebra somewhat off guard, since, e.g., an equation like $!P = !P|P$, although has a *least* domain-theoretic solution, does not have a *unique* coalgebraic solution. We have to think of the *progress* of $!P$ instead. Progress of $!P$ along a trace w occurs necessarily from the parallel composition of an arbitrary number k of P-clones, whose w-continuation placed in parallel with the "factory" $!P$ yields an w-continuation of $!P$. In our domain, this leads to the following. Let $pcfg_n = (n, id_n, id_n, \emptyset)$, where $id_n = \lambda i.$ if $i \in \overline{0,n}$ then i else $\perp$ – this is the canonical configuration for composing in parallel two processes that share the public channels $\overline{0,n}$, but no private link. We define $\mathbf{Repl} : I\!N \to Proc \to Proc$ corecursively by $\mathbf{Repl}\ n\ \pi = \{(\epsilon, \mathbf{Repl}\, n\, \pi)\} \cup \{(w, \mathbf{Parc}(pcfg_n, \pi', \mathbf{Repl}\, n\, \pi)).\exists k.(w, \pi') \in \mathbf{Parc}^k(pcfg_n, \pi)\}$, where $\mathbf{Parc}^k(pcfg_n, \pi)$ is $\mathbf{Parc}(pcfg_n, \pi, ..., \mathbf{Parc}(pcfg_n, \pi, \pi)...)$ ("**Parc**" k times).

Interpreting the syntax. We interpret pterms in *Proc* via *n-environments* $\rho \in Env_n$, where Env_n is $Var \to \overline{0,n}$. The interpretation therefore depends on

the information given not only by a map ρ from variables to channels, but also by the number n indicating the available channels. Thus, the semantic map $[[\_]]$ has type $Pterm \to \prod_{n\in\mathbb{N}} Env_n \to Proc$ and is defined as follows (where, identifying functions with their graphs, we assume $Env_m \subseteq Env_n$ if $m \leq n$):

- $[[\overline{x}y.P]]\, n\, \rho = \mathbf{Out}\ n\, (\rho\, x, \rho\, y, [[P]]\, n\, \rho)$.
- $[[x(y).P]]\, n\, \rho = \mathbf{Inp}\ n\, (\rho\, x, \lambda j : \overline{0, n}.\ [[P]]\, n\, (\rho[y \leftarrow j]), [[P]]\, (n+1)\, (\rho[y \leftarrow n+1]))$.
- $[[\nu y.P]]\, n\, \rho = \mathbf{Parc}((n, id_n, \mathbb{1}, \{(n+1, 0)\}), [[P]]\, (n+1)\, \rho[y \leftarrow n+1], \mathbf{Zero})$.
- $[[P \mid Q]]\, n\, \rho = \mathbf{Parc}(pcfg_n, [[P]]\, n\, \rho, [[Q]]\, n\, \rho)$.
- $[[!\, P]]\, n\, \rho = \mathbf{Repl}\ n\, ([[P]]\, n\, \rho)$.

The interpretations for input, output and replication should be clear. $\mid$ and $\nu y.\_$ are both interpreted using our general operator **Parc**, by choosing the configuration appropriately: for $\mid$, we use the already discussed standard configuration $pcfg_n$; for ν, in the context of the known channels being $\overline{0, n+1}$, we restrict on the last channel, $n+1$ – to obtain this behavior, we create a configuration where the to-be-restricted channel $n+1$ is part of a private link, $(n+1, 0)$, with **Zero** (where the choice (here 0) for the channel of **Zero** to participate in the link is immaterial, since **Zero** is inert) and the other channels, $\overline{0, n}$, receive global public channels via the identity function.

Full Abstraction Theorem. The denotation map captures bisimilarity congruence: $P \equiv_{OS} Q$ iff $[[P]] = [[Q]]$. Then, as expected, *fixing* (distinct denotations for) *the free variables* (through a fixed injective environment), we also capture bisimilarity: $P \sim_{OS} Q$ iff $[[P]]\, n\, \rho = [[Q]]\, n\, \rho$ for some n and $\rho \in Env_n$ injective.

The proof of full abstraction, although technically involved, follows naturally the interpretation we have given to configurations and the view on processes modulo bisimilarity as multi-step processes. We illustrate the main ideas of the proof on the example from the beginning of this section (referring to the items from there: P_1, π_1, c, c', etc.) In OS, we can derive $\dfrac{P_1 \xrightarrow{\overline{x}(y)} P_1' \quad P_2 \xrightarrow{x(y)} P_2'}{P_1|P_2 \xrightarrow{\tau} \nu y.(P_1'|\, P_2')}$ and $\dfrac{P_1' \xrightarrow{y(z)} P_1'' \qquad P_2' \xrightarrow{\overline{y}x} P_2''}{\nu y.(P_1'|P_2') \xrightarrow{\tau} \nu y.(P_1''[x/z]|\, P_2'')}$ (where the second derivation involves two rules).

Such derivations can be faithfully simulated, *via environments*, by derivations in a system obtained from asking the semantic processes to unfold their one-step deep continuations, where the continuations of the composites depend on the continuations of the components according to the definitions of the semantic operations. Here, (keeping the environment implicit) the corresponding "semantic" derivations are $\dfrac{\pi_1 \xrightarrow{(\mathsf{sd},i,j_1)} \pi_1' \qquad \pi_2 \xrightarrow{(\mathsf{rc},i,j_2)} \pi_2'}{\pi_1|_c\pi_2 \xrightarrow{\epsilon} \pi_1'|_{c'}\pi_2'}$ and $\dfrac{\pi_1' \xrightarrow{(\mathsf{rc},j_1,i)} \pi_1'' \qquad \pi_2' \xrightarrow{(\mathsf{sd},j_2,i)} \pi_2''}{\pi_1'|_{c'}\pi_2' \xrightarrow{\epsilon} \pi_1''|_{c'}\pi_2''}$ (where we wrote $\pi|_d\pi'$ instead of $Parc(d, \pi, \pi')$). On the other hand, the identity of our semantic items is determined by the continuations after arbitrarily long traces of actions, so that, if we want the semantic transition system to capture this identity, we would need to also consider chains of derivations directly, as in: $\dfrac{\pi_1 \xrightarrow{(\mathsf{sd},i,j_1)\#(\mathsf{rc},j_1,i)} \pi_1'' \qquad \pi_2 \xrightarrow{(\mathsf{rc},i,j_2)\#(\mathsf{sd},j_2,i)} \pi_2''}{\pi_1|_c\pi_2 \xrightarrow{\epsilon} \pi_1''|_{c'}\pi_2''}$ (where, by the definition of *par*, we have $(c', \epsilon) \in par(c, (\mathsf{sd}, i, j_1)\#(\mathsf{rc}, j_1, i), (\mathsf{rc}, i, j_2)\#(\mathsf{sd}, j_2, i)))$.

The proof applies the above observations as follows. First we define a (syntactic) system, CS, with concrete actions and configurations (like the ones in the

semantic domain), where one derives single-step actions. It is shown that this system simulates faithfully, via environments, the system OS. Then we show that weak bisimilarity of CS, roughly corresponding to a version of CS with traces of actions, determines by finality a notion of a process which coincides with our semantic processes in *Proc*. These two facts combined yield the result.

4 Conclusions and related work

We have defined a denotational semantics for the π-calculus under weak early bisimilarity which is both *fully abstract* and *compositional*. Other works on the denotation of π and related calculi either do not feature compositionality [10, 11] or cover only strong bisimilarity and related strong equivalencies [7, 18, 5, 19]. Moreover, previous approaches to the semantics of weak bisimilarity in name-free calculi such as CCS [3, 15] seem to be less satisfactory than the corresponding adaptation of our approach to these calculi. (See Appendix C for details.) Our technique of combining traces with coalgebra seems able to capture weak bisimilarity of a wide range of process calculi. Also, since the difference between early and late bisimilarity in the π-calculus can be regarded as a matter of granularity, by distinguishing between processes and abstractions (semantically, functions from channels to processes) one could use our technique to deal with late bisimilarity too. With a bit of extra effort, we could have finetuned our definitions into domain-theoretic ones using Abramski powerdomains instead of powersets, along the lines of [7, 18]. However, our semantics would then have not captured bisimilarity, but the weaker relation of having all finite subtrees bisimilar [1].

On the name bookkeeping side, our notion of configuration can be seen as reminiscent of the swapping and shifting operators used for keeping the channels consistently sorted in [7] – here we use configurations instead of allowing (the denotation of) the term to grow with swaps and shifts as computation proceeds; since here we need to deal with whole traces rather than individual actions like in [7], it is more manageable to delay, via configurations, the enforcement of channel-name consistency until the time of the action rather than applying it early via shifting and swapping. Other precursors of our configurations can be found, in a purely operational form, in [6, 9] (among other places).

References

1. S. Abramski. A domain equation for bisimulation. *Inf. Comput.*, 92(2):161–218, 1991.
2. P. Aczel. *Non-Well-Founded Sets*. Stanford, 1988.
3. P. Aczel. Final universes of processes. In *MFPS'93*, pages 1–28, 1993.
4. M. Baldamus, J. Parrow, and B. Victor. A fully abstract encoding of the *pi*-calculus with data terms. In *ICALP'05*, pages 1202–1213, 2005.
5. M. Baldamus and T. Stauner. Modifying esterel concepts to model hybrid systems. *Electr. Notes Theor. Comput. Sci.*, 65(5), 2002.
6. G. L. Ferrari, U. Montanari, and P. Quaglia. A pi-calculus with explicit substitutions. *Theor. Comput. Sci.*, 168(1):53–103, 1996.
7. M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *LICS'96*, pages 43–54, 1996.
8. M. Hennessy. A fully abstract denotational semantics for the π-calculus. *Theor. Comput. Sci.*, 278(1-2):53–89, 2002.
9. D. Hirschkoff. Handling substitutions explicitely in the pi-calculus. Explicit Substitutions: Theory and Applications to Programs and Proofs, 1999.

10. F. Honsell, M. Lenisa, U. Montanari, and M. Pistore. Final semantics for the pi-calculus. In *PROCOMET'98*, pages 225–243, 1998.
11. M. Lenisa. *Themes in Final Semantics*. Dipartimento di Informatica, Universita' di Pisa, TD 6, 1998.
12. R. Milner. *Communication and concurrency*. Prentice Hall, 1989.
13. R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge, 2001.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Inf. Comput.*, 100(1):1–77, 1992.
15. J. J. M. M. Rutten. Processes as terms: Non-well-founded models for bisimulation. *Math. Struct. Comp. Sci.*, 2(3):257–275, 1992.
16. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
17. D. Sangiorgi and D. Walker. *The π-calculus. A theory of mobile processes*. Cambridge, 2001.
18. I. Stark. A fully-abstract domain model for the π-calculus. In *LICS'96*, pages 36–42, 1996.
19. S. Staton. *Name-passing process calculi: operational models and structural operational semantics*. PhD thesis, University of Cambridge, 2007.
20. D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *LICS'97*, pages 280–291, 1997.

THE APPENDIX

Here is the structure of this appendix. Section A discusses semantic domains for weak bisimilarity in general, extending Section 2 of the main paper with proofs and more details. Section B contains the proof of well-defined-ness of the semantic operations in our π-calculus domain *Proc* (introduced in the main paper in Section 3) and the proof development of the full abstraction result (stated also in Section 3). Section C elaborates on related work, extending Section 4 of the main paper in two directions: by providing some technical details regarding more closely related work and by commenting briefly on less related work.

A More details about domains for weak bisimilarity in general

This section is an extended version of Section 2, containing more details and proofs of the stated facts.

Cls denotes the category of classes and functions between classes. In what follows, we shall employ basic concepts and results about coalgebras [22] and non-well-founded sets [2]. We only recall here a couple of definitions from coalgebra. Given a functor $F : \mathit{Cls} \to \mathit{Cls}$, an F-coalgebra is a pair $(A, \alpha : A \to F\,A)$. A *morphism* between two F-coalgebras (A, α) and (B, β) is a map $f : A \to B$ such that $(F\,f) \circ \alpha = \beta \circ f$. A *stable part* of an F-coalgebra (A, α) is a subclass $X \subseteq A$ such that $\forall x \in X.\, \alpha\, x \in F\,X$; a stable part X yields a *subcoalgebra* $(X, \alpha|_X)$. Given a fixed set Z, the functor $\mathcal{P}(Z \times \_)$ on *Cls* is defined as expected: it maps each class X to $\mathcal{P}(Z \times X)$ and each function $f : X \to Y$ to $\lambda K.\, \{(z, f\,x).\, (z, x) \in K\}$. Fix *Act*, a set of items called *actions*. Recall that $(\mathit{Act}^*, \#, \epsilon)$ is the monoid of sequences (or words) of actions. We shall only be interested in coalgebras for two functors: $\mathcal{P}((\mathit{Act} \cup \{\epsilon\}) \times \_)$ and $\mathcal{P}(\mathit{Act}^* \times \_)$; we call these *one-step coalgebras* and *multi-step coalgebras*, respectively.

Fully abstract domains of processes under strong bisimilarity are typically modeled as final one-step coalgebras. However, if the desired process identity is (weak) bisimilarity, then (also keeping in mind that operations such as parallel composition need to take a deeper, multi-step look into the argument processes) it is more natural to consider a suitable *multi-step* coalgebra as the domain. But on the other hand, we would like to keep sight of the fact that bisimilarity is a weakening of strong bisimilarity by internalizing the τ actions, in particular, to be able to infer bisimilarity from strong bisimilarity without any detour, and also to infer bisimilarity as usual, by showing how to simulate single steps only (by multisteps). These lead to the following constructions.

Let *Preproc*, the class of *preprocesses*, be the largest class X satisfying the equation $X = \mathcal{P}(\mathit{Act}^* \times X)$. (Such a class exists by the Knasser-Tarski theorem.) Moreover, since we work under the AFA hypothesis, $(\mathit{Preproc}, 1_{\mathit{Preproc}})$ is the final multi-step coalgebra [2].

Given a multi-step coalgebra (D, δ), an element $d \in D$ is said to be: *reflexive*, if $(\epsilon, d) \in \delta\, d$; *transitive*, if $\forall w, w', d', d''.\ (w, d') \in \delta\, d \wedge (w', d'') \in \delta\, d' \Rightarrow (w \# w', d'') \in \delta\, d$; *prefix-closed*, if $\forall w, w', d''.\, (w \# w', d'') \in \delta\, d \Rightarrow (\exists d'.\, (w, d') \in \delta\, d \wedge (w', d'') \in \delta\, d')$; *monoidal*, if it is reflexive, transitive and prefix-closed. (D, δ) is said to be *monoidal* if all elements of D are monoidal.

Let *Proc*, the class of *processes*, be the stable part of the multi-step coalgebra $(\mathit{Preproc}, 1_{\mathit{Preproc}})$ cogenerated by the class of all monoidal preprocesses. This notion of process encompasses two ideas. First, processes have a *linear-time* structure which is compatible with the monoid of action sequences (via reflexivity and transitivity) and has no discontinuities (via prefix-closeness). Second, processes have the above linear-time properties preserved by the *branching-time*, coalgebraic structure – one should think of a process as an *hereditarily* monoidal preprocess, that is, a preprocess π such that π is monoidal and the

preprocesses from all its arbitrarily deep unfoldings are so. (Notice that prefix-closeness is hereditary by definition; this is not the case for reflexivity and transitivity, however.) The properties of the linear-time structure, making sense for any transition-system labeled with sequences of actions (i.e., for any multi-step coalgebra), are the essential features of weak bismilarity. Our choice to impose these properties hereditarily deep for elements in the final multi-step coalgebra has the following consequence:

Proposition 1 $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$ *is the final monoidal multi-step coalgbebra.*

Proof. Although it may be possible to infer this proposition, after a suitable encoding, from more general results regarding modal logic for coalgebras [20] (or, alternatively, from the results on hidden algebra [19, 6]), we prefer to give the (very simple) direct proof here. Fix a reflexive-transitive multi-step coalgebra $(D, \delta : D \to \mathcal{P}(Act^* \times D))$. Let $h : D \to Preproc$ be the unique multi-step coalgebra morphism given by the finality of $(Preproc, 1_{Preproc})$. Then $Im(h)$ is known (and easily seen) to be a a stable part of *Preproc*.

We now show that all the elements of $Im(h)$ are monoidal. Fix $\pi \in Im(h)$. Then $\pi = h\,d$ for some $d \in D$.
- Reflexiveness of π: Since $(\epsilon, d) \in \delta\,d$ (by the transitivity of d) and h is a morphism, it follows that $(\epsilon, h\,d) \in 1_{Preproc}(h\,d)$, i.e., $(\epsilon, \pi) \in \pi$.
- Transitivity of π: Fix w, w', π', π'' and assume $(w, \pi') \in \pi$ and $(w', \pi'') \in \pi'$. Since h is a morphism, there exists $d' \in D$ such that $(w, d') \in \delta\,d$ and $h\,d' = \pi'$, and then there exists $d'' \in D$ such that $(w', d'') \in \delta\,d'$ and $h\,d'' = \pi''$. With the assumption that d is transitive, we obtain $(w\#w', d'') \in \delta\,d$, hence, with the fact that h is a morphism, $(w\#w', \pi'') \in 1_{Preproc}(h\,d)$, i.e., $(w\#w', \pi'') \in \pi$.
- Prefix-closeness of π: Fix w, w', π'' and assume $(w\#w', \pi'') \in \pi$. With the fact that h is a morphism, we have that there exists d'' such that $h\,d'' = \pi''$ and $(w\#w', d'') \in \delta\,d$. With the assumption that d is prefix-closed, we have that there exists d' such that $(w, d') \in \delta\,d$ and $(w', d'') \in \delta\,d'$. With the fact that h is a morphism, we obtain $(w, h\,d') \in 1_{Preproc}(h\,d)$ and $(w', h\,d'') \in 1_{Preproc}(h\,d')$, i.e., $(w, h\,d') \in \pi$ and $(w', \pi'') \in \pi''$. Therefore, $h\,d'$ is the desired element in $Im(h)$.

Since $Im(h)$ is a coalgebra and consists of monoidal elements only, it follows that $Im(h) \subseteq Proc$, making h a multi-step coalgebra morphism between (D, δ) and $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$. We thus proved the existence of the morphism. Uniqueness follows immediately by the finality of $(Preproc, 1_{Preproc})$ and the fact that the inclusion $Proc \hookrightarrow Preproc$ is a multi-step coalgebra morphism. □

The above proposition yields standardly a *corecursive* definition principle, employed in Section 3 for giving semantics to the π-calculus: to define a (parameterized) operation $f : Param \times Proc^n \to Proc$, it suffices to organize $Param \times Proc^n$ as a monoidal multi-step coalgebra by defining $\delta : Param \times Proc^n \to \mathcal{P}(Act^* \times (Param \times Proc^n))$ (and then take f to be the unique mor-

phism between $(Param \times Proc^n, \delta)$ and $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$ given by the finality of the latter).

Sometimes it is useful to employ *Proc-parameterized corecursion* in definitions (where elements of *Proc* appear as parameters at the corecursive calls). This type of corecursion is reducible to the previous corecursion principle, as shown next by an adaptation (to cope with monoidality) of known coalgebraic constructions and results regarding parameterized corecursion.

A *parameterized multi-step coalgebra* is a pair (D, δ) consisting of a class D and a map $\delta : D \to \mathcal{P}(Act^* \times (D + Proc))$, where $D + Proc$ is the direct sum (disjoint union) of D and *Proc*, consisting of items of the forms $(0, d)$ and $(1, \pi)$. (Notice that a parameterized multi-step coalgebra is *not* a multi-step coalgebra.) Given a parameterized multi-step coalgebra, (D, δ), an element $d \in D$ is said to be: *reflexive*, if $(0, d) \in \delta\, d$; 00-*transitive*, if $(w, (0, d')) \in \delta\, d \wedge (w', (0, d'')) \in \delta\, d' \Rightarrow (w \# w', (0, d'')) \in \delta\, d$; 01-*transitive*, if $(w, (0, d')) \in \delta\, d \wedge (w', (1, \pi)) \in \delta\, d' \Rightarrow (w \# w', (1, \pi)) \in \delta\, d$; 11-*transitive*, if $(w, (1, \pi)) \in \delta\, d \wedge (w', \pi') \in \pi \Rightarrow (w \# w', (1, \pi')) \in \delta\, d$; 0-*prefix-closed*, if $(w \# w', (0, d'')) \in \delta\, d \Rightarrow (\exists d'.\, (w, (0, d')) \in \delta\, d \wedge (w', (0, d'')) \in \delta\, d')$; 1-*prefix-closed*, if $(w \# w', (1, \pi')) \in \delta\, d \Rightarrow ((\exists d'.\, (w, (0, d')) \in \delta\, d \wedge (w', (1, \pi')) \in \delta\, d') \vee (\exists \pi.\, (w, (1, \pi)) \in \delta\, d \wedge (w', \pi') \in \pi))$; *monoidal*, if it is reflexive, [00, 01, 11]-transitive, and [0, 1]-prefix-closed. (D, δ) is said to be *monoidal* if all elements of D are monoidal.

In the next proposition and in its proof, we write F for the functor $\mathcal{P}(Act^* \times \_)$ and *unf* (meaning "unfold") for the inclusion map $Proc \hookrightarrow F(Proc)$. Given two classes A and B, we write $inl^{A,B}$ and $inr^{A,B}$ for the two structural morphisms $A \to A+B$ and $B \to A+B$, and, given $f : A \to C$ and $g : B \to C$, we write $\langle f, g \rangle$ for the unique map $h : A + B \to C$ such that $h \circ inl^{A,B} = f$ and $h \circ inr^{A,B} = g$.

Proposition 2 *Let (D, δ) be a parameterized monoidal multi-step coalgebra. Then there exists a unique map $f : D \to Proc$ such that $unf \circ f = (F\langle f, 1_{Proc}\rangle) \circ \delta$.*

Proof. We write Consider the multi-step coalgebra $(D + Proc, \langle \delta, (F\, inr^{D,Proc}) \circ unf \,\rangle : D + Proc \to F(D + Proc)$. We show that it is monoidal. Fix $x \in D + Proc$. If x has the form $(1, \pi)$ for $\pi \in Proc$, then $\langle \delta, (F\, inr^{D,Proc}) \circ unf \,\rangle\, x = ((F\, inr^{D,Proc}) \circ unf\,)\, \pi = (F\, inr^{D,Proc})\, \pi = \{(w, (1, \pi')).\ (w, \pi') \in \pi\}$, and the desired monoidality of x in the multi-step coalgebra $D + Proc$ follows from the monoidality of π in *Proc*. On the other hand, if x has the form $(0, d)$ for $d \in D$, then the desired monoidality of x in the multi-step coalgebra $D + Proc$ follows from the monoidality of x in the parameterized multi-step coalgebra D.

Form the above and the finality of *Proc*, it follows that there exists a unique map $g : D + Proc \to Proc$ such that $(F\, g) \circ \langle \delta, (F\, inr^{D,Proc}) \circ unf \,\rangle = unf \circ g$. Moreover, a diagram chase (with applying the universality of sums and the functoriality of F) shows that there exists a bijective correspondence between:
- maps $g : D + Proc \to Proc$ such that $(F\, g) \circ \langle \delta, (F\, inr^{D,Proc}) \circ unf \,\rangle = unf \circ g$, on the one hand;
- maps $f : D \to Proc$ such that $unf \circ f = (F\langle f, 1_{Proc}\rangle) \circ \delta$, on the other hand.
(The correspondence is given by $g \mapsto g \circ inl^{D,Proc}$ and $f \mapsto \langle f, 1_{Proc}\rangle$.)

This proves the desired existence and uniqueness of f. □

The fact that $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$ is *simple* (being a subcoalgebra of the (absolutely) final coalgebra), yields standardly a proof principle. Assume $\theta \subseteq Proc \times Proc$ is a strong bisimulation on $Proc$ (regarded as an Act^*-labeled transition system),[1] in that the following hold for all $(\pi, \pi') \in \theta$: **(i)** $\forall (w, \pi'') \in \pi. \exists \pi'''. (w, \pi''') \in \pi' \wedge (\pi'', \pi''') \in \theta$; **(ii)** $\forall (w, \pi'') \in \pi'. \exists \pi'''. (w, \pi''') \in \pi \wedge (\pi''', \pi'') \in \theta$. Then $\pi = \pi'$ for all $(\pi, \pi') \in \theta$.

Thanks to the affinity between $Proc$ and the monoidal structure of Act^*, we also have a simpler (but equally powerful) proof principle which reflects more closely the traditional way of dealing with weak bisimilarity, by showing how to simulate single actions only:

Proposition 3 *Let $\theta \subseteq Proc \times Proc$ be such that the following hold for all $(\pi, \pi') \in \theta$:* **(i)** $\forall (w, \pi'') \in \pi. |w| \in \{0, 1\} \Rightarrow (\exists \pi'''. (w, \pi''') \in \pi' \wedge (\pi'', \pi''') \in \theta)$*;* **(ii)** $\forall (w, \pi'') \in \pi'. |w| \in \{0, 1\} \Rightarrow (\exists \pi'''. (w, \pi''') \in \pi \wedge (\pi''', \pi'') \in \theta)$*. Then $\pi = \pi'$ for all $(\pi, \pi') \in \theta$.*

Proof. It suffices to prove, by induction on n, that the following two properties hold:
- $\forall (\pi, \pi') \in \theta. \forall (w, \pi'') \in \pi. |w| = n \Rightarrow (\exists \pi'''. (w, \pi''') \in \pi' \wedge (\pi'', \pi''') \in \theta)$;
- $\forall (\pi, \pi') \in \theta. \forall (w, \pi'') \in \pi'. |w| = n \Rightarrow (\exists \pi'''. (w, \pi''') \in \pi \wedge (\pi''', \pi'') \in \theta)$.

(Indeed, this would mean that θ is a strong bisimilarity on $Proc$, hence included in equality.)

The cases when $n \in \{0, 1\}$ are handled by assumption. So assume $n \geq 2$. Fix w, π, π', π'' such that $|w| = n$, $(\pi, \pi') \in \theta$ and $(w, \pi'') \in \pi$. Then w has the form $w' \# a$ with $a \in Act$, $w' \in Act^*$ and $|w'| = n - 1$. By the prefix-closeness of π, there exists π_0 such that $(w', \pi_0) \in \pi$ and $(a, \pi'') \in \pi_0$. With IH for w', there exists π_1 such that $(\pi_1, w') \in \pi'$ and $(\pi_0, \pi_1) \in \theta$. With IH for a, there exists π''' such that $(\pi''', a) \in \pi_1$ and $(\pi'', \pi''') \in \pi'$. With the transitivity of π', we obtain $(w' \# a, \pi''') \in \pi'$, i.e., $(w, \pi''') \in \pi'$. Hence π''' is the desired process. The other (symmetric) property follows similarly. □

The proof principle from Proposition 3 may appear, at first, as if employing *strong* bisimulation (w.r.t. *single* actions) – but remember that processes have the monoidal properties of action sequences absorbed in their structure; in particular: **(1)** ϵ is identified with (any sequence in the language) ϵ^*, thus meaning "zero or more silent steps"; **(2)** the single action a is identified with $\epsilon^* \# a \# \epsilon^*$, thus meaning "$a$ preceded and succeeded by zero or more silent steps". Therefore, *weak* bisimulation is what we really deal with here, strong bisimulation being only a particular case.

The fact that each process is uniquely identified by its behavior w.r.t. sequences of length 0 or 1 (i.e., elements of $Act \cup \{\epsilon\}$) suggests a more compact representation of the domain of processes as a one-step coalgebra. As already

[1] Remember that strong bisimilarity w.r.t. traces of actions (with the silent actions absorbed) is in effect weak bisimilarity w.r.t. single actions.

mentioned, the choice of the (absolutely) final one-step coalgebra as the semantic domain for strong bisimulation typically already yields the desired properties (in particular, full abstraction for various operational semantics [21]). However, here, since we are after (weak) bismilarity, we shall require that processes, even in this more compact representation with single steps, retain the affinity with the monoidal structure on sequences of actions – this means here that zero or more ϵ-steps can always be appended and/or prepended "silently", yielding a property similar to monoidality that we shall call ϵ-monoidality. (Although the constructions below are not needed for our full abstraction result, they are nevertheless useful for placing our monoidal multi-step coalgebra in a context that clarifies its connection with the traditional view on weak bisimilarity, as well as its advantage over the latter view w.r.t. compositionality.)

Let *Cpreproc*, the class of *compact (representations of) preprocesses*, be the largest class X satisfying the equation $X = \mathcal{P}((Act \cup \{\epsilon\}) \times X)$. Similarly to the case of *Preproc*, we have that $(Cpreproc, 1_{Cpreproc})$ is the final one-step coalgebra. Given a one-step coalgebra (D, δ), an element $d \in D$ is said to be: *ϵ-reflexive*, if $(\epsilon, d) \in \delta\, d$; *$\epsilon$-transitive*, if $\forall d', d'' \in D.\ (\epsilon, d') \in \delta\, d \wedge (\epsilon, d'') \in \delta\, d' \Rightarrow (\epsilon, d'') \in \delta\, d$; *$\epsilon$-loud-transitive*, if $\forall d', d'', d''' \in D. \forall a \in Act.\ (\epsilon, d') \in \delta\, d \wedge (a, d'') \in \delta\, d' \wedge (\epsilon, d''') \in \delta\, d'' \Rightarrow (a, d''') \in \delta\, d$; *$\epsilon$-monoidal*, if it is ϵ-reflexive, ϵ-transitive and ϵ-loud-transitive. (D, δ) is said to be *ϵ-monoidal* if all elements of D are ϵ-monoidal.

Let *Cproc*, the class of *compact (representations of) processes*, be the stable part of the one-step coalgebra $(Cpreproc, 1_{Cpreproc})$ cogenerated by the class of all ϵ-monoidal compact preprocesses.

Proposition 4 *$(Cproc, Cproc \hookrightarrow \mathcal{P}((Act \cup \{\epsilon\}) \times Cproc))$ is the final ϵ-monoidal one-step coalgebra.*

Proof. The proof can be performed very similarly to that of Proposition 1, and is omitted. □

In what follows, we shall consistently use π for processes and σ for compact processes. We shall define two maps, $pack : Proc \to Cproc$ and $unpack : Cproc \to Proc$, for moving back and forth between a process and its compact representation, and shall prove that they are inverse to each other. Essentially, $pack\ \pi$ is obtained by retaining from π (and from all its arbitrarily deep continuations) only the one-step continuations, while $unpack\ \sigma$ is obtained by expanding σ (and all its continuations) along all possible sequences of steps.

To define *pack* using the finality of *Cproc*, we organize *Proc* as an ϵ-monoidal one-step coalgebra by defining $\delta : Proc \to \mathcal{P}((Act \cup \{\epsilon\}) \times Proc)$ as follows: $\delta\, \pi = \{(w, \pi').\ (w, \pi') \in \pi \wedge w \in Act \cup \{\epsilon\}\}$.

Lemma 5 *$(Proc, \delta)$ is ϵ-monoidal (as a one-step coalgebra).*

Proof. Fix $\pi \in Proc$. The desired facts about π as an item in the one-step coalgebra $(Proc, \delta)$ follow immediately from corresponding facts about π as an item in the monoidal multi-step coalgebra $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$:
- ϵ-reflexivity of π, from the reflexivity of π.

- ϵ-transitivity of π, from the transitivity of π.
- ϵ-loud-transitivity of π, from the transitivity of π (applied twice). □

We obtain a unique one-step coalgebra morphism, *pack*, between $(Proc, \delta)$ and $(Cproc, Cproc \hookrightarrow \mathcal{P}((Act \cup \{\epsilon\}) \times Cproc))$. In other words, *pack* is defined to be the unique map $Proc \to Cproc$ such that, for all π, $pack\,\pi = \{(w, pack\,\pi').\ (w, \pi') \in \pi \wedge w \in Act \cup \{\epsilon\}\}$.

Similarly, to define *unpack* using the finality of *Proc*, we organize *Cproc* as a monoidal multi-step coalgebra by defining $\gamma : Cproc \to \mathcal{P}(Act^* \times Cproc)$ as follows: $\gamma\,\sigma = \{(w_1\#\ldots\#w_n, \sigma').\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\, w_i \in Act \cup \{\epsilon\}) \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\, \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\, (w_i, \sigma_{i+1}) \in \sigma_i))\}$.

Lemma 6 *$(Cproc, \gamma)$ is monoidal (as a multi-step coalgebra).*

Proof. Immediate from the definition of γ. □

We obtain a unique multi-step coalgebra morphism, *unpack*, between $(Cproc, \gamma)$ and $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$. In other words, *unpack* is defined to be the unique map $Cproc \to Proc$ such that, for all σ,
$unpack\,\sigma = \{(w_1\#\ldots\#w_n, unpack\,\sigma').\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\, w_i \in Act \cup \{\epsilon\}) \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\, \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\, (w_i, \sigma_{i+1}) \in \sigma_i))\}$.

Proposition 7 *pack and unpack are mutually inverse bijections.*

Proof. To prove that $pack \circ unpack = 1_{Cproc}$, it suffices (by the simplicity of the one-step coalgebra $(Cproc, Cproc \hookrightarrow \mathcal{P}((Act \cup \{\epsilon\}) \times Cproc))$), to show that $pack \circ unpack$ satisfies the property that identifies the morphism 1_{Cproc} uniquely, i.e., that, for all σ, $pack(unpack\,\sigma) = \{(w, pack(unpack\,\sigma')).\ (w, \sigma') \in \sigma\}$. This is established by the following chain of equalities:
$pack(unpack\,\sigma) =$ (by the definition of *pack*)
$\{(w, pack\,\pi').\ w \in Act \cup \{\epsilon\} \wedge (w, \pi') \in unpack\,\sigma\} =$ (by the def. of *unpack*)
$\{(w, pack\,\pi').\ w \in Act \cup \{\epsilon\} \wedge (\exists \sigma'.\, \pi' = unpack\,\sigma' \wedge (\exists n \in I\!N.\, \exists w_1, \ldots, w_n \in Act \cup \{\epsilon\}.\, w = w_1\#\ldots\#w_n \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\, \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\, (w_i, \sigma_{i+1}) \in \sigma_i))))\} =$
$\{(w_1\#\ldots\#w_n, pack(unpack\,\sigma')).\ n \in I\!N \wedge w_1\#\ldots\#w_n \in Act \cup \{\epsilon\} \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\, \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\, (w_i, \sigma_{i+1}) \in \sigma_i))\} =$
(by case-analyzing whether $w_1\#\ldots\#w_n$ is ϵ (case in which all w_i-s are ϵ) or is in *Act* (case in which all w_i-s are ϵ, except for one of them which is in *Act*))
$\{(\epsilon, pack(unpack\ \sigma')).\ \exists n \in I\!N.\, \exists \sigma_1, \ldots, \sigma_{n+1}.\, \sigma_1 = \sigma \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\, (\epsilon, \sigma_{i+1}) \in \sigma_i)\} \cup$
$\{(a, pack(unpack\ \sigma')).\ \exists m, n \in I\!N.\, \exists \sigma_1, \ldots, \sigma_{m+n+2}.\, \sigma_1 = \sigma \wedge \sigma_{m+n+2} = \sigma' \wedge (\forall i \in \overline{1, m+n+2} \setminus \{m+1\}.\, (\epsilon, \sigma_{i+1}) \in \sigma_i) \wedge (a, \sigma_{m+2}) \in \sigma_{m+1}\} =$
(by the ϵ-monoidality of $(Cproc, Cproc \hookrightarrow \mathcal{P}((Act \cup \{\epsilon\}) \times Cproc))$
$\{(\epsilon, pack(unpack\,\sigma')).\ (\epsilon, \sigma') \in \sigma\} \cup \{(a, pack(unpack\,\sigma')).\ a \in Act \wedge (a, \sigma') \in \sigma\} =$
$\{(w, pack(unpack\,\sigma')).\ (w, \sigma') \in \sigma\}$.

To prove that $unpack \circ pack = 1_{Proc}$, it suffices (by the simplicity of the multi-step coalgebra $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Cproc))$), to show that $unpack \circ pack$

satisfies the property that identifies the morphism 1_{Proc} uniquely, i.e., that, for all π, $\mathit{unpack}(\mathit{pack}\ \pi) = \{(w, \mathit{unpack}(\mathit{pack}\ \pi')).\ (w, \pi') \in \pi\}$. This is established by the following chain of equalities:
$\mathit{unpack}(\mathit{pack}\ \pi) =$ (by the definition of *unpack*)
$\{(w_1\# \ldots \#w_n, \mathit{unpack}\ \sigma').\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\ w_i \in \mathit{Act} \cup \{\epsilon\}) \wedge (\exists \sigma_1, \ldots, \sigma_{n+1}.\ \sigma_1 = \mathit{pack}\ \pi \wedge \sigma_{n+1} = \sigma' \wedge (\forall i \in \overline{1,n}.\ (w_i, \sigma_{i+1}) \in \sigma_i))\} =$
(by the definition of *pack*, applied n times)
$\{(w_1\# \ldots \#w_n, \mathit{unpack}(\mathit{pack}\ \pi')).\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\ w_i \in \mathit{Act} \cup \{\epsilon\}) \wedge (\exists \pi_1, \ldots, \pi_{n+1}.\ \pi_1 = \pi \wedge \pi_{n+1} = \pi' \wedge (\forall i \in \overline{1,n}.\ (w_i, \pi_{i+1}) \in \pi_i))\} =$
(by the monoidality of $(\mathit{Proc}, \mathit{Proc} \hookrightarrow \mathcal{P}(\mathit{Act}^* \times \mathit{Cproc}))$, where n applications of reflexivity and transitivity are needed for "$\subseteq$" and n applications of prefix-closeness are needed for "$\supseteq$")
$\{(w_1\# \ldots \#w_n, \mathit{unpack}(\mathit{pack}\ \pi')).\ n \in I\!N \wedge (\forall i \in \overline{1,n}.\ w_i \in \mathit{Act} \cup \{\epsilon\}) \wedge (w_1\# \ldots \#w_n, \pi') \in \pi\} =$
$\{(w, \mathit{unpack}(\mathit{pack}\ \pi')).\ (w, \pi') \in \pi\}.$ □

Since inhabitants of final coalgebras may be (somewhat imprecisely, but) conveniently regarded as infinite trees, Proposition 7 appears to be exploiting the agreement between two alternative ways of presenting infinite trees: by *corecursive* definitions (in the one-step coalgebra *Cproc*) and as *sets of (possibly infinite) traces* (obtained from forgetting the "nodes", in the multi-step coalgebra *Proc*). Recall however that the sets of traces alone (even including partial traces, as well as infinite traces obtained as limits) provide a whole less amount of information than the one needed to establish the identity of a process that has produced those traces (to see this, it suffices to consider the classic simple example of two processes π and π', where $\pi = \{(a, \pi_1), (a, \pi_2)\}$, $\pi_1 = \{(b, \emptyset)\}$, $\pi_2 = \{(c, \emptyset)\}$, $\pi' = \{(a, \pi_3)\}$, $\pi_3 = \{(b, \emptyset), (c, \emptyset)\}$). In effect, we argue below that our multi-step process model, that essentially puts together a trace semantics with a coalgebraic semantics, although redundant stricto sensu (since the one-step compact process model has been seen to keep the same information), is not redundant w.r.t. the requirements for compositional semantics, but keeps only the minimal coalgebraic information needed there.

The compact processes are coalgebraically only one-step deep, hence correspond more directly to the traditional operational semantics presentation of process calculi. However, in our context of (weak) bisimilarity, these compact one-step representations have a salient disadvantage compared to (multi-step) processes: crucial operations in process calculi, such as parallel composition and iteration, are not definable purely coalgebraically on compact processes, since the one-step coalgebra falls short on providing the means of describing the composite process.

To illustrate this point, assume that Act is endowed with a bijection $^- : \mathit{Act} \to \mathit{Act}$ which is involutive (in that $\overline{\overline{a}} = a$ for all $a \in \mathit{Act}$), and say we would like to define CCS-like parallel composition on processes (thus, we assume that a and $\overline{a}$ are to be interpreted as complementary actions, whose synchronization yields a silent action). The main task in front of us is to show how sequences of actions interact, possibly nondeterministically. Knowing how single actions interact, and

assuming an interleaving semantics, we obtain the following definition of the *parallel composition* (or *synchronized shuffle*) $| : Act^* \times Act^* \to \mathcal{P}(Act^*)$ (where we extend $\#$ to sets of sequences in the obvious way): **(i)** $\epsilon|\epsilon = \{\epsilon\}$; **(ii)** if $(w_1, w_2) \neq (\epsilon, \epsilon)$, then $w_1|w_2 = \{a \# w_1'|w_2 : w_1 = a \# w_1'\} \cup \{a \# w_1|w_2' : w_2 = a \# w_2'\} \cup \{w_1|w_2' : \exists a.\, w_1 = a \# w_1' \wedge w_2 = \overline{a} \# w_2'\}$.

Now parallel composition of processes, $| : Proc \times Proc \to Proc$, simply follows coinductively the interaction law prescribed by action sequence composition: $\pi_1|\pi_2 = \{(w, \pi_1'|\pi_2') : \exists w_1, w_2.\, w \in w_1|w_2 \wedge (w_1, \pi_1') \in \pi_1 \wedge (w_2, \pi_2') \in \pi_2\}$. More precisely, $|$ on processes is defined by the finality of $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$, organizing $Proc \times Proc$ as a monoidal multi-step coalgebra as follows: $\delta : Proc \times Proc \to \mathcal{P}(Act^* \times (Proc \times Proc))$, $\delta(\pi_1, \pi_2) = \{(w, (\pi_1', \pi_2')) : \exists w_1, w_2.\, w \in w_1|w_2 \wedge (w_1, \pi_1') \in \pi_1 \wedge (w_2, \pi_2') \in \pi_2\}$. To show that $(Proc \times Proc, \delta)$ is indeed monoidal, we first need a simple lemma about $|$ on actions:

Lemma 1. *(1)* $\epsilon \in \epsilon|\epsilon$.
(2) If $w \in w_1|w_2$ *and* $w' \in w_1'|w_2'$*, then* $w\#w' \in (w_1\#w_1')|(w_2\#w_2')$.
(3) If $w\#w' \in w_1''|w_2''$*, then there exist* w_1, w_1', w_2, w_2' *such that* $w_1'' = w_1\#w_1'$, $w_2'' = w_2\#w_2'$, $w \in w_1|w_2$*, and* $w' \in w_1'|w_2'$.

Proof. (1): Immediate
(2): Easy induction on the structures of w_1 and w_2
(3): Easy induction on the structures of w_1'' and w_2''. □

Lemma 2. $(Proc \times Proc, \delta)$ *is indeed monoidal, and therefore* $| : Proc \times Proc \to Proc$ *is well-defined by finality.*

Proof. The desired facts follow from Lemma 1: reflexivity from point (1) of that lemma, transitivity from point (2), and prefix-closeness from point (3). □

Notice the separation of concerns in our definition of $|$: first we dealt with action sequence combinatorics, so that aftewards the definition of parallel composition of processes was stated *purely coalgebraically*, by composing together the continuations of the components to yield all continuations of the composite. On the other hand, if trying to define parallel composition on *compact processes*, one finds the colagebraic one-step depth of the latter insufficient for describing even the one-step behavior of the composite – this is because a step of the composite $\sigma_1|\sigma_2$ may result from an *arbitrarily long* sequence of interactions between steps taken by continuations of σ_1 and σ_2. In fact, the reader is invited to try to define parallel composition on *Cproc* and to notice that any reasonable definition would essentially appeal to our multi-step domain *Proc*, unpacking the components σ_1 and σ_2, composing these unpacked versions in *Proc*, and then packing the result (i.e., the operation on *Cproc* would be essentially $pack((unpack\ \sigma_1)|(unpack\ \sigma_2))$). This shows, in our opinion, that *Proc*, and not *Cproc*, is the fundamental domain for compositional weak bisimilarity.

Even though one may argue that, via the bijections *pack* and *unpack*, *Proc* and *Cproc* are really the same domain, so that considering one or the other are two sides of the same coin, *Proc* and *Cproc* take nevertheless two *distinct views* to

processes, with the multi-step view brought by *Proc*, as explained above, allowing for a cleaner compositionality and separation on concerns when defining process composition. When the effect of actions becomes more complex, with possible changes of the communication topology, the multi-step view brings a much better insight into the semantics of a calculus under bisimilarity. Section 3 employs this view for the semantics of the π-calculus.

B Proof of full abstraction

The main purpose of this section is to give fairly detailed proofs of the following two theorems, expressing full abstraction of our semantic domain for OS early bisimilarity congruence and OS early bismimilarity, respectively.

Theorem 8 *For all pterms P and Q, $P \equiv_{OS} Q$ (i.e., P and Q are early bisimilarly congruent) iff $[[P]] = [[Q]]$.*

Theorem 9 *For all pterms P and Q, the following are equivalent:*
- $P \sim_{OS} Q$ (i.e., P are Q are early bisimilar).
- $[[P]]\, n\, \rho = [[Q]]\, n\, \rho$ for all *n and $\rho \in Env_n$ such that ρ is injective on the free variables of P and Q.*
- $[[P]]\, n\, \rho = [[Q]]\, n\, \rho$ for some *n and $\rho \in Env_n$ such that ρ is injective on the free variables of P and Q.*

Here is the overview of the proofs. First we define a transition system, CS, involving concrete actions, like the ones in Section 3, rather than generic actions like in the original calculus. CS will also employ parallel composition in configurations, behaving as prescribed by the clauses that define $\_ \bar{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\rightsquigarrow} \_$ from Section 3. We relate the original system OS with CS via a notion of environment that assigns names to concrete channels, as suggested in Section 3 for the example $(\nu y.\, \overline{x}(y).\, y(z).\, P_1'')|(x(y).\, \overline{y}x.\, P_2'')$ – we obtain a tight relationship between the bisimilarities in these two systems, OS and CS. Then we show that bisimilarity in CS is captured precisely by our denotational model. Putting these pieces together, full abstraction (for both $\equiv_{OS}$ and $\sim_{OS}$ follows).

Before we engage in the proof of full abstraction, we owe the reader a proof that the operations we have considered on the semantic domain *Proc* are well-defined. Indeed, the corecursive definitions of **Parc**, **Zero**, **Out**, **Inp** and **Repl** require a proof that the multi-step coalgebras and parameterized multi-step coalgebras involved in these definitions are monoidal.

Proposition 10 *The aforementioned operators on Proc are well-defined.*

Proof. For **Parc** and **Zero**, we implicitly employed the corecursive definition principle following from the finality of *Proc* among monoidal multi-setp coalgebras, while for **Out**, **Inp** and and **Repl** we used parameterized corecursion (Proposition 2 from Section A). Below we spell out the involved multi-step coalgebras and parmeterized multi-step coalgebras and show that they satisfy the required property, monoidality.

For the definition of the constant **Zero** $\in$ *Proc*, we (implicitly) used the singleton multi-step coalgebra $(\{*\}, \delta : \{*\} \to \mathcal{P}(Act^* \times \{*\}))$ where $\delta * = \{(\epsilon, *)\}$, which is clearly monoidal.

For the definition of **Out**, given $n \in I\!N$, $i, j \in \overline{1,n}$, and $\pi \in$ *Proc*, we used the singleton parameterized multi-step coalgebra $(\{*\}, \delta : \{*\} \to \mathcal{P}(Act^* \times (\{*\} + Proc)))$, where $\delta * = \{(\epsilon, (0, *))\} \cup \{((\mathsf{sd}, i, j)\#w, (1, \pi')).\ (w, \pi') \in \pi\}$.
(Thus, **Out** is defined as:

$\lambda n.\,\lambda(i,j,\pi).\ \mathit{fix}\ \pi''.\{(\epsilon,\pi'')\} \cup \{((\mathsf{sd},i,j)\#w,(1,\pi')).\ (w,\pi') \in \pi\}$, and *not* as:
$\lambda n.\,\lambda(i,j,\pi).$ let $f = \mathit{fix}\ f.\,\lambda\pi.\,\{(\epsilon,\pi)\} \cup \{((\mathsf{sd},i,j)\#w,(1,\pi')).\ (w,\pi') \in \pi\}$ in $f\,\pi$,
as one may be inclined to think at first sight. The latter would yield
$\mathbf{Out}\ n\,(i,j,\pi) = \{(\epsilon,\mathit{Out}\,n\,(i,j,\pi))\} \cup \{((\mathsf{sd},i,j)\#w,(1,\pi')).\ (w,\pi') \in \mathbf{Out}\ n\,(i,j,\pi)\}$,
which is *not* what we intended!)
We only need to show that the element $*$ is monoidal.
- Since the only element of $\delta *$ of the form $(w,(0,d))$ with $d \in \{*\}$ is $(\epsilon,*)$, it follows immediately that $*$ is reflexive, 00-transitive, 01-transitive and 0-prefix-closed.
- To check 11-transitivity, assume $(w,(1,\pi')) \in \delta *$ and $(w',\pi'') \in \pi'$. Then w has the form $(\mathsf{sd},i,j)\#w''$ with $(w'',\pi') \in \pi$. By the transitivity of π, we have that $(w''\#w',\pi'') \in \pi$, implying $((\mathsf{sd},i,j)\#w''\#w',(1,\pi'')) \in \delta *$, i.e., $(w\#w',(1,\pi'')) \in \delta *$, as desired.
- To check 1-prefix-closeness, assume $(w\#w',(1,\pi'')) \in \delta *$. If $w = \epsilon$, then $(w,(0,*)) \in \delta *$ and $(w',(1,\pi'')) \in \delta *$, hence $*$ is the desired item. If $w \neq \epsilon$, then necessarily w has the form $(\mathsf{sd},i,j)\#w''$ where $(w''\#w',\pi'') \in \pi$. By the prefix-closeness of π, we obtain π' such that $(w'',\pi') \in \pi$ and $(w',\pi'') \in \pi'$; the former implies $((\mathsf{sd},i,j)\#w'',(1,\pi')) \in \delta *$, i.e., $((w,(1,\pi')) \in \delta *$, and therefore π' is the desired item.

For the definition of **Inp**, given $n \in I\!N$, $i \in \overline{1,n}$, $f : \overline{0,n} \to \mathit{Proc}$, and $\pi \in \mathit{Proc}$, we used the singleton multi-step parameterized coalgebra $(\{*\},\delta : \{*\} \to \mathcal{P}(\mathit{Act}^* \times (\{*\} + \mathit{Proc})))$, where $\delta * = \{(\epsilon,(0,*))\} \cup \{((\mathsf{rc},i,j)\#w,(1,\pi')).\ j \in \overline{1,n} \wedge (w,\pi') \in f\,j\} \cup \{((\mathsf{rc},i,n+1)\#w,(1,\pi')).\ (w,\pi') \in \pi\}$. The proof that this is monoidal is very similar to the one for **Out** and is omitted.

For the definition of **Parc**, we used the multi-step coalgebra $(\mathit{Config} \times \mathit{Proc} \times \mathit{Proc}, \delta : \mathit{Config} \times \mathit{Proc} \times \mathit{Proc} \to \mathcal{P}(\mathit{Act}^* \times (\mathit{Config} \times \mathit{Proc} \times \mathit{Proc}))$, where $\delta(c,\pi_1,\pi_2) = \{(w,(c',\pi_1',\pi_2')).\ \exists w_1,w_2.\,(w_1,\pi_1') \in \pi_1 \wedge (w_2,\pi_2') \in \pi_2 \wedge (c',w) \in \mathit{par}(c,w_1,w_2)\}$. Fix $(c,\pi_1,\pi_2) \in \mathit{Config} \times \mathit{Proc} \times \mathit{Proc}$. We show that (c,π_1,π_2) is monoidal. As expected, the proofs of the desired properties will rely on corresponding properties of *par* – these are the subject of Lemma 11, stated afterwards.
- We check reflexivity: By the reflexivity of π_1 and π_2, we have $(\epsilon,\pi_1) \in \pi_1$ and $(\epsilon,\pi_2) \in \pi_2$. Moreover, from Lemma 11.(1), we have $(c,\epsilon) \in \mathit{par}(c,\epsilon,\epsilon)$. It follows that $(\epsilon,(c,\pi_1,\pi_2)) \in \delta(c,\pi_1,\pi_2)$, as desired.
- We check transitivity: Assume $(w,(c',\pi_1',\pi_2')) \in \delta(c,\pi_1,\pi_2)$ and $(w',(c'',\pi_1'',\pi_2'')) \in \delta(c',\pi_1',\pi_2')$. Then there exist w_1,w_2,w_1',w_2' such that the following hold:
— $(w_1,\pi_1') \in \pi_1$, $(w_2,\pi_2') \in \pi_2$, $(c',w) \in \mathit{par}(c,w_1,w_2)$;
— $(w_1',\pi_1'') \in \pi_1'$, $(w_2',\pi_2'') \in \pi_2'$, $(c'',w') \in \mathit{par}(c',w_1',w_2')$.
With the transitivity of π_1 and π_2, we have $(w_1\#w_1',\pi_1'') \in \pi_1$ and $(w_2\#w_2',\pi_2'') \in \pi_2$. Moreover, by Lemma 11.(2), we have that $(c'',w\#w') \in \mathit{par}(c,w_1\#w_1',w_2\#w_2')$. It follows that $(w\#w',(c'',\pi_1'',\pi_2'')) \in \delta(c,\pi_1,\pi_2)$, as desired.
- We check prefix-closeness: Assume $(w\#w',(c'',\pi_1'',\pi_2'')) \in \delta(c,\pi_1,\pi_2)$. Then there exist w_1'' and w_2'' such that $(w_1'',\pi_1'') \in \pi_1$, $(w_2'',\pi_2'') \in \pi_2$, and $(c'',w\#w') \in \mathit{par}(c,w_1'',w_2'')$. By Lemma 11.(3), there exist w_1,w_1',w_2,w_2' and c' such that the following hold:

— $w_1'' = w_1 \# w_1'$, $w_2'' = w_2 \# w_2'$;
— $(c', w) \in par(c, w_1, w_2)$, $(c'', w') \in par(c', w_1', w_2')$.
With the prefix-closeness of π_1'' and π_2'', we have that there exist π_1', π_2' such that the following hold:
— $(w_1, \pi_1') \in \pi_1$, $(w_1', \pi_1'') \in \pi_1'$;
— $(w_2, \pi_2') \in \pi_2$, $(w_2', \pi_2'') \in \pi_2'$.
It follows that $(c', \pi_1', \pi_2') \in par(c, \pi_1, \pi_2)$ and $(c'', \pi_1'', \pi_2'') \in par(c', \pi_1', \pi_2')$, making (c, π_1, π_2) the desired item.

□

Lemma 11 *(1)* $(c, \epsilon) \in par(c, \epsilon, \epsilon)$.
(2) $(c', w) \in par(c, w_1, w_2)$ *and* $(c'', w') \in par(c', w_1', w_2')$ *implies* $(c'', w \# w') \in par(c, w_1 \# w_1', w_2 \# w_2')$.
(3) If $(c'', w \# w') \in par(c, w_1'', w_2'')$, *then there exist* w_1, w_1', w_2, w_2' *and* c' *such that* $w_1'' = w_1 \# w_1'$, $w_2'' = w_2 \# w_2'$, $(c', w) \in par(c, w_1, w_2)$, *and* $(c'', w') \in par(c', w_1', w_2')$.

Proof. The three facts hold true regardless of who are the relations $\_ \bar{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\rightsquigarrow} \_$, used in the definition *par*. (That is, they hold for any other relations $R_1 \subseteq Config \times Act \times Act \times Config$ and $R_2 \subseteq Config \times Act \times Config \times Act$ instead of $\_ \bar{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\rightsquigarrow} \_$). The reason is that at most one action is consumed from the beginning of each of the last two arguments during recursive calls in the definition of *par*. This being said, fact (1) is immediate, and easy inductions prove the other two facts. □

The rest of this section is dedicated to the proof of full abstraction. Figure 1 gives, for future reference, the full description of the system OS (without any rule omission).

Fig. 1. The system OS

$$\frac{}{\overline{x}y.P \xrightarrow{\overline{x}y} P}(\mathsf{Out}) \qquad \frac{}{x(y).P \xrightarrow{x(y)} P}(\mathsf{Inp}) \qquad \frac{P \mid !P \xrightarrow{\gamma} P'}{!P \xrightarrow{\gamma} P'}(\mathsf{Repl})$$

$$\frac{P \xrightarrow{\gamma} P'}{\nu\, z.\, P \xrightarrow{\gamma} \nu\, z.\, P'}\ (\mathsf{Nu})\ [z \notin V(\gamma)] \qquad \frac{P \xrightarrow{\overline{x}y} P'}{\nu\, y.\, P \xrightarrow{\overline{x}(y)} P'}\ (\mathsf{Open})\ [x \neq y]$$

$$\frac{P \xrightarrow{\gamma} P'}{P \mid Q \xrightarrow{\gamma} P' \mid Q}\ (\mathsf{ParL})\ [BV(\gamma) \cap FV(Q) = \emptyset] \qquad \frac{Q \xrightarrow{\gamma} Q'}{P \mid Q \xrightarrow{\gamma} P \mid Q'}\ (\mathsf{ParR})\ [BV(\gamma) \cap FV(P) = \emptyset]$$

$$\frac{P \xrightarrow{\overline{x}z} P' \qquad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid (Q'[z/y])}(\mathsf{ComOL}) \qquad \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}z} Q'}{P \mid Q \xrightarrow{\tau} (P'[z/y]) \mid Q'}(\mathsf{ComOR})$$

$$\frac{P \xrightarrow{\overline{x}(y)} P' \qquad Q \xrightarrow{x(y)} Q'}{P \mid Q \xrightarrow{\tau} \nu\, y.\, P' \mid Q'}(\mathsf{ComNL}) \qquad \frac{P \xrightarrow{x(y)} P' \qquad Q \xrightarrow{\overline{x}(y)} Q'}{P \mid Q \xrightarrow{\tau} \nu\, y.\, P' \mid Q'}(\mathsf{ComNR})$$

We start by recalling some folklore properties of OS w.r.t. free variables and substitution (see [16], [15], [23]).

Lemma 12 *(1) If* $\vdash_{OS} P \xrightarrow{\gamma} P'$, *then* $FV(\gamma) \subseteq FV(P)$ *and* $FV(P') \subseteq BV(\gamma) \cup FV(P)$.
(2) If x' *is fresh for* P *and* $\vdash_{OS} P \xrightarrow{\overline{x}y} P'$, *then equally*
$\vdash_{OS} P[x'/y'] \xrightarrow{\overline{x[x'/y']}\, y[x'/y']} P'[x'/y']$.
(3) If x' *is fresh for* P, $y \notin \{x', y'\}$ *and* $\vdash_{OS} P \xrightarrow{\overline{x}(y)} P'$, *then equally*
$\vdash_{OS} P[x'/y'] \xrightarrow{\overline{x[x'/y']}\,(y)} P'[x'/y']$.
(4) If x' *is fresh for* P, $y \notin \{x', y'\}$ *and* $\vdash_{OS} P \xrightarrow{x(y)} P'$, *then equally*
$\vdash_{OS} P[x'/y'] \xrightarrow{x[x'/y']\,(y)} P'[x'/y']$.
(5) If y' *is fresh for* P *and* $\vdash_{OS} P \xrightarrow{\overline{x}(y)} P'$, *then equally* $\vdash_{OS} P \xrightarrow{\overline{x}(y')} P'[y'/y]$.
(6) If y' *is fresh for* P *and* $\vdash_{OS} P \xrightarrow{x(y)} P'$, *then equally* $\vdash_{OS} P \xrightarrow{x(y')} P'[y'/y]$.
(7) If x' *is fresh for* P *and* $\vdash_{OS} P \xrightarrow{\tau} P'$, *then equally* $\vdash_{OS} P[x'/y'] \xrightarrow{\tau} P'[x'/y']$.

Next, we mention a minor technical point concerning our semantic map $[[\_]] : Pterm \to \prod_{n\in\mathbb{N}} Env_n \to Proc$ defined at the end of Section 3. Since we chose to identify pterms modulo alpha-equivalence, one needs to check that the above definition is correct, in that, for the cases of input and restriction, it does not depend on the choice of the representatives. This can be readily achieved by proving recursively on k that the following facts *together* :

- $[[\_]]$ is well defined for pterms of depth $\leq k$;
- $[[\, P[y/x]\,]]\, n\, \rho = [[P]]\, n\, (\rho[y \leftarrow \rho(x)])$ for all $n \in \mathbb{N}$, $\rho \in Env_n$ and P pterm of depth $\leq k$.

Now we proceed with our planed proof development.

B.1 CS - a system with concrete actions

The set *Pterm_con*, of *concrete process terms* (*cpterms* for short), ranged over by U, V, is given by the following grammar:

$$\begin{array}{rcl} U & ::= & Zero \mid Out_n(i,j,U) \mid Inp_n(i,U_0,\ldots,U_n,V) \mid Repl_n(U) \\ & & Parc(c,U,V) \mid Lm(c,U,V) \mid Com(c,U,V) \end{array}$$

where Out_n, Inp_n, and $Repl_n$ represent distinct symbols for each n and where one has the additional constraints that $i, j \in \overline{0,n}$ in $Out_n(i,j,U)$ and $i \in \overline{0,n}$ in $Inp_n(i,U_0,\ldots,U_n,V)$. (Recall that c ranges over configurations.)

Looking at the semantic definitions of restriction and parallel composition in terms of parallel composition in configurations, we define, for each $n \in \mathbb{N}$:

- $rcfg(n)$, the n-restriction configuration, to be $(n, id_n, \mathbf{I}, \{(n+1, 0)\})$.
- $pcfg(n)$, the n-parallel-composition configuration, to be $(n, id_n, id_n, \emptyset)$,

Also, for each $c = (n, f_1, f_2, R)$, we define:

- *Con*(c), the set of *connections of* c, to be $R \cup \{(i_1, i_2) : f_1 i_1 = f_2 i_2 \neq \bot\}$;
- *sym*(c), the configuration symmetric to c, to be $(n, f_2, f_1, R^\smile)$.

Now we introduce the system CS (where "C" stands for "concrete", as it deals with concrete, rather than generic actions), for inferring triples $U \xrightarrow{\alpha} U'$, with $U, U' \in \mathit{Pterm\_con}$ and $\alpha \in \mathit{Act} \cup \{\tau\}$ – listed in Figure 2. From now on, α will range over $\mathit{Act} \cup \{\tau\}$. (Recall that a ranges over *Act*.)

The strong and weak bisimulation-related notions that we have in mind for CS are completely standard. However, we present them here to avoid any confusion. (Below, for a set L of words over $\mathit{Act} \cup \{\tau\}$ (i.e., a language over the alphabet $\mathit{Act} \cup \{\tau\}$), we write $\vdash_{CS} U \xrightarrow{L} U'$ to indicate that there exist $n \in I\!N$, $V_0, \ldots, V_n$ and a word $\alpha_0 \ldots \alpha_n \in L$ such that $V_0 = U$, $V_n = U'$, and $\vdash_{CS} V_i \xrightarrow{\alpha_i} V_{i+1}$ for all $i \in \overline{0, n-1}$. Given $a \in \mathit{Act}$, the expressions $\tau^* a \, \tau^*$, $\tau^* a$, $a \, \tau^*$ and τ^* denote the obvious regular languages.)

- A relation θ on *Pterm_con* is called a *strong CS-simulation* if the following holds for all $(U_0, U_1) \in \theta$: whenever $\vdash_{CS} U_0 \xrightarrow{\alpha} U_0'$ for some α and U_0', there exists U_1' such that $\vdash_{CS} U_1 \xrightarrow{\alpha} U_1'$ and $(U_0', U_1') \in \theta$. θ is called a *strong CS-bisimulation* if θ and $\theta^\smile$ are both strong CS-simulations. The *strong CS-bisimilarity* relation, denoted by $\sim_{s,CS}$, is the union of all strong CS-bisimulations, and since $\sim_{s,CS}$ is a strong CS-bisimulation itself, it is the largest one.
- A relation θ is called a *CS-simulation* if the following hold for all $(U_0, U_1) \in \theta$:
  - whenever $\vdash_{CS} U_0 \xrightarrow{\tau^*} U_0'$ for some U_0', there exists U_1' such that $\vdash_{CS} U_1 \xrightarrow{\tau^*} U_1'$ and $(U_0', U_1') \in \theta$.
  - whenever $\vdash_{CS} U_0 \xrightarrow{\tau^* a \tau^*} U_0'$ for some a and U_0', there exists U_1' such that $\vdash_{CS} U_1 \xrightarrow{\tau^* a \tau^*} U_1'$ and $(U_0', U_1') \in \theta$.

  θ is called a *CS-bisimulation* if θ and $\theta^\smile$ are both CS-simulations. The *CS-bisimilarity* relation, denoted by $\sim_{CS}$, is the union of all CS-bisimulations, and since $\sim_{CS}$ is a CS-bisimulation itself, it is the largest one.

What is to be noted about the system CS is that all the syntactic constructs have the one-step behavior of their semantic counterparts in *Proc*. In particular, *Parc* behaves essentially as prescribed by the relations $\_ \tilde{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\rightsquigarrow} \_$ from Section 3 (we are also using the auxiliary operators *Lm* and *Com*, for which we did not define semantic counterparts on *Proc* – but these are just cosmetics, solely meant to split in two the behavior of *Parc*, having a similar role with *com* and *lmerge* for *par* on traces).

We shall show that CS, on the one hand, simulates OS faithfully via environments, and, on the other hand, its weak bisimilarity is captured faithfully by the semantic domain *Proc*. The rest of this subsection is dedicated to the study of CS in preparation for these results. For a quicker glimpse into the overall proof, the reader may skip now directly to the next subsection, B.2.

Fig. 2. The system CS

In the rules below, we assume $c = (n, f_1, f_2, R)$ and $c' = (n', f_1', f_2', R')$.

$$\frac{\cdot}{Out_n(i,j,U) \xrightarrow{(\mathsf{sd},i,j)} U}\ (\mathsf{OutC})\ [i,j \in \overline{0,n}]$$

$$\frac{\cdot}{Inp_n(i,U_0,\ldots,U_n,V) \xrightarrow{(\mathsf{rc},i,j)} U_j}\ (\mathsf{InpOC})\ [i,j \in \overline{0,n}]$$

$$\frac{\cdot}{Inp_n(i,U_0,\ldots,U_n,V) \xrightarrow{(\mathsf{rc},i,n+1)} V}\ (\mathsf{InpNC})\ [i \in \overline{0,n}]$$

$$\frac{Parc(pcfg(n),U,Repl_n(U)) \xrightarrow{\alpha} U'}{Repl_n(U) \xrightarrow{\alpha} U'}(\mathsf{ReplC})$$

$$\frac{Lm(c,U,V) \xrightarrow{\alpha} U'}{Parc(c,U,V) \xrightarrow{\alpha} U'}(\mathsf{ParcToLm}) \qquad \frac{Lm(sym(c),V,U) \xrightarrow{\alpha} U'}{Parc(c,U,V) \xrightarrow{\alpha} U'}(\mathsf{ParcToLmSym})$$

$$\frac{Com(c,U,V) \xrightarrow{\alpha} U'}{Parc(c,U,V) \xrightarrow{\alpha} U'}(\mathsf{ParcToCom}) \qquad \frac{Com(sym(c),V,U) \xrightarrow{\alpha} U'}{Parc(c,U,V) \xrightarrow{\alpha} U'}(\mathsf{ParcToComSym})$$

$$\frac{U \xrightarrow{\tau} U'}{Lm(c,U,V) \xrightarrow{\tau} Parc(c,U',V)}(\mathsf{LmTau})$$

$$\frac{U \xrightarrow{(vb,i_1,j_1)} U'}{Lm(c,U,V) \xrightarrow{(vb,f_1i_1,f_1j_1)} Parc(c,U',V)}\ (\mathsf{LmVbO})\ [\bot \notin \{f_1i_1, f_1j_1\}]$$

$$\frac{U \xrightarrow{(\mathsf{sd},i_1,j_1)} U'}{Lm(c,U,V) \xrightarrow{(\mathsf{sd},f_1i_1,n+1)} Parc(c',U',V)}\ (\mathsf{LmSdPr})\ [f_1i_1 \neq \bot,\ (j_1,j_2) \in R]$$
(where $n' = n+1$, $f_1' = f_1[j_1 \leftarrow n+1]$, $f_2' = f_2[j_2 \leftarrow n+1]$, $R' = R \setminus \{(j_1,j_2)\}$)

$$\frac{U \xrightarrow{(vb,i_1,j_1)} U'}{Lm(c,U,V) \xrightarrow{(vb,f_1i_1,n+1)} Parc(c',U',V)}\ (\mathsf{LmVbN})\ [f_1i_1 \neq \bot = f_1j_1,\ j_1 \notin Prj_1(R)]$$
(where $n' = n+1$, $f_1' = f_1[j_1 \leftarrow n+1]$, $f_2' = f_2$, $R' = R$)

$$\frac{U \xrightarrow{(\mathsf{rc},i_1,j_1)} U'}{Lm(c,U,V) \xrightarrow{(\mathsf{rc},f_1i_1,j)} Parc(c',U',V)}\ (\mathsf{LmRcOasN})\ [f_1i_1 \neq \bot = f_1j_1,\ j_1 \notin Prj_1(R),\ j \in \overline{0,n} \setminus Im(f_1)]$$
(where $n' = n$, $f_1' = f_1[j_1 \leftarrow j]$, $f_2' = f_2$, $R' = R$)

$$\frac{U \xrightarrow{(\mathsf{sd},i_1,j_1)} U' \qquad V \xrightarrow{(\mathsf{rc},i_2,j_2)} V'}{Com(c,U,V) \xrightarrow{\tau} Parc(c,U',V')}\ (\mathsf{ComCon})\ [(i_1,i_2),(j_1,j_2) \in Con(c)]$$

$$\frac{U \xrightarrow{(\mathsf{sd},i_1,j_1)} U' \qquad V \xrightarrow{(\mathsf{rc},i_2,j_2)} V'}{Com(c,U,V) \xrightarrow{\tau} Parc(c',U',V')}\ (\mathsf{ComON})\ [(i_1,i_2) \in Con(c),\ \bot \neq f_1j_1 \notin Im(f_2),\ f_2j_2 = \bot]$$
(where $n' = n$, $f_1' = f_1$, $f_2' = f_2[j_2 \leftarrow f_1\, j_1]$, $R' = R$)

$$\frac{U \xrightarrow{(\mathsf{sd},i_1,j_1)} U' \qquad V \xrightarrow{(\mathsf{rc},i_2,j_2)} V'}{Com(c,U,V) \xrightarrow{\tau} Parc(c',U',V')}\ (\mathsf{ComNN})\ [(i_1,i_2) \in Con(c),\ f_1j_1 = f_2j_2 = \bot]$$
(where $n' = n$, $f_1' = f_1$, $f_2' = f_2$, $R' = R \cup \{(j_1,j_2)\}$)

Lemma 13 *$\sim_{s,CS}$ and $\sim_{CS}$ are congruences on Pterm_con.*

Proof. One can regard the system CS as given by a specification in *non-circular tyft format* [11] (one does not look at configurations as arguments of the term constructs, but rather considers, for instance, an operation symbol Par_c for each configuration c). Then, according to [11], $\sim_{s,CS}$ is a congruence. Moreover, adding, in a standard way, rules for τ-reflexivity and τ-transitivity to our system, namely adding the rules:

$$\frac{\cdot}{U \stackrel{\tau}{\longrightarrow} U}(\mathsf{TauRf}) \quad \frac{U \stackrel{\tau}{\longrightarrow} U' \quad U' \stackrel{\alpha}{\longrightarrow} U''}{U \stackrel{\alpha}{\longrightarrow} U''}(\mathsf{TauTrL}) \quad \frac{U \stackrel{\alpha}{\longrightarrow} U' \quad U' \stackrel{\tau}{\longrightarrow} U''}{U \stackrel{\alpha}{\longrightarrow} U''}(\mathsf{TauTrR})$$

one obtains that $\sim_{CS}$ is the *strong* bisimilarity of the extended system, and since adding these new rules keeps the *non-circular tyft/tyxt-format* [11], we obtain that $\sim_{CS}$ is a congruence too. (Of course, one needs some verifications to make sure that indeed strong bisimilarity in the extended system yields bisimilarity in CS, but these verifications are completely standard and do not bring any difficulty additional to, say, the case of CCS without sum. Had we considered a guarded sum operator into the original calculus OS, it would have corresponded to a guarded sum operator in CS, and the facts stated in this lemma would have still been true.) □

Next we introduce some further notations and conventions, useful for our subsequent proofs. Let $n \in I\!N$. Throughout this paragraph, by "item" we mean an element of $I\!N \to \overline{0,n}_\perp$. Recall our convention to associate to its item a partial function from $I\!N$ to $\overline{0,n}$. If $\sigma : \overline{0,n} \to \overline{0,n}$ is a permutation, i.e., a bijective function, then σ_n denotes the item defined by $\sigma_n\, k = \sigma\, k$ if $k \leq n$ and $= \perp$ otherwise. If $i, j \in \overline{0,n}$, then $sw_n(i,j)$ is the item defined by: $sw_n(i,j)i = j$, $sw_n(i,j)j = i$, $sw_n(i,j)k = k$ if $k \leq n$ and $i \neq k \neq j$, and $sw_n(i,j)k = \perp$ if $k > n$. Recall that id_n is σ_n, where σ is the identity of $\overline{0,n}$ – we extend this notation, writing , for $m, m' \leq n$, $id_{m,m'}$ for the item in $I\!N \to \overline{0,n}_\perp$ defined as $id_{m,m'}(k) = k$ if $k \in \overline{m,m'}$ and $= \perp$ otherwise. Also, we sometimes write (i,j) for the item that maps i to j and anything else to $\perp$. If $k \leq i \leq j \leq n$, $dn_{k,i,j}$ denotes the item whose associated partial function is $\{(u, u-k) : u \in \overline{i,j}\}$. $up_{k,i,j}$ denotes the item whose associated partial function is $\{(u, u+k) : u \in \overline{i,j}\}$. $dn_{i,j}$ denotes $dn_{1,i,j}$ and $up_{i,j}$ denotes $up_{1,i,j}$. Given two items f_1 and f_2, we sometimes write $f_1 \circ f_2$ for the item obtained from composing f_1 and f_2 as partial functions; moreover, if f_1 and f_2 have disjoint domains when regarded as partial functions, we write $f_1 + f_2$ for their union/supremum as partial functions. Given a relation $R \subseteq I\!N \times I\!N$ and an item f, we sometimes write $R \circ f$ and $f \circ R$ for the relation in $I\!N \times I\!N$ obtained by (pre or post) composing, as relations, R with the partial function given by f. If an item f is injective when regarded as a partial function, $inv(f)$ is the item corresponding to the inverse of this partial function.

For a configuration $c = (n, f_1, f_2, R)$, we shall need to use the following numbers: $n_1 = \max(Prj_1(R) \cup \{i_1 : f_1\, i_1 \neq \perp\})$ and $n_2 = \max(Prj_2(R) \cup \{i_2 : f_2\, i_2 \neq \perp\})$. These numbers, representing the greatest channels which the presumptive processes composed in this configuration may regard as known, exist

provided the involved sets are non-empty because of the finiteness of R and the injectivity, hence domain-finiteness of f_1 and f_2 regarded as partial functions. If the involved set is empty, we agree to let n_1 or n_2 be 0. For the rest of this section we regard n_1 and n_2 as part of the structure of the configurations c, writing c as $(n, n_1, n_2, f_1, f_2, R)$.

We define, for all $n \in I\!N$, the sets $Pterm\_con_n$, of *cpterms at level n* (*n-cpterms* for short), mutually recursively by the following grammar, where U^m and V^m (with possible additional subscripts) are temporarily set to range over (i.e., be nonterminals in the grammar for) $Pterm\_con_m$:

$$\begin{array}{rcl} U^n & ::= & Zero \mid Out_n(i,j,U^n) \mid Inp_n(i,U_0^n,\ldots,U_n^n,V^{n+1}) \mid Repl_n(U^n) \\ & & Parc((n,n_1,n_2,f_1,f_2,R),U^{n_1},V^{n_2}) \\ & & Lm((n,n_1,n_2,f_1,f_2,R),U^{n_1},V^{n_2}) \\ & & Com((n,n_1,n_2,f_1,f_2,R),U^{n_1},V^{n_2}) \end{array}$$

with the usual constraints that $i,j \in \overline{0,n}$ in $Out_n(i,j,U^n)$ and $i \in \overline{0,n}$ in $Inp_n(i,U_0^n,\ldots,U_n^n,V^{n+1})$.

Intuitively, a cpterm at level n is one that may only send or receive on the ports in $\overline{0,n}$ and only data in $\overline{0,n}$ as old, known data, or $n+1$ as a new, unknown datum. (Both the definition of n-cpterms and Lemma 14 below back up this intuition.) Notice that, for all $m \leq n$, $Pterm\_con_m \subseteq Pterm\_con_n \subseteq Pterm\_con$, although it is not the case that $Pterm\_con = \bigcup_{n \in I\!N} Pterm\_con_n$.

We define, for each n, the set of *actions at level n*, Act_n, to be $\{\tau\} \cup \{(vb,i,j) \in Act : i,j \in \overline{0,n}\}$, and the set of *actions between levels n and n + 1*, $Act_{n,n+1}$, to be $\{(vb,i,n+1) \in Act : i \in \overline{0,n}\}$.

Lemma 14 *If $n \in I\!N$, $\vdash_{CS} U \xrightarrow{\alpha} U'$ and $U \in Pterm\_con_n$, then:*
— either $U' \in Pterm\_con_n$ and $\alpha \in Act_n$;
— or $U' \in Pterm\_con_n$ and $\alpha \in Act_{n,n+1}$.

Proof. Immediate induction on the structure of CS-derivation trees. □

For all n, we define the *n-to-(n+1) shift function*, $\uparrow_n : Pterm\_con_n \to Pterm\_con_{n+1}$, as follows:

- $\uparrow_n(Zero) = Zero$;
- $\uparrow_n(Out_n(i,j,U)) = Out_n(i,j,\uparrow_n(U))$;
- $\uparrow_n(Inp_n(i,U_0,\ldots,U_n,V)) = Inp_n(i,\uparrow_n(U_0),\ldots,\uparrow_n(U_n),V,Parc((n+2,n+2,0,sw_{n+2,n+1,n+2},\mathbf{1},\emptyset),\uparrow_{n+1}(V),Zero))$.
- $\uparrow_n(Repl_n(U)) = Repl_{n+1}(\uparrow_n(U))$
- $\uparrow_n(Parc((n,n_1,n_2,f_1,f_2,R),U,V)) = Parc((n+1,n_1+1,n_2+1,f_1[n_1+1 \leftarrow n+1],f_2[n_2+1 \leftarrow n+1],R),\uparrow_{n_1}(U),\uparrow_{n_2}(V))$

The function $\uparrow_n$, applied to a cpterm U at level n, makes room for an extra channel $n+1$, that will be treated as known. $n+1$ cannot be initially used as a port or sent as datum, since it is not "really known". However, $\uparrow_n(U)$ can receive $n+1$ as a known channel, and then behave just like U would behave

upon receiving a new channel. Moreover, $\uparrow_n(U)$ can also receive a new channel, identified as $n+2$, becoming whatever continuation U had upon receiving a new channel, except that this continuation still needs to make room for channel $n+1$ – and the way to make room, while at level $n+1$, for $n+1$, is to make room for a new channel $n+2$, and then swap $n+1$ with $n+2$ in all the subsequent actions.

Lemma 15
(1) If $U \in Pterm\_con_n$, *then* $U \sim_{s,CS} Parc((n,n,0,id_n,\mathbb{1},\emptyset),U,Zero)$.
(2) If $U \in Pterm\_con_n$ *and* $V \in Pterm\_con_{n_2}$, *then*
$Parc((n,n_1,n_2,f_1,f_2,R),U,Parc((n_2,n_2,0,id_{n_2},\mathbb{1},\emptyset),V,Zero) \sim_{s,CS}$
$Parc((n,n_1,n_2,f_1,f_2,R),U,V)$.
(3) If $U \in Pterm_{n+1}$, *then* $Parc((n,n,0,\sigma_n,\mathbb{1},\emptyset),Parc(rcfg(n),U,Zero),Zero) \sim_{s,CS}$
$Parc(rcfg(n),Parc((n+1,n+1,0,\sigma_n[n+1 \leftarrow n+1],\mathbb{1},\emptyset),U,Zero),Zero)$.
(4) If $U,V \in Pterm_{n+1}$, then $Parc((n,n+1,n+1,id_n,id_n,\{(n+1,n+1)\}),U,V)) \sim_{s,CS}$
$Parc((n,n,0,id_n,\mathbb{1},\{(n+1,0)\}),Parc((n+1,n+1,n+1,id_{n+1},id_{n+1},\emptyset),U,V),Zero)$.
(5) If $U \in Pterm\_con_{n+1}$ and $V \in Pterm\_con_n$, then
$Parc((n+1,n+1,n,id_{n+1},id_n,\emptyset),U,V) \sim_{s,CS}$
$Parc((n+1,n+1,n+1,id_{n+1},id_{n+1},\emptyset),U,\uparrow_n(V))$.
(6) If $U \in Pterm\_con_n$ and $V \in Pterm\_con_{n+1}$, then
$Parc((n+1,n,n+1,id_n,id_{n+1},\emptyset),U,V) \sim_{s,CS}$
$Parc((n+1,n+1,n+1,id_{n+1},id_{n+1},\emptyset),\uparrow_n(U),V)$.

Proof. (1): It would be enough to show that the set $\{(U, Parc((n,n,0,id_n,\mathbb{1},\emptyset),U,Zero)) : n \in I\!N,\ U \in Pterm\_con\}$ is a strong CS-bisimulation. And the latter is true since any derivation tree for a transition of the form $Parc((n,n,0,id_n,\mathbb{1},\emptyset),U,Zero)) \xrightarrow{\alpha} V'$ has as last applied rules (ParcToLm), and then one of (LmTau), (LmVbO), case in which $V' = Parc((n,n,0,id_n,\mathbb{1},\emptyset),U',Zero)$ and $\vdash_{CS} U \xrightarrow{\alpha} U'$, or (LmVbNew) with $j_1 = n+1$ (thanks to Lemma 14), case in which $V' = Parc((n+1,n+1,0,id_{n+1},\mathbb{1},\emptyset),U',Zero)$ and $\vdash_{CS} U \xrightarrow{\alpha} U'$.
(2): Again, one can check that the set of all pairs satisfying the indicated pattern is a strong CS-bisimulation.
(3): The desired strong CS-bisimulation consists of the following pairs, obtained by tracing the effect that CS transitions have on pairs with the initial pattern:
– pairs given by series of actions that do not contain the action of sending out the only private link, $(n+1,0)$, existing in each of the two systems:
$(Parc((n+k,n+k,0,\sigma_n+id_{n+1,n+k},\mathbb{1},\emptyset),Parc((n+k,n+k+1,0,id_n+dn_{n+2,n+k+1},\mathbb{1},\{(n+1,0)\}),U,Zero),Zero)$,
$Parc((n+k,n+k+1,0,id_n+dn_{n+2,n+k+1},\mathbb{1},\{(n+1,0)\}),Parc((n+k+1,n+k+1,0,\sigma_n+id_{n+1,n+k+1},\mathbb{1},\emptyset),U,Zero),Zero))$,
with arbitrary k and $U \in Pterm_{n+k+1}$;
- pairs after the aforementioned private link has been sent out:
$(Parc((n+k+1+l,n+k+1+l,0,\sigma_n+id_{n+1,n+k+1+l},\mathbb{1},\emptyset),Parc((n+k+1+l,n+k+1+l,0,id_n+dn_{n+2,n+k+1}+(n+1,n+k+1)+id_{n+k+2,n+k+1+l},\mathbb{1},\emptyset),U,Zero),Zero)$,
$Parc((n+k+1+l,n+k+1+l,0,id_n+(n+1,n+k+1))+dn_{n+2,n+k+1}+$

$id_{n+k+2,n+k+1+l}, \mathbf{1}, \emptyset), Parc((n+k+1, n+k+1, 0, \sigma_n + id_{n+1,n+k+1+l}, \mathbf{1}, \emptyset), U, Zero), Zero))$, with arbitrary k, l and $U \in Pterm_{n+k+1+l}$.
(4): Similar to (3).
(5): Let θ be the relation consisting of the following pairs:
-(a) $(Parc((n+1+k_1+k_2-p, n+1+k_1, n+k_2, id_{n+1}+f_1, id_n+f_2, R), U, V)$, $Parc((n+1+k_1+k_2-p, n+1+k_1, n+1+k_2, id_{n+1}+f_1, id_n+f_2+(n+1+k_2, n+1), R), U, \uparrow_{n+k_2}(V)))$,
where k_1, k_2, p are such that $p \leq k_1, k_2$, f_1 and f_2, have their graphs, when regarded as partial functions, included in $\overline{n+2, n+1+k_1} \times \overline{n+2, n+1+k_1+k_2-p}$ and $\overline{n+1, n+k_2} \times \overline{n+2, n+1+k_1+k_2-p}$, respectively, $card(R) = p$, $U \in Pterm\_con_{n+1+k_1}$, $V \in Pterm\_con_{n+k_2}$ and the involved items do constitute valid configurations.
-(b) (U, U), with $U \in Pterm\_con_m$ for $m \geq n$.

Intuitively, the (synchronized) evolution of the systems $W_1 = Parc((n+1, n+1, n, id_{n+1}, id_n, \emptyset), U, V)$ and $W_2 = Parc((n+1, n+1, n+1, id_{n+1}, id_{n+1}, \emptyset), U, \uparrow_n (V))$ (which need to be shown bisimilar) goes through systems (strongly bisimilar to ones) satisfying pattern (a) (where R is the set of private links created in the meantime inside these systems and f_1, f_2 give connect their components to public channels added added in the meantime) as long as channel $n+1$ is not received by the second component of these systems; after this event happens, the two systems become identical. One can prove, by induction on k, the following statement: if $(U, V) \in \theta$, then:
- if, for some α, U', $U \xrightarrow{\alpha} U'$ has a derivation tree of depth k, then there exists V' such that $(U', V') \in \sim_{s,CS} \circ \theta \circ \sim_{s,CS}$ and $\vdash_{CS} V \xrightarrow{\alpha} V'$;
- if, for some α, V', $V \xrightarrow{\alpha} V'$ has a derivation tree of depth k, then there exists U' such that $(U', V') \in \sim_{s,CS} \circ \theta \circ \sim_{s,CS}$ and $\vdash_{CS} U \xrightarrow{\alpha} U'$.
(For the case of input, one needs to employ point (3) of the lemma.)
(6): The fact is symmetric to (5) and has a symmetric justification. □

B.2 CS versus OS

Next we define a translation from OS to CS, via environments. In the clauses of this translation the reader will of course recognize the clauses of the definition for the denotation map in Section 3. Recall from Section 3 that, for each n, Env_n is the set of n-environments and is ranged over by ρ, and that we agree that $Env_m \subseteq Env_n$ if $m \leq n$. When $\rho \in Env_n$, we refer to ρ when regarded as an element of Env_{n+1} as the *shifted* version of ρ. For each $n \in I\!N$, $\_[\_]_n : Pterm \times Env_n \to Pterm\_con$ is defined recursively as follows:

- $0[\rho]_n = Zero$.
- $(\overline{x}y.\, P)[\rho]_n = Out_n(\rho(x), \rho(y), P[\rho]_n)$.
- $(x(y).\, P)[\rho]_n = Inp_n(\rho(x), P[\rho[y \leftarrow 0]]_n, \ldots, P[\rho[y \leftarrow n]]_n, P[\rho[y \leftarrow n+1]]_{n+1})$.
- $(\nu\, y.\, P)[\rho]_n = Parc(rcfg(n), P[\rho[y \leftarrow n+1]]_{n+1}, Zero)$.
- $(P|Q)[\rho]_n = Parc(pcfg(n), P[\rho]_n, Q[\rho]_n)$.
- $(!P)[\rho]_n = Repl_n(P[\rho]_n)$.

Thus, given interpretations of the names/variables as "concrete" channels (represented as numbers) $\leq n$, one can regard, via $\_[\_]_n$, a pterm as a cpterm. $\_[\_]_n$ commutes with all syntactic constructs except for input and restriction. Moreover, the $n+1$ possible continuations of an Inp_n-cpterm in CS hardwire all the relevant inputs for an input-prefixed pterm in OS with formal input parameter y, namely:

- any input of an old channel i, i.e., one in $\overline{0,n}$, yielding a biding of y to i in the environment;
- an input of a new channel, viewed as $n+1$, yielding a binding of y to $n+1$ in the shifted environment.

Similarly to the input of a bound name, a ν-pterm is mapped to a Res_n-cpterm where the bound variable is bound, in the shifted environment, to $n+1$.

Lemma 16 *For all $n \in \mathbb{N}$, the function $[\_]_n$ is well-defined, and, moreover, for all $P \in Pterm_n$ and $\rho \in Env_n$, the following hold:*
(1) $P[x/y][\rho]_n = P[\rho[y \leftarrow \rho(x)]]_n$.
(2) $\rho|_{FV(P)} = \rho'|_{FV(P)}$ *implies* $P[\rho]_n = P[\rho']_n$.
(3) $P[\rho]_n \in Pterm_n$.
(4) If σ is a permutation on $\overline{0,n}$, then
$P[\sigma \circ \rho]_n \sim_{s,CS} Parc((n,n,0,\sigma_n,\mathbf{1},\emptyset), P[\rho], Zero)$.
(5) $P[\rho]_{n+1} \sim_{s,CS} \uparrow_n(P[\rho]_n)$.

Proof. (1)-(4): Easy induction on the structure of pterms.
(5): By induction on the structure of the pterm P. The only case that does not follow at once from the fact that $\sim_{s,CS}$ is a congruence together with IH is that of input. Here, the goal is $(x(y).P)[\rho]_{n+1} \sim_{s,CS} \uparrow_n((x(y).P)[\rho]_n)$. Rewriting both terms according to the definitions of the involved functions, we obtain two Inp_{n+1}-cpterms, whose first n (maximal, proper) subterms are bisimilar by IH, whose $(n+1)$'st subterms are identical, and whose $(n+2)$'nd terms are $P[\rho[y \leftarrow n+2]]_{n+2}$ and $Parc((n+2,n+2,0,sw_{n+2,n+1,n+2},\mathbf{1},\emptyset), \uparrow_{n+1}(P[\rho[y \leftarrow n+1]]_{n+1}), Zero)$. Let σ be the transposition $(n+1,n+2)$ on $\overline{0,n+2}$. Then, since $\sigma \circ (\rho[y \leftarrow n+1]) = \rho[y \leftarrow n+2]$, we obtain, by IH and point (4), that $Parc((n+2,n+2,0,sw_{n+2,n+1,n+2},\mathbf{1},\emptyset), \uparrow_{n+1}(P[\rho[y \leftarrow n+1]]_{n+1}), Zero) \sim_{s,CS} Parc((n+2,n+2,0,sw_{n+2,n+1,n+2},\mathbf{1},\emptyset), P[\rho[y \leftarrow n+1]]_{n+2}, Zero) \sim_{s,CS} P[\sigma \circ (\rho[y \leftarrow n+1])]_{n+2} = P[\rho[y \leftarrow n+2]]_{n+2}$, as desired. □

Proposition 17 *(from OS to CS) Let $n \in \mathbb{N}$, $\rho \in Env_n$, $P, P' \in Pterm$ such that $\rho|_{FV(P)}$ is injective. Then:*
(1) $\vdash_{OS} P \xrightarrow{\overline{x}y} P'$ *implies* $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{sd},\rho(x),\rho(y))} \sim_{s,CS} P'[\rho]_n$.
(2) $\vdash_{OS} P \xrightarrow{\overline{x}(y)} P'$ *implies* $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{sd},\rho(x),n+1)} \sim_{s,CS} P'[\rho[y \leftarrow n+1]]_{n+1}$.
(3) $\vdash_{OS} P \xrightarrow{x(y)} P'$ *and* $i \in \overline{0,n}$ *implies* $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{rc},\rho(x),i)} \sim_{s,CS} P'[\rho[y \leftarrow i]]_n$.
(4) $\vdash_{OS} P \xrightarrow{x(y)} P'$ *implies* $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{rc},\rho(x),n+1)} \sim_{s,CS} P'[\rho[y \leftarrow n+1]]_{n+1}$.
(5) $\vdash_{OS} P \xrightarrow{\tau} P'$ *implies* $\vdash_{CS} P[\rho]_n \xrightarrow{\tau} \sim_{s,CS} P'[\rho]_n$.

Proof. Points (1)-(5) can be proved together by induction on the structure of OS-derivation trees. Below we only treat the cases that cannot be solved by just an application of IH together with the fact that $\sim_{s,CS}$ is a congruence. For convenience, we shall use the same notations as for the listing of the OS rules in Figure 1, and thus, to avoid overlapping of metavariables, we rename P and P' in the statements to be proved to P_0 and P_0'. In the arguments below, taking advantage of the fact that $\sim_{s,CS}$ is a congruence, we shall directly apply the rules in CS up to $\sim_{s,CS}$, i.e., we apply the rules in CS as if they were stated in terms of $\ldots \stackrel{\cdots}{\longrightarrow} \sim_{s,CS} \ldots$ rather than $\ldots \stackrel{\cdots}{\longrightarrow} \ldots$. Cases, according to the last applied rule:

- (Open). Then $P_0 = \nu\, y.P$ and $P_0' = P'$. Let $\rho' = \rho[y \leftarrow n+1]$. Since ρ is injective on $FV(\nu\, y.P)$ and $\rho \in Env_n$, ρ' is injective on $FV(P)$. By IH,

$\vdash_{CS} P[\rho']_{n+1} \stackrel{(\mathsf{sd},\rho'(x),\rho'(y))}{\longrightarrow} \sim_{s,CS} P'[\rho']_{n+1}$,

that is,

$\vdash_{CS} P[\rho']_{n+1} \stackrel{(\mathsf{sd},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} P'[\rho']_{n+1}$

Applying rules (LmSdPr) and (ParcToLm),

$\vdash_{CS} Parc(rcfg(n), P[\rho'], Zero) \stackrel{(\mathsf{sd},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} Parc((n+1, id_{n+1}, \mathbb{1}, \emptyset), P'[\rho'], Zero)$,

that is,

$\vdash_{CS} (\nu\, y.P)[\rho] \stackrel{(\mathsf{sd},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} Parc((n+1, id_{n+1}, \mathbb{1}, \emptyset), P'[\rho']_{n+1}, Zero)$.

Moreover,

$Parc((n+1, id_{n+1}, \mathbb{1}, \emptyset), P'[\rho']_{n+1}, Zero) \sim_{s,CS} P'[\rho']_{n+1}$,

and therefore

$\vdash_{CS} (\nu\, y.P)[\rho]_n \stackrel{(\mathsf{sd},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} P[\rho']_{n+1}$, as desired.

- (ParL), with γ being $x(y)$ (the case of output is similar). Then $P_0 = P|Q$, $P_0' = P'|Q$ and $y \notin FV(Q)$. Let $\rho' = \rho[y \leftarrow n+1]$. Then $\uparrow_n (Q[\rho]_n) \sim_{s,CS} Q[\rho]_{n+1} = Q[\rho']_{n+1}$. By IH,

$\vdash_{CS} P[\rho]_n \stackrel{(\mathsf{rc},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} P'[\rho']_{n+1}$.

Applying rules (LmVbN) and (ParcToLm),

$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \stackrel{(\mathsf{rc},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} Parc((n+1, id_{n+1}, id_n, \emptyset), P'[\rho']_{n+1}, Q[\rho]_n)$.

Moreover,

$Parc((n+1, id_{n+1}, id_n, \emptyset), P'[\rho']_{n+1}, Q[\rho]_n) \sim_{s,CS} Parc(pcfg(n+1), P'[\rho']_{n+1}, \uparrow_n(Q[\rho]_n))$,

and therefore

$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \stackrel{(\mathsf{rc},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} Parc(pcfg(n+1), P'[\rho']_{n+1}, \uparrow_n(Q[\rho]_n))$,

that is,

$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \stackrel{(\mathsf{rc},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} Parc(pcfg(n+1), P'[\rho']_{n+1}, Q[\rho']_{n+1})$,

that is,

$\vdash_{CS} (P|Q)[\rho]_n \stackrel{(\mathsf{rc},\rho(x),n+1)}{\longrightarrow} \sim_{s,CS} (P'|Q)[\rho[y \leftarrow n+1]]_{n+1}$,

as desired.

- (ComOL). Then $P_0 = P|Q$ and $P_0' = P'|(Q'[z/y])$. We have $Q'[\rho[y \leftarrow \rho(z)]]_n = Q'[z/y][\rho]_n$. By IH,

$\vdash_{CS} P[\rho]_n \stackrel{(\mathsf{sd},\rho(x),\rho(z))}{\longrightarrow} \sim_{s,CS} P'[\rho]_n$ and $\vdash_{CS} P[\rho]_n \stackrel{(\mathsf{rc},\rho(x),\rho(z))}{\longrightarrow} \sim_{s,CS} Q'[\rho[y \leftarrow \rho(z)]]_n$.

Applying rules (LmVbO) and (ParcToCom),

$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \xrightarrow{\tau} \sim_{s,CS} Parc(pcfg(n), P'[\rho]_n, Q'[\rho[y \leftarrow \rho(z)]]_n)$,
that is,
$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \xrightarrow{\tau} \sim_{s,CS} Parc(pcfg(n), P'[\rho]_n, Q'[z/y][\rho]_n)$,
that is,
$\vdash_{CS} (P|Q)[\rho]_n \xrightarrow{\tau} \sim_{s,CS} (P'|(Q'[z/y]))[\rho]_n$,
as desired.
- (ComNL). Then $P_0 = P|Q$ and $P'_0 = \nu\, y.\, P'|Q'$. Let $\rho' = \rho[y \leftarrow n+1]$. By IH,
$\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{sd},\rho(x),n+1)} \sim_{s,CS} P'[\rho']_{n+1}$ and $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{rc},\rho(x),n+1)} \sim_{s,CS} Q'[\rho']_{n+1}$.
Applying rules (ComNN) and (ParcToCom),
$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \xrightarrow{\tau} \sim_{s,CS} Parc((n, id_n, id_n, \{(n+1, n+1)\}), P'[\rho']_{n+1}, Q'[\rho']_{n+1})$.
Moreover,
$Parc((n, id_n, id_n, \{(n+1, n+1)\}), P'[\rho']_{n+1}, Q'[\rho']_{n+1}) \sim_{s,CS} Parc(rcfg(n), Parc(pcfg(n+1), P'[\rho']_{n+1}, Q'[\rho']_{n+1}))$,
and therefore
$\vdash_{CS} Parc(pcfg(n), P[\rho]_n, Q[\rho]_n) \xrightarrow{\tau} \sim_{s,CS} Parc(rcfg(n), Parc(pcfg(n+1), P'[\rho']_{n+1}, Q'[\rho']_{n+1}))$,
that is,
$\vdash_{CS} (P|Q)[\rho]_n \xrightarrow{\tau} \sim_{s,CS} (\nu\, y.\, P'|Q')[\rho]_n$, as desired. □

Proposition 18 *(from CS to OS) Let $n \in \mathbb{N}$, $\rho \in Env_n$, $P \in Pterm$ and $U' \in Pterm\_con$ such that $\rho|_{FV(P)}$ is injective. Then:*

(1) If $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{sd},i,j)} U'$, then one of the following two (mutually exclusive) cases holds:
—(a) either $i, j \in \overline{0,n}$ and there exist x, y, P' such that $\rho(x) = i$, $\rho(y) = j$, $P'[\rho]_n \sim_{s,CS} U'$, and $\vdash_{OS} P \xrightarrow{\overline{x}y} P'$.
—(b) or $i \in \overline{0,n}$, $j = n+1$ and, for all $y \notin FV(P)$, there exist x, P' such that $\rho(x) = i$, $P'[\rho[y \leftarrow n+1]]_{n+1} \sim_{s,CS} U'$, and $\vdash_{OS} P \xrightarrow{\overline{x}(y)} P'$.
(2) If $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{rc},i,j)} U'$ and $y \notin FV(P)$, then one of the following two (mutually exclusive) cases holds:
—(a) $i, j \in \overline{0,n}$ and there exist x, P' such that $\rho(x) = i$, $P'[\rho[y \leftarrow j]]_n \sim_{s,CS} U'$, and $\vdash_{OS} P \xrightarrow{x(y)} P'$.
—(b) or $i \in \overline{0,n}$, $j = n+1$ and there exist x, P' such that $\rho(x) = i$, $P'[\rho[y \leftarrow n+1]]_{n+1} \sim_{s,CS} U'$, and $\vdash_{OS} P \xrightarrow{x(y)} P'$.
(3) If $\vdash_{CS} P[\rho]_n \xrightarrow{\tau} U'$, then there exists P' such that $P'[\rho]_n \sim_{s,CS} U'$ and $\vdash_{OS} P \xrightarrow{\tau} P'$.

Proof. Points (1)-(3) shall be proved together by induction on the structure of CS-derivation trees. For convenience, we shall use the same notations as for the listing of the CS rules in Figure 2, and thus, to avoid overlapping of metavariables, we rename P, U in the statements to be proved to P_0, U'_0. We have the following cases, according to the last applied rule:
- (OutC): Then $P_0[\rho]_n = Out_n(i, j, U)$ and $U'_0 = U$. Thus P_0 has the form $\overline{x}y.P$ such that $\rho(x) = i$, $\rho(y) = j$ and $P[\rho]_n = U$. Hence $i, j \in \overline{0,n}$. Applying (Out), $\vdash_{OS} \overline{x}y.P \xrightarrow{\overline{x}y} P$, therefore x, y, P are the desired items.

- (InpOC): Recall that $y \notin FV(P_0)$. Then $P_0[\rho]_n = Inp_n(i, U_0, \ldots, U_n, V)$, and thus P_0 has the form $x(y').P$ $\rho(x) = i$, $P[\rho[y' \leftarrow j']_n = U_j$ for all $j' \in \overline{0,n}$, and $P[\rho[y' \leftarrow n+1]]_{n+1} = V$. Since $y \notin FV(P_0)$, we can assume $y' = y$ by doing a renaming of the bound variable y' in $x(y').P$. Also, necessarily $i \in \overline{0,n}$, $j \in \overline{0,n}$, $U'_0 = U_j$. Applying (Inp), $\vdash_{OS} x(y).P \xrightarrow{x(y)} P$, making x and P the desired items.
- (InpNC): Recall that $y \notin FV(P)$. Then $P_0[\rho_n] = Inp_n(i, U_0, \ldots, U_n, V)$, and thus P_0 has the form $x(y').P$ (and again since $y \notin FV(P_0)$ we can assume $y' = y$) such that $\rho(x) = i$, $P[\rho[y \leftarrow j']_n = U_j$ for all $j' \in \overline{0,n}$, and $P[\rho[y \leftarrow n+1]]_{n+1} = V$. Also, necessarily $i \in \overline{0,n}$, $j = n+1$, $U'_0 = V$. Applying (Inp), $\vdash_{OS} x(y).P \xrightarrow{x(y)} P$, making x and P the desired items.
- (ReplC): Immediate, using IH and applying the rule (Repl).
- Any of the rules whose names start with "Lm" or "Com". Impossible, since $P_0[\rho]_n$ cannot have the construct *Lm* or *Com* on top.
- Any of the rules whose name starts with "ParcTo". Then $P_0[\rho]_n$ has the form $Parc(c, U, V)$. We have three subcases:
— c does not have the form $rcfg(n)$ or $pcfg(n)$. Impossible.
— c has the form $rcfg(n)$. Thus $c = (n, f_1, f_2, R)$ with $f_1 = id_n$, $f_2 = \mathbf{1}$ and $R = \{(n+1, 0)\}$. Then P_0 has the form $\nu\, y.P$, $P[\rho[y \leftarrow n+1]]_{n+1} = U$ and $V = Zero$. Let $\rho' = \rho[y \leftarrow n+1]$. ρ' is injective on $FV(P)$. Subcases, according to which particular "ParcTo" rule the last applied rule is:
—— (ParcToLm). Subcases, according to the last but one applied rule:
———- (LmTau). Then $U'_0 = Parc(rcfg(n), U', Zero)$ where $\vdash_{CS} U \xrightarrow{\tau} U'$, that is, $\vdash_{CS} P[\rho']_{n+1} \xrightarrow{\tau} U'$. By IH, for some P', $U' \sim_{s,CS} P'[\rho']_{n+1}$ and $\vdash_{OS} P \xrightarrow{\tau} P'$. Applying (Nu), $\vdash_{OS} \nu\, y.P \xrightarrow{\tau} \nu\, y.P'$. Moreover, $(\nu\, y.P')[\rho]_n = Parc(rcfg(n), P'[\rho']_{n+1}, Zero) = U'_0$, therefore $\nu\, y.P'$ is the desired item.
———- (LmVbO), with the verb being sd. Then $U'_0 = Parc(rcfg(n), U', Zero)$ where $\vdash_{CS} U \xrightarrow{(\mathsf{sd}, i_1, j_1)} U'$, that is, $\vdash_{CS} P[\rho']_{n+1} \xrightarrow{(\mathsf{sd}, i_1, j_1)} U'$, with $f_1 j_1 \neq \perp \neq f_1 j_1$, that is, with $i_1, j_1 \in \overline{0,n}$. By IH, for some P', x and z, $U' \sim_{s,CS} P'[\rho']_{n+1}$ and $\vdash_{OS} P \xrightarrow{\overline{x}z} P'$ with $\rho'(x) = i_1$ and $\rho'(z) = j_1$. Since $\rho'(y) = n+1$, we have $\rho(x) = i_1$, $\rho(y) = j_1$ and $y \notin \{x, z\}$. Applying (Nu), $\vdash_{OS} \nu\, y.P \xrightarrow{\overline{x}z} \nu\, y.P'$. Moreover, $(\nu\, y.P')[\rho]_n = Parc(rcfg(n), P'[\rho']_{n+1}, Zero) \sim_{s,CS} Parc(rcfg(n), U', Zero) = U'_0$, therefore x, z and $\nu\, y.P'$ are the desired items.
———- (LmVbO), with the verb being rc. To avoid overlapping, we rename the metavariable y from the statement at point (2) to z. Thus $z \notin FV(P_0)$. We can assume (performing, if necessary, a renaming of the bound variable y in $\nu\, y.P$) that $z \neq y$, and thus $z \notin FV(P)$. Then $U'_0 = Parc(rcfg(n), U', Zero)$ where $\vdash_{CS} U \xrightarrow{(\mathsf{rc}, i_1, j_1)} U'$, that is, $\vdash_{CS} P[\rho']_{n+1} \xrightarrow{(\mathsf{rc}, i_1, j_1)} U'$, with $f_1 i_1 \neq \perp \neq f_1 j_1$, that is, with $i_1, j_1 \in \overline{0,n}$. By IH, for some P' and x, $U' \sim_{s,CS} P'[\rho'[z \leftarrow j_1]]_{n+1}$ and $\vdash_{OS} P \xrightarrow{x(z)} P'$ with $\rho'(x) = i_1$. Note that $\rho(x) = i_1$ and $y \notin \{x, z\}$. Applying (Nu), $\vdash_{OS} \nu\, y.P \xrightarrow{x(z)} \nu\, y.P'$. Moreover, $(\nu\, y.P')[\rho[z \leftarrow j]]_n = Parc(rcfg(n), P'[\rho[z \leftarrow j][y \leftarrow n{+}1]]_{n+1}, Zero) = Parc(rcfg(n), P'[\rho'[z \leftarrow j]]_{n+1}, Zero) \sim_{s,CS} Parc(rcfg(n), U', Zero) = U'_0$, therefore x and $\nu\, y.P'$ are the desired items.

——- (LmSdPr). Then $U_0' = Parc(c', U', Zero)$ where $\vdash_{CS} U \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, that is, $\vdash_{CS} P[\rho']_{n+1} \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, with $f_1 i_1 \neq \bot$ and $(j_1, j_2) \in R$, that is, $i_1 \in \overline{0,n}$ and $(j_1, j_2) = (n+1, 0)$, where $c' = (n+1, id_{n+1}, (0, n+1), \emptyset)$. By IH, for some P', x, z, $U' = P'[\rho']_{n+1}$ and $\vdash_{OS} P \xrightarrow{\overline{x}z} P'$ with $\rho'(x) = i_1$ and $\rho'(z) = j_1 = n+1$, implying $z = y$. Moreover, $x \neq y$. Applying (Open), $\vdash_{OS} \nu\, y.P \xrightarrow{\overline{x}(y)} P'$. Moreover, $P'[\rho']_{n+1} \sim_{s,CS} Parc(c', P'[\rho']_{n+1}, Zero) = U_0'$, therefore x, y and $\nu\, y.P'$ are the desired items.
——— (LmVbN). Say the verb is rc (the case of sd is similar). Again we rename the metavariable y from the statement at point (2) to z. Thus $z \notin FV(P_0)$. We may assume $z \neq y$, thus $z \notin FV(P)$. Then $U_0' = Parc(c', U', Zero)$ where $\vdash_{CS} U \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, that is, $\vdash_{CS} P[\rho']_{n+1} \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, with $f_1 i_1 \neq \bot = f_1 j_1$ and $j_1 \notin Prj_1(R)$, that is, $i_1 \in \overline{0,n}$ and $j_1 = n+2$ (the latter because, by IH, $j_1 \in \overline{0, n+2}$, and values in $\overline{0, n+1}$ are not possible due to the conditions on j_1), where $c' = (n+1, id_n[n+2 \leftarrow n+1], \mathbf{I}, \{(n+1, 0)\})$. Set $\rho'' = \rho'[z \leftarrow n+2]$. By IH, for some P' and x, , $U' \sim_{s,CS} P'[\rho'']_{n+2}$ and $\vdash_{OS} P \xrightarrow{x(z)} P'$ with $\rho'(x) = i_1$. Moreover, $\rho(x) = i_1$ and $x \neq y$. Applying (Nu), $\vdash_{OS} \nu\, y.P \xrightarrow{x(z)} \nu\, y.P'$. Moreover, $(\nu\, y.P')[\rho[z \leftarrow n+1]]_{n+1} = Parc(rcfg(n+1), P'[\rho[z \leftarrow n+1][y \leftarrow n+2]], Zero) \sim_{s,CS} Parc(c', P'[\rho'']_{n+2}, Zero) = U_0'$, therefore x and $\nu\, y.P'$ are the desired items.
——— (LmRcOasN). Impossible, since $Im(f_1) = \overline{0,n}$ in $rcfg(n)$.
—— (ParcToCom). Impossible, since then there would exist in CS an inferrable transition from *Zero* to somewhere, which cannot be the case.
— c has the form $pcfg(n)$. Thus $c = (n, f_1, f_2, R)$ with $f_1 = id_n$, $f_2 = id_n$ and $R = \emptyset$. Then P_0 has the form $P|Q$, where $P[\rho]_n = U$ and $Q[\rho]_n = V$. Subcases, according to the last applied rule:
—— (ParcToLm). Subcases, according to the last but one applied rule:
——- (LmTau). Then $U_0' = Parc(pcfg(n), U', V)$ and $\vdash_{CS} U \xrightarrow{\tau} U'$, that is, $\vdash_{CS} P[\rho]_n \xrightarrow{\tau} U'$. By IH, for some $P', U' \sim_{s,CS} P'[\rho]_n$ with $\vdash_{OS} P \xrightarrow{\tau} P'$. Applying (ParL), $\vdash_{OS} P|Q \xrightarrow{\tau} P'|Q$. Moreover, $(P'|Q)[\rho]_n = Parc(pcfg(n), P'[\rho]_n, Q[\rho]_n) \sim_{s,CS} Parc(pcfg(n), U', V) = U_0'$, therefore $P'|Q$ is the desired item.
——- (LmVbO), with the verb being sd. Then $U_0' = Parc(pcfg(n), U', V)$ and $\vdash_{CS} U \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, that is, $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{sd},i_1,j_1)} U'$, with $f_1 i_1 \neq \bot \neq f_1 j_1$, that is, with $i_1, j_1 \in \overline{0,n}$. By IH, for some x, y and P', U' has the form $P'[\rho]_n$ with $\vdash_{OS} P \xrightarrow{\overline{x}y} P'$, $\rho(x) = i_1$ and $\rho(y) = j_1$. Applying (ParL), $\vdash_{OS} P|Q \xrightarrow{\overline{x}y} P'|Q$. Moreover, $(P'|Q)[\rho]_n = Parc(pcfg(n), P'[\rho]_n, Q[\rho]_n) \sim_{s,CS} Parc(pcfg(n), U', V) = U_0'$, therefore x, y and $P'|Q$ are the desired items.
——- (LmVbO), with the verb being rc. Recall that $y \notin FV(P_0)$. Then $U_0' = Parc(pcfg(n), U', V)$ and $\vdash_{CS} U \xrightarrow{(\mathsf{rc},i_1,j_1)} U'$, that is, $\vdash_{CS} P[\rho]_n \xrightarrow{(\mathsf{rc},i_1,j_1)} U'$, with $f_1 i_1 \neq \bot \neq f_1 j_1$, that is, $i_1, j_1 \in \overline{0,n}$. By IH, for some x and P', $U' \sim_{s,CS} P'[\rho[y \leftarrow j_1]]_n$ with $\vdash_{OS} P \xrightarrow{\overline{x}y} P'$ and $\rho(x) = i_1$. Applying (ParL), $\vdash_{OS} P|Q \xrightarrow{\overline{x}y} P'|Q$. Because $y \notin FV(Q)$, $V = Q[\rho]_n = Q[\rho[y \leftarrow j]]_n$. Moreover,

$(P'|Q)[\rho[y \leftarrow j_1]]_n = Parc(pcfg(n), P'[\rho[y \leftarrow j_1]]_n, Q[\rho[y \leftarrow j_1]]_n) \sim_{s,CS} Parc(pcfg(n), U', V) = U'_0$, therefore x and $P'|Q$ are the desired items.

——- (LmSdPr). Impossible, since $R = \emptyset$.

——- (LmVbN), say with the verb being rc (the case of sd is similar). Recall that $y \notin FV(P_0)$. Then $U'_0 = Parc((n+1, id_{n+1}, id_n, \emptyset), U', V)$ and $\vdash_{CS} U \stackrel{(\mathsf{rc},i_1,j_1)}{\longrightarrow} U'$, that is, $\vdash_{CS} P[\rho]_n \stackrel{(\mathsf{rc},i_1,j_1)}{\longrightarrow} U'$, with $f_1 i_1 \neq \bot = f_1 j_1$, that is, $i_1 \in \overline{0,n}$ and $j_1 = n+1$ (the latter because, by IH, $j_1 \in \overline{0,n+1}$, and values in $\overline{0,n}$ are not possible due to the conditions on j_1). Let $y \notin FV(P_0)$. By IH, for some x and P', $U' \sim_{s,CS} P'[\rho[y \leftarrow n+1]]_n$ with $\vdash_{OS} P \stackrel{x(y)}{\longrightarrow} P'$. Note that $y \notin FV(Q)$, and therefore $Q[\rho[y \leftarrow n+1]]_{n+1} = Q[\rho]_{n+1} \sim_{s,CS} \uparrow_n (Q[\rho]_n)$. Applying (ParL), $\vdash_{OS} P|Q \stackrel{x(y)}{\longrightarrow} P'|Q$. Moreover, $(P'|Q')[\rho[y \leftarrow n+1]]_{n+1} = Parc(pcfg(n+1), P'[\rho[y \leftarrow n+1]]_{n+1}, Q[\rho[y \leftarrow n+1]]_{n+1}) \sim_{s,CS} Parc(pcfg(n+1), P'[\rho[y \leftarrow n+1]]_{n+1}, \uparrow_n (Q[\rho]_n)) \sim_{s,CS} Parc((n+1, id_{n+1}, id_n, \emptyset), P'[\rho[y \leftarrow n+1]]_{n+1}, Q[\rho]_n) = Parc((n+1, id_{n+1}, id_n, \emptyset), U', V) = U'_0$ therefore x and $P'|Q$ are the desired items.

——- (LmRcOasN). Impossible, since $Im(f_1) = \overline{0,n}$ in $pcfg(n)$.

—— (ParcToCom). Subcases, according to the last but one applied rule:

——- (ComCon). Then $U'_0 = Parc(c, U', V')$, $\vdash_{CS} U \stackrel{(\mathsf{sd},i_1,j_1)}{\longrightarrow} U'$ and $\vdash_{CS} V \stackrel{(\mathsf{rc},i_2,j_2)}{\longrightarrow} V'$ with $(i_1,i_2),(j_1,j_2) \in Con(c)$, that is, $i_1 = i_2 \in \overline{0,n}$ and $j_1 = j_2 \in \overline{0,n}$. By IH, for some x, y and P', $U' \sim_{s,CS} P'[\rho]_n$ with $\vdash_{OS} P \stackrel{\overline{x}y}{\longrightarrow} P'$, $\rho(x) = i_1$ and $\rho(y) = j_1$. Choose z fresh for Q. By IH, for some x' and P', $V' \sim_{s,CS} Q'[\rho[z \leftarrow j_1]]_n$ with $\vdash_{OS} Q \stackrel{x'(z)}{\longrightarrow} Q'$ and $\rho(x') = i_1$. Because $x, x' \in FV(P_0)$ and $\rho(x) = \rho(x')$, $x = x'$. Also, $Q'[\rho[z \leftarrow j_1]]_n = Q'[\rho[z \leftarrow \rho(y)]]_n = Q'[z/y][\rho_n]$. Applying (ComOL), $\vdash_{OS} P|Q \stackrel{\tau}{\longrightarrow} P'|(Q'[z/y])$. Moreover, $(P'|(Q'[z/y]))[\rho]_n = Parc(pcfg(n), P'[\rho]_n, Q'[z/y][\rho]_n) \sim_{s,CS} Parc(pcfg(n), U', V') = U'_0$, therefore $P'|Q'$ is the desired item.

——- (ComON). Impossible, since $Im(f_1) = Im(f_2)$ in $pcfg(n)$.

——- (ComNN). Then $U'_0 = Parc((n, id_n, id_n, \{(n+1, n+1)\}), U', V')$, where $\vdash_{CS} U \stackrel{(\mathsf{sd},i_1,j_1)}{\longrightarrow} U'$ and $\vdash_{CS} V \stackrel{(\mathsf{rc},i_2,j_2)}{\longrightarrow} V'$ with $(i_1,i_2) \in Con(c)$ and $f_1 j_1 = f_2 j_2 = \bot$, that is, with $i_1 = i_2 \in \overline{0,n}$ and $j_1 = j_2 = n+1$. (The latter because, by IH, $j_1, j_2 \in \overline{0,n+1}$, and neither of j_1 and j_2 can be in $\overline{0,n}$ due to the above conditions imposed on them.) Choose y fresh for P_0. By IH, for some x and P', $U' \sim_{s,CS} P'[\rho[y \leftarrow n+1]]_{n+1}$ with $\vdash_{OS} P \stackrel{\overline{x}(y)}{\longrightarrow} P'$ and $\rho(x) = i_1$. By IH, for some x' and P', $V' \sim_{s,CS} Q'[\rho[y \leftarrow n+1]]_{n+1}$ with $\vdash_{OS} Q \stackrel{x'(y)}{\longrightarrow} Q'$ and $\rho(x') = i_1$. Because $x, x' \in FV(P_0)$ and $\rho(x) = \rho(x')$, $x = x'$. Applying (ComNL), $\vdash_{OS} P|Q \stackrel{\tau}{\longrightarrow} \nu\, y.P'|Q'$. Moreover, $(\nu\, y.P'|Q')[\rho]_n = Parc(rcfg(n), Parc(pcfg(n+1), P'[\rho[y \leftarrow n+1]]_{n+1}, Q'[\rho[y \leftarrow n+1]]_{n+1}), Zero) \sim_{s,CS} Parc((n, id_n, id_n, \{(n+1, n+1)\}), P'[\rho[y \leftarrow n+1]]_{n+1}, Q'[\rho[y \leftarrow n+1]]_{n+1}) = Parc((n, id_n, id_n, \{(n+1, n+1)\}), U', V') = U'_0$, therefore $P'|Q'$ is the desired item.

—— (ParcToLmSym) or (ParcToComSym). Perfectly analogously to (ParcToLm) and (ParcToCom). □

We now come to the main result of this subsection, for which the whole previous machinery was developed. It roughly states that the mapping $P \mapsto P[\rho]_n$ between the terms of the two systems preserves and reflects bisimilarity for injective environments ρ.

Proposition 19 *(OS versus CS) Let $P, Q \in Pterm$.*
(1) The following are equivalent:
—(a) $P \sim_{OS} Q$.
—(b) $P[\rho]_n \sim_{CS} Q[\rho]_n$ for all *$n \in I\!N$ and $\rho \in Env_n$ such that $\rho|_{FV(P)\cup FV(Q)}$ is injective.*
—(c) $P[\rho]_n \sim_{CS} Q[\rho]_n$ for some *$n \in I\!N$ and $\rho \in Env_n$ such that $\rho|_{FV(P)\cup FV(Q)}$ is injective.*
(2) The following are equivalent:
—(a) $P \equiv_{OS} Q$.
—(b) $P[\rho]_n \sim_{CS} Q[\rho]_n$ for all $n \in I\!N$ and $\rho \in Env_n$.

Proof. (1):
(a) implies (b): For any relation θ on *Pterm*, we define ω_θ to be the following relation on *Pterm_con*: $\{(P[\rho]_n, Q[\rho]_n) : n \in I\!N,\ \rho \in Env_n$ such that $\rho|_{FV(P)\cup FV(Q)}$ is injective$\}$. One can readily check, using Propositions 17 and 18, that ω_θ is a CS-simulation provided θ is an early simulation. Moreover, since $\omega_{\theta^\smile} = (\omega_\theta)^\smile$, we obtain that θ early bisimulation implies ω_θ CS-bisimulation. This yields the desired implication.
(b) implies (c): Trivial.
(c) implies (a): For any relation ω on *Pterm_con*, we define θ_ω to be the following relation on *Pterm*: $\{(P, Q)$: there exist $n \in I\!N$, $\rho \in Env_n$ such that $\rho|_{FV(P)\cup FV(Q)}$ is injective and $(P[\rho]_n, Q[\rho]_n) \in \omega\}$. One can readily check, using Propositions 17 and 18, that θ_ω is an early simulation provided ω is a CS-simulation. Moreover, since $\theta_{\omega^\smile} = (\theta_\omega)^\smile$, we obtain that ω CS-bisimulation implies θ_ω early bisimulation. This yields the desired implication.
(2):
(a) implies (b): Assume $P \equiv_{OS} Q$ and let $n \in I\!N$ and $\rho \in Env_n$. Let $\{i_1, \ldots, i_k\} = \rho(FV(P) \cup FV(Q))$, where $k = card(\rho(FV(P) \cup FV(Q)))$. For each $j \in \overline{1,k}$, let $\{x_j^1, \ldots, x_j^{p_j}\} = \rho^{-1}(i_j) \cap (FV(P) \cup FV(Q))$, where $p_j = card(\rho^{-1}(i_j) \cap (FV(P) \cup FV(Q)))$. Let $z_1, \ldots, z_k$ be distinct variables fresh for P and Q and let $\rho' \in Env_n$ defined as follows: $\rho'(z_j) = i_j$ for all $j \in \overline{1,k}$ and $\rho'(x) = \rho(x)$ if $x \notin \{z_1, \ldots, z_k\}$. Let $P' = P[z_1/x_1^1] \ldots [z_1/x_1^{p_1}] \ \ldots \ [z_k/x_k^1] \ldots [z_k/x_k^{p_k}]$ and
$Q' = Q[z_1/x_1^1] \ldots [z_1/x_1^{p_1}] \ \ldots \ [z_k/x_k^1] \ldots [z_k/x_k^{p_k}]$.
Then, applying Lemma 16.(1) a $card(FV(P) \cup FV(Q))$ number of times, we obtain:
$P'[\rho']_n = P[z_1/x_1^1] \ldots [z_1/x_1^{p_1}] \ \ldots \ [z_k/x_k^1] \ldots [z_k/x_k^{p_k}][\rho']_n =$
$P[z_1/x_1^1] \ldots [z_1/x_1^{p_1}] \ \ldots \ [z_k/x_k^1] \ldots [z_k/x_k^{p_k}][\rho'[z_k \leftarrow i_k]]_n =$
$P[z_1/x_1^1] \ldots [z_1/x_1^{p_1}] \ \ldots \ [z_k/x_k^1] \ldots [z_k/x_k^{p_k-1}][\rho'[x_k^{p_k} \leftarrow i_k]]_n =$
$\ldots =$
$P[\rho'[x_1^1 \leftarrow i_1] \ldots [x_1^{p_1} \leftarrow i_1] \ \ldots \ [x_k^1 \leftarrow i_k] \ldots [x_k^{p_k} \leftarrow i_k]] =$
$P[\rho]$.

Similarly, $Q'[\rho']_n = Q[\rho]_n$. Now, since ρ' is injective on $FV(P') \cup FV(Q') = \{z_1, \ldots, z_k\}$, by point (1), $P'[\rho']_n \sim_{CS} Q'[\rho']_n$, that is, $P[\rho]_n \sim_{CS} Q[\rho]_n$, as desired.
(b) implies (a): We want to prove $P \equiv_{CS} Q$. Let $x_1, \ldots, x_k$ be distinct variables and $y_1, \ldots, y_k$ variables and let $P' = P[y_1/x_1, \ldots, x_k/y_k]$ and $Q' = Q[y_1/x_1, \ldots, x_k/y_k]$. We need to prove that $P' \sim_{CS} Q'$. By point (1), it suffices to prove that, for all $n \in I\!N$ and $\rho \in Env_n$ with ρ injective on $FV(P') \cup FV(Q')$, $P'[\rho]_n \sim_{CS} Q'[\rho]_n$. So let n and ρ be as above. Applying Lemma 16.(1) a $2 * k$ number of times for P' and ρ and a $2 * k$ number of times for Q' and ρ, we find $\rho' \in Env_n$ such that $P'[\rho]_n = P[\rho']_n$ and $Q'[\rho]_n = Q[\rho']_n$. According to our hypothesis, $P[\rho']_n \sim_{CS} Q[\rho' b]]_n$, that is, $P'[\rho]_n \sim_{CS} Q'[\rho]_n$, as desired. □

Note: The statements of the above proposition remain true if consider the strong versions of bisimilarities in OS and CS instead of the weak ones.

B.3 The semantic domain captures bisimilarity of CS

Briefly stated, the reason why *Proc* captures bisimilarity on CS is that the multi-action-step behavior of processes in *Proc* is defined using the same means of interleaving and melting actions (into silent actions) as the one-action-step behavior of cpterms in CS. Indeed, the clauses for $\_ \tilde{\rightsquigarrow} (\_, \_)$ and $\_ \overset{(\_,\_)}{\tilde{\rightsquigarrow}} \_$ (which guide the word-combinatorics operations *par*, *com*, *lmerge*, which in turn guide the guide parallel composition in *Proc*), correspond quite exactly to the CS rules for parallel composition.

Recall that *Act*, ranged over by a, is the set of "loud" (concrete) actions, not including τ, that the system CS uses as actions the extended set $Act \cup \{\tau\}$, ranged over by α, and that the semantic notion of a process (i.e., element of *Proc*) employs sequences of loud actions, i.e., words over the alphabet *Act*, with ϵ denoting the empty word. In this section we let v range over Act^* and w over $(Act \cup \{\tau\})^*$. Recall the convention of writing $\vdash_{CS} U \xrightarrow{L} V$, for some language $L \subseteq (Act \cup \{\tau\})^*$, to indicate that there exists a word in L whose letters yield CS-inferrable transitions that lead from U to V. In particular, if the language L consists of a single word w, we write $\vdash_{CS} U \xrightarrow{w} V$ instead of $\vdash_{CS} U \xrightarrow{\{w\}} V$. We define the map $del_\tau : (Act \cup \{\tau\})^* \to Act^*$ so to delete all the occurrences of τ in its input. Notice that $del_\tau(w) = \epsilon$ if and only if $w \in \tau^*$.

In order to set clear the relationship between CS-bisimilarity and the semantic domain, let us rephrase the concept of CS-simulation. $\theta \subseteq Pterm\_con \times Pterm\_con$ is a CS-simulation iff, for all $(U, V) \in \theta$, $w \in (Act \cup \{\tau\})^*$ and U' such that $\vdash_{CS} U \xrightarrow{w} U'$, there exists $w' \in (Act \cup \{\tau\})^*$ and V' such that $del_\tau(w) = del_\tau(w')$, $(U', V') \in \theta$ and $\vdash_{CS} V \xrightarrow{w'} V'$. That is, a CS-(bi)simulation yields now a game where the players may take any (possibly empty) sequence of loud actions with arbitrary sequences of silent actions interspersed, and only the pattern consisting of loud actions needs to be matched with those of the opponent. One can readily check that this seemingly stronger version of CS-simulation is identical to the previous one.

Next, let us regard *Proc* as a transition system, by setting $\vdash_{Proc} \pi \xrightarrow{w} \pi'$ iff $(w, \pi') \in \pi$ for all $\pi, \pi' \in Proc$ and $w \in Act^*$. Thanks to its coalgebraic simplicity, this system is *internally fully abstract*, in that its (standard notion of) strong bisimilarity is the equality relation. Moreover, if we think of ϵ as its silent transition, the monoidality of *Proc* yields another remarkable property: its strong bisimilarity coincides with its (standard notion of) (weak) bisimilarity.

Now we can state the relationship between CS and the semantic domain. We have that, for all $w \in (Act \cup \{\tau\})^*$, a w-transition in CS corresponds to a $del_\tau(w)$-transition in *Proc*, making weak CS-bisimilarity and *Proc*-bisimilaity equivalent via the canonical interpretation of cpterms as processes.

In order to phrase the above statement transparently, let us cosmeticize the definition of the operations on *Proc* to match the constructs for cpterms, by introducing two auxiliary operators **Lmerge** and **Com** of type $Config \times Proc \times Proc \to Proc$ that match the auxiliary word-combinatorics operators *lmerge* and *com*. (Also, we use subscripting for the first numeric argument of **Out**, **Inp** and **Repl**.)

$\mathbf{Parc}(c, \pi_1, \pi_2) = \mathbf{Lmerge}(c, \pi_1, \pi_2) \cup \mathbf{Lmerge}(sym(c), \pi_2, \pi_1) \cup \mathbf{Com}(c, \pi_1, \pi_2) \cup \mathbf{Com}(sym(c), \pi_2, \pi_1)$.

$\mathbf{Lmerge}(c, \pi_1, \pi_2) = \{(w, \mathbf{Parc}(c', \pi_1', \pi_2')) : \exists w_1, w_2.\ \ (w_1, \pi_1') \in \pi_1,\ (w_2, \pi_2') \in \pi_2,\ (c', w) \in lmerge(c, w_1, w_2)\}$.

$\mathbf{Com}(c, \pi_1, \pi_2) = \{(w, \mathbf{Parc}(c', \pi_1', \pi_2')) : \exists w_1, w_2.\ \ (w_1, \pi_1') \in \pi_1,\ (w_2, \pi_2') \in \pi_2,\ (c', w) \in com(c, w_1, w_2)\}$.

$\mathbf{Zero} = \{(\epsilon, \mathbf{Zero})\}$.

$\mathbf{Out}_n = \lambda(i, j, \pi).\{(\epsilon, \mathbf{Out}_n(i, j, \pi))\} \cup \{((\mathsf{sd}, i, j) \# w, \pi') : (w, \pi') \in \pi\}$.

$\mathbf{Inp}_n = \lambda(i, h, \pi).\{(\epsilon, \mathbf{Inp}_n(i, h, \pi))\} \cup \{((\mathsf{rc}, i, j) \# w, \pi') : j \in \overline{0, n},\ (w, \pi') \in hj\}$
$\cup \{((\mathsf{rc}, i, n+1) \# w, \pi') : (w, \pi') \in \pi\}$.

$\mathbf{Repl}_n = \lambda\pi.\{(w, \mathbf{Parc}(pcfg(n), \pi', \mathbf{Repl}\, n\, \pi)) : \exists k.\, (w, \pi') \in \mathbf{Parc}^k(pcfg(n), \pi, \ldots, \pi)\}$.

(Above, $\mathbf{Parc}^k(c, \pi_1, \ldots, \pi_k)$ denotes $\mathbf{Parc}(c, \pi_0, \ldots, Parc(c, \pi_{k-1}, \pi_k)\ldots)$, that is, the parallel composition of the k processes in the configuration c – in particular, $\mathbf{Parc}^k(c, \pi_0, \ldots, \pi_k) = \pi_0$ if $k = 0$.)

Now we let $[\_] : Pterm\_con \to Proc$ denote the unique algebra homomorphism between the initial algebra *Pterm_con* and *Proc*, regarded as (single sorted) algebras. That is, $[Zero] = \mathbf{Zero}$, $[Out_n(i, j, U)] = \mathbf{Out}_n(i, j, [U])$, etc.

Proposition 20 *(CS versus the semantic domain)*
(1) For all $w \in (Act \cup \{\tau\})^*$ and $U, U' \in Pterm\_con$, $\vdash_{CS} U \xrightarrow{w} U'$ implies

$\vdash_{Proc} [U] \stackrel{del_\tau(w)}{\longrightarrow} [U']$.
(2) For all $v \in Act^*$ *and* $U \in Pterm\_con$, *if* $\vdash_{Proc} [U] \stackrel{v}{\longrightarrow} \pi'$, *then there exist* $w \in (Act \cup \{\tau\})^*$ *and* $U' \in Pterm\_con$ *such that* $del_\tau(w) = v$, $[U'] = \pi'$ *and* $\vdash_{CS} U \stackrel{w}{\longrightarrow} U'$.
(3) For all $U, U' \in Pterm\_con$, $U \sim_{CS} U'$ *iff* $[U] = [U']$.

For being able to prove this proposition, we need a lemma.

Lemma 21 *(1)* $\mathbf{Repl}_n(\pi) = \mathbf{Parc}(pcfg(n), \pi, \mathbf{Repl}_n(\pi))$ *for all* $\pi \in Proc$.
(2) Let $U, U', V, V' \in Pterm\_con$, $w, w' \in (Act \cup \{\tau\})^*$, $v \in Act^*$, *and* $c, c' \in Config$ *such that* $(c', v) \in par(c, del_\tau(w), del_\tau(w'))$. *If* $\vdash_{CS} U \stackrel{w}{\longrightarrow} U'$ *and* $\vdash_{CS} V \stackrel{w'}{\longrightarrow} V'$, *then there exists* $w'' \in (Act \cup \{\tau\})^*$ *such that* $del_\tau(w'') = v$ *and* $\vdash_{CS} Parc(c, U, U') \stackrel{w''}{\longrightarrow} Parc(c', V, V')$.
(3) Let $U, U' \in Pterm\_con$ *and* $w \in (Act \cup \{\tau\})^*$ *such that* $\vdash_{CS} Parc^k(pcfg(n), U \ldots, U) \stackrel{w}{\longrightarrow} U'$ *for some* $k \in \mathbb{N}$ *(where* $Parc^k(c, U_1, \ldots, U_k)$ *denotes* $Parc(c, U_1, \ldots, Parc(c, U_k)\ldots)$ *for all* c *and* $U_1, \ldots, U_k$*). Then* $\vdash_{CS} Repl_n(U) \stackrel{w}{\longrightarrow} Parc(pcfg(n), U', Repl_n(U))$.

Proof. (1): One can check that the set of pairs of the form
$(\mathbf{Parc}^k(pcfg(n), \pi, \ldots, \pi, \mathbf{Repl}_n(\pi)), \mathbf{Parc}^{k+1}(pcfg(n), \pi, \ldots, \pi, \mathbf{Repl}_n(\pi)))$,
with $k \in \mathbb{N}$, is a bisimilarity in *Proc*, hence, being included in the equality relation, yields in particular, for $k = 0$, the desired equality.
(2): By a straightforward, but tedious induction on the sum of lengths of w and w', using the recursive definition of *par*. If this sum is 0, then $U = U'$ and $V = V'$, and, because $par(c, \epsilon, \epsilon) = \{(c, \epsilon)\}$, it follows that $c = c'$ and $v = \epsilon$. Now the desired fact follows from the monoidality of *Proc*.

Now assume that this sum is nonzero. The desired fact follows by a tedious case-subcase-subsubcase analysis on the recursive definitions of *par*, *lmerge* and *com*, which in turn spawn cases on the definitions of $\_ \tilde{\rightsquigarrow} (\_, \_)$ and $\_ \stackrel{(\_,\_)}{\rightsquigarrow} \_$. We only pick one trace among these case scenarios. (The others follow similarly). Write c as (n, f_1, f_2, R). Let us assume, out of the four possible cases, that $(c', v) \in lmerge(c, del_\tau(w), del_\tau(w'))$. Moreover, let us assume, out of the four situations that lead to possibly non-empty results in the definition of *lmerge*, that $del_\tau(w)$ has the form $(vb, i_1, j_1)\#v'$, that $f_1\, i_1 \neq \bot \neq f_1\, j_1$, and that $(c', v) \in (vb, f_1 i_1, f_1 j_1)\#par(c, v', del_\tau(w'))$. Then v has the form $(vb, f_1 i_1, f_1 j_1)\#v''$ with $(c', v'') \in par(c, v', del_\tau(w'))$. Thus w has the form $(vb, i_1, j_1)\#w''$ and $del_\tau(w'') = v'$, and therefore $(c', v'') \in par(c, del_\tau(w''), del_\tau(w'))$, $\vdash_{CS} U \stackrel{(vb,i_1,j_1)}{\longrightarrow} U''$, and $\vdash_{CS} U'' \stackrel{w''}{\longrightarrow} U'$. By IH, there exists w''' such that $del_\tau(w''') = v''$ and $\vdash_{CS} Parc(c, U'', V) \stackrel{w'''}{\longrightarrow} Parc(c', U', V')$. Applying rule (LmVbO), we obtain
$\vdash_{CS} Parc(c, U, V) \stackrel{(vb, f_1 i_1, f_1 j_1)}{\longrightarrow} Parc(c', U'', V)$.
Thus $\vdash_{CS} Parc(c, U, V) \stackrel{(vb, f_1 i_1, f_1 j_1)\#w'''}{\longrightarrow} Parc(c', U'', V)$. And since $del_\tau((vb, f_1 i_1, f_1 j_1)\#w''') = (vb, f_1 i_1, f_1 j_1)\#v'' = v$, we obtain that $(vb, f_1 i_1, f_1 j_1)\#w'''$ is the desired word.

(3): The desired derived rule is obtained by k applications of the rule (*Replc*) and

an application of the couple of rules (ParcToLm) and (LmTau) or (ParcToLm) and (LmVbO), or (ParcToLm) and (LmVbN), depending on whether α is τ, (vb, i, j) with $i, j \in \overline{0,n}$, or $(vb, i, n+1)$ with $i \in \overline{0,n}$. □

Proof of Proposition 20.
(1): We perform lexicographic induction on two criteria: first on the length of the word w, and second, if w is nonempty, i.e., has the form $\alpha \# w'$, meaning $\vdash_{CS} U \xrightarrow{\alpha} U_1$ and $\vdash_{CS} U_1 \xrightarrow{w'} U'$ for some U_1, on the depth of a derivation tree for $\vdash_{CS} U \xrightarrow{\alpha} U_1$. (Expressed completely rigorously, the second criterion reads: on the depth of a CS-derivation tree of minimal depth among the CS-derivation trees for $U \xrightarrow{\alpha} U_1$, where U_1 is such that $\vdash_{CS} U_1 \xrightarrow{w'} U'$.)

If $w = \epsilon$, then $U = U'$ and the desired fact follows from the monoidality of *Proc*.

Now assume $w = \alpha \# w'$. By IH, $\vdash_{Proc} [U_1] \xrightarrow{del_\tau(w')} [U']$. We only discuss three cases (the others follow similarly):
- The last applied rule for deriving $\vdash_{CS} U \xrightarrow{\alpha} U_1$ was (OutC). Then $\alpha = (\mathsf{sd}, i, j)$ and $U = Out_n(i, j, U_1)$. By the definition of $\mathbf{Out}_n$, $\vdash_{Proc} \mathbf{Out}_n(i, j, [U_1]) \xrightarrow{(\mathsf{sd},i,j)\ \#\ del_\tau(w')} [U']$, that is, $\vdash_{Proc} [Out_n(i, j, U_1)] \xrightarrow{(\mathsf{sd},i,j)\ \#\ del_\tau(w')} [U']$, that is, $\vdash_{Proc} [U] \xrightarrow{del_\tau((\mathsf{sd},i,j)\ \#\ w')} [U']$, that is, $\vdash_{Proc} [U] \xrightarrow{del_\tau(w)} [U']$, as desired.
- The last applied rule was (ParcToCom). We have three subcases, according to the last but one applied rule being (ComCon), (ComON) or (ComNN). We only treat the case of (ComCon), the others following similarly. We have $\alpha = \tau$, $U = Parc(c, V_1, V_2), U_1 = Parc(c, V_1', V_2'), \vdash_{CS} V_1 \xrightarrow{(\mathsf{sd},i_1,j_1)} V_1', \vdash_{CS} V_2 \xrightarrow{(\mathsf{rc},i_2,j_2)} V_2'$, and $\vdash_{CS} Com(c, V_1, V_2) \xrightarrow{\tau} Parc(c, V_1', V_2')$, where $(i_1, i_2), (j_1, j_2) \in Con(c)$. By IH, $\vdash_{Proc} \mathbf{Parc}(c, [V_1'], [V_2']) \xrightarrow{del_\tau(w')} [U']$. By the definition of **Parc**, $[U'] = \mathbf{Parc}(c', \pi_1, \pi_2), \vdash_{Proc} [V_1'] \xrightarrow{v_1} \pi_1$ and $\vdash_{Proc} [V_2'] \xrightarrow{v_2} \pi_2$, for some $v_1, v_2 \in Act^*$, c' and π_1, π_2 such that $(c', del_\tau(w')) \in par(c, v_1, v_2)$. By the definition of *com*, $(c', del_\tau(w')) \in com(c, (\mathsf{sd}, i_1, j_1)\ \#\ v_1, (\mathsf{rc}, i_2, j_2)\ \#\ v_2)$. By IH, $\vdash_{Proc} [V_1] \xrightarrow{(\mathsf{sd},i_1,j_1)} [V_1']$ and $\vdash_{Proc} [V_2] \xrightarrow{(\mathsf{rc},i_2,j_2)} [V_2']$. By the monoidality of *Proc*, $\vdash_{Proc} [V_1] \xrightarrow{(\mathsf{sd},i_1,j_1)\#v_1} \pi_1$ and $\vdash_{Proc} [V_2] \xrightarrow{(\mathsf{sd},i_2,j_2)\#v_2} \pi_2$. By the definition of **Com**, $\vdash_{Proc} \mathbf{Parc}(c, [V_1], [V_2]) \xrightarrow{del_\tau(w')} \mathbf{Parc}(c', \pi_1, \pi_2)$, that is, $\vdash_{Proc} [Parc(c, V_1, V_2)] \xrightarrow{del_\tau(\tau \# w')} [U']$, that is, $\vdash_{Proc} [U] \xrightarrow{del_\tau(w)} [U']$, as desired.
- The last applied rule was (ReplC). Then $U = Repl(V)$, and $\vdash_{CS} Parc(pcfg(n), V, Repl_n(V)) \xrightarrow{\alpha} U_1$. By IH, $\vdash_{Proc} [Parc(pcfg(n), V, Repl_n(V))] \xrightarrow{del_\tau(\alpha)} [U_1]$, that is, $\vdash_{Proc} \mathbf{Parc}(pcfg(n), [V], \mathbf{Repl}_n([V])) \xrightarrow{del_\tau(\alpha)} [U_1]$. By the monoidality of *Proc*,
$\vdash_{Proc} \mathbf{Parc}(pcfg(n), [V], \mathbf{Repl}_n([V])) \xrightarrow{del_\tau(\alpha) \# del_\tau(w')} [U']$, that is,
$\vdash_{Proc} \mathbf{Parc}(pcfg(n), [V], \mathbf{Repl}_n([V])) \xrightarrow{del_\tau(w)} [U']$. By Lemma 21.(1), $\vdash_{Proc} \mathbf{Repl}_n([V]) \xrightarrow{del_\tau(w)} [U']$, that is, $\vdash_{Proc} [U] \xrightarrow{del_\tau(w)} [U']$, as desired.

(2) We perform lexicographic induction on two criteria: the first is the number of occurrences of constructs $Repl_n$ in the cpterm U, and the second is the depth of U. We split the discussion in cases according to the construct at the top of U, and only discuss the cases of output, parallel composition and replication (each of the others follow similarly to one of the sampled cases):
- Assume $U = Out_n(i, j, V)$. Then $\vdash_{Proc} \mathbf{Out}_n(i, j, [V]) \xrightarrow{v} \pi$. By the definition of $\mathbf{Out}_n$,
— Either $v = \epsilon$ and $\pi = [U]$ and we are done.
— Or $v = (\mathsf{sd}, i, j) \# v'$ and $\vdash_{Proc} [V] \xrightarrow{v'} \pi$. By IH, $\vdash_{CS} V \xrightarrow{w'} V'$ for some w' and V' such that $del_\tau(w') = v'$ and $[V'] = \pi$. Applying rule (OutC), $\vdash_{CS} Out_n(i, j, V) \xrightarrow{(\mathsf{sd},i,j)} V$, and thus $\vdash_{CS} Out_n(i, j, V) \xrightarrow{(\mathsf{sd},i,j) \# w'} V'$, that is, $\vdash_{CS} U \xrightarrow{(\mathsf{sd},i,j) \# w'} V'$. Moreover, $del_\tau((\mathsf{sd}, i, j) \# w') = v$ and we are done.
- Assume $U = Parc(c, U_1, U_2)$. Then $\vdash_{Proc} \mathbf{Parc}(c, [U_1], [U_2]) \xrightarrow{v} \pi$. By the definition of $Parc$, $\vdash_{Proc} [U_1] \xrightarrow{v_1} \pi_1$, $\vdash_{Proc} [U_2] \xrightarrow{v_2} \pi_2$, $(c', v) \in Parc(c', \pi_1, \pi_2)$ and $\pi = Parc(c', \pi_1, \pi_2)$ for some $v_1, v_2, c', \pi_1, \pi_2$. By IH, we find w_1, w_2, V_1, V_2 such that $\vdash_{CS} U_1 \xrightarrow{w_1} V_1$, $\vdash_{CS} U_2 \xrightarrow{w_2} V_2$, $[V_1] = \pi_1$, $[V_2] = \pi_2$, $del_\tau(w_1) = v_1$, and $del_\tau(w_2) = v_2$. By Lemma 21.(2), we find w'' such that $del_\tau(w'') = v$ and $\vdash_{CS} Parc(c, U_1, U_2) \xrightarrow{w''} Parc(c', V_1, V_2)$, i.e., $\vdash_{CS} U \xrightarrow{w''} Parc(c', V_1, V_2)$. Moreover, $[Parc(c', V_1, V_2)] = \pi$ and we are done.
- Assume $U = Repl_n(V)$. Then $\vdash_{Proc} \mathbf{Repl}_n([V]) \xrightarrow{v} \pi$. By the definition of $\mathbf{Repl}_n$, $\vdash_{Proc} \mathbf{Parc}^k(pcfg(n), [V], \ldots, [V]) \xrightarrow{v} \pi_1$ and $\pi = \mathbf{Parc}(pcfg(n), \pi_1, \mathbf{Repl}_n([V]))$ for some k and V. Thus $\vdash_{Proc} [Parc^k(pcfg(n), V, \ldots, V)] \xrightarrow{v} \pi_1$. By IH, we find w and U_1 such that $del_\tau(w) = v$, $[U_1] = \pi_1$ and $\vdash_{CS} Parc^k(pcfg(n), V, \ldots, V) \xrightarrow{w} U_1$. By Lemma 21.(3), $\vdash_{CS} Repl_n(V) \xrightarrow{w} Parc(pcfg(n), U_1, Repl_n(V))$. Moreover, $[Parc(pcfg(n), U_1, Repl_n(V))] = \pi$ and we are done.
(3): Let $\theta \subseteq Pterm\_con \times Pterm\_con$ be a CS-bisimulation. It follows immediately, using points (1) and (2), that $\{([U], [V]) : (U, V) \in \theta\}$ is a $Proc$-bisimulation. Now, if $U \sim_{CS} V$, then (U, V) belongs to some CS-bisimulation, hence $([U], [V])$ belongs to some $Proc$-bisimulation, thus $[U]$ and $[V]$ are $Proc$-bisimilar, hence equal.

Conversely, from points (1) and (2), we obtain that $\{(U, V) : [U] = [V]\}$ is a CS-bisimulation. Therefore $[U] = [V]$ implies that (U, V) belongs to some CS-bisimulation, hence $U \sim_{CS} V$. □

B.4 Putting the pieces together

Let us recall our constructions so far.

1. A map $[[\_]] : Pterm \to \prod_{n \in \mathbb{N}} Env_n \to Proc$ from the original syntax, via environments, to the semantic domain.
2. For each n, a map $\_[\_]_n : Pterm \times Env_n \to Pterm\_con$ from the original syntax, via environments, to the intermediary syntax of cpterms.

3. A map $[\_] : \mathit{Pterm\_con} \to \mathit{Proc}$ from the intermediary syntax to the semantic domain.

What we further need to notice is that the first map is the composition of the second and third maps, in the following sense:

Lemma 22 *$[[P]]\, n\, \rho = [P[\rho]_n]$ for all $P \in \mathit{Pterm}$, $n \in \mathbb{N}$ and $\rho \in \mathit{Env}_n$.*

Proof. Immediate induction on the structure of P. □

Now, Theorem 8 follows from Propositions 19.(2) and 20 and Lemma 22, and Theorem 9 from Propositions 19.(1) and 20 and Lemma 22.

C Related work in more detail

To our knowledge, this paper introduces the first denotational model for the π-calculus under weak bisimilarity which is both compositional and fully abstract. Our construction had to solve two rather orthogonal problems:

- How to deal with weak bisimilarity.
- How to deal with names.

We consider our approach novel w.r.t. both these problems. We therefore separate our discussion of related work according to proposed resolutions to these two problems.

C.1 On weak bisimilarity

In this subsection, we shall largely ignore issues regarding names, binding and scope extrusion.

Models for process calculi which are *intensional* (in that the identity of the semantic items does not coincide with bisimilarity or other targeted notion of operational equivalence) were proposed in [5, 18, 8, 4] (among other places). The framework of [8] (extending the one of [18]) offers facilities to define and reason "syntax freely" about weak bisimilarity in models which are already fully abstract w.r.t. strong bisimilarity, via a suitable hiding functor. The relationship between the abstract categorical machinery from there aimed at hiding the τ-actions and our packing and unpacking bijections between processes and their compact representations deserves future research.

Next we discuss work more related to our contribution, dealing specifically with full abstraction, where (extensional) equality in the semantic models captures weak bisimilarity. Although when defining our model for the π-calculus we focussed directly on compositionality, in what follows it will be useful to distinguish between *operational* denotational semantics [sic] and *compositional* denotational semantics in the following sense. (The term "operational semantics" is used in [21] to describe the denotational map that we discuss next under the name "denoational operational semantics"; we prefer the latter slightly oxymoronic phrase in order to prevent any confusion with the standard notion of operational semantics.) The terms of the system S to be given denotational semantics form an initial algebra over a certain signature Σ with the syntactic constructs as operations; moreover, they are organized as a coalgebra for the appropriate functor F expressing the dynamics of the system. Assume that F admits a final coalgebra P, and let us choose P as our semantic domain (of behaviors). The *operational denotational semantics*, called so because it follows automatically from the operational semantics, that is, from organizing the syntax as a coalgebra, is the unique map from S to P *obtained for free from the finality of* P, and it is by construction fully abstract w.r.t. the strong bisimilarity of S. (We recall that in many calculi, including π, weak bisimilarity can also be viewed as strong bisimilairty in an extended system.) On the other hand, it might be

the case that P itself (or a suitable subcoalgebra of P) can be organized as a Σ-algebra in a meaningful way *independently of the syntactic universe* S – then the unique map from S to P given by the *initiality* of S represents a *compositional denotational semantics* for S, which may or may not agree with the operational denotational semantics – in case it agrees, *compositional* full abstraction holds. Note that our semantics for the π-calculus in Section 3 is a compositional one in the sense introduced above (slightly generalized to cope with bindings). (We did not take the trouble of discussing an operational denotational semantics for π too; rather, we kept it implicit in the notion of bisimilarity.) For well-known reasons, including the ability to shift completely to the semantic domain and study its properties without any syntactic "intrusion", an insightful and useful (fully-abstract) denotational semantics should be compositional.

The paper [13] gives a fully abstract *operational* denotational semantics for the π-calculus under weak bisimilarity which is not compositional (Proposition 5.5.1 in the paper); moreover, it is not clear how semantic operations should be given on that domain so that compositionality is achieved. The thesis [14], which essentially applies the approach of [13] to many other calculi and mini-programming languages, introduces the following terminology regarding compositionality: it distinguishes between *a priori compositionality* and *a posteriori compositionality*. A priori compositionality is compositionality in our sense: one defines a domain and operations on it independently of the syntax, and then (by initiality or other means) a map from the syntax to this domain. On the other hand, a posteriori compositionality asks for the existence of semantic operations not on the whole semantic domain, but on the image of the operational-denotational-semantics map from the syntax to the domain. In our opinion, a posteriori compositionality (which says nothing more than that bisimilarity is a congruence) is only a half-way denotational-semantics achievement:

- On the one hand, it shows that the syntactic constructs can be seen to operate on *behaviors defined syntactically*;
- On the other hand, it does not show that arbitrary behaviors can be composed independently of the transition system, in a way that is consistent with the operational equivalence.

What we call "compositionality", coinciding, as mentioned, with what [14] calls "a priori compositionality", shall be henceforth, probably reflecting our semantic biases, referred to as "true compositonality".

Next we discuss related work in the area of truly compositional semantics, that is, semantics compositional in our sense and a priori compositional in the sense of [14]. We shall illustrate this related work on a small example. Consider a "mini CCS" system called MCCS, describing finite processes with prefix and parallel composition operations.

Let Act be an infinitely countable set of *actions* ranged over by a, b. We assume the existence of a bijection $a \mapsto \overline{a}$ on Act which is involutive, in that $\overline{\overline{a}} = a$ for all a. We consider an element $\tau \notin Act$ and let $Eact$, the set of *extended actions*, ranged over by α, β, be $Act \cup \{\tau\}$. We let $Pterm$, the set of process terms, ranged over by P, Q, be generated by the following grammar:

$P ::= 0 \mid \alpha.P \mid P|Q$

The system MCCS, listed below, has standard notions of strong bisimilarity and (weak) bisimilarity.

$$\frac{\cdot}{\alpha.P \xrightarrow{\alpha} P}(\mathsf{Pref}) \qquad \frac{P \xrightarrow{a} P' \quad Q \xrightarrow{\overline{a}} Q'}{P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'}(\mathsf{Comm})$$

$$\frac{P \xrightarrow{\alpha} P'}{P \,|\, Q \xrightarrow{\alpha} P' \,|\, Q}(\mathsf{ParL}) \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \,|\, Q \xrightarrow{\alpha} P \,|\, Q'}(\mathsf{ParR})$$

Peter Aczel's approach [3]. Let *Proc* be the greatest set X such that $X = \mathcal{P}(Eact \times X) - X$ with the identity function is the final coalgebra for the functor $F(X) = \mathcal{P}(Eact \times X)$. An F-coalgebra $(S, \alpha_S : S \to FS)$ is called a τ-coalgebra if for all $s \in Proc$ the following hold:
(a) $(\tau, \pi) \in \pi$.
(b) $(\tau, \pi') \in \pi$ and $(\alpha, \pi'') \in \pi'$ implies $(\alpha, \pi'') \in \pi$ for all α, π', π'' .
(c) $(\alpha, \pi') \in \pi$ and $(\tau, \pi'') \in \pi'$ implies $(\alpha, \pi'') \in \pi$ for all α, π', π''.

Let *Tproc*, the set of τ-processes, be the carrier of the greatest subcoalgebra of *Proc* which is a τ-coalgebra. Then $Tproc \subseteq \mathcal{P}(Eact \times Tproc)$ and this inclusion map is the final τ-coalgebra.[2] Now, for each α, let $\Rightarrow_\alpha$ be the relation on *Pterm* given by $P \Rightarrow_\alpha P'$ iff $P = P'$ or $\vdash_{MCCS} P \xrightarrow{\tau^* \alpha \tau^*} P'$. Then *Pterm* together with $\Rightarrow_\alpha$ yields a coalgebra in the standard way; this coalgebra is a τ-coalgebra, and therefore there exists a unique coalgebra morphism f : *Pterm* $\to$ *Tproc* between *Pterm* and *Tproc* whose kernel is the bisimilarity on *Pterm*. Hence f is a fully-abstract operational denotational semantics for MCCS.

Tproc is endowed with operations corresponding to the syntactic constructs on *Pterm* as follows:

- **Zero** : *Tproc*,
  Zero $= \{(\tau, Zero)\}$.
- **Pref** : $Eact \times Tproc \to Tproc$,
  Pref$(\tau, \pi) = \pi$,
  Pref$(a, \pi) = \{(\tau, \mathbf{Pref}(a, \pi))\} \cup \{(a, \pi') : (\tau, \pi') \in \pi\}$.
- **Par** : $Tproc \times Tproc \to Tproc$,
  Par$(\pi_1, \pi_2) = \{(\alpha, \mathbf{Par}(\pi_1', \pi_2')) : (\pi_1, \pi_2) \Rightarrow_\alpha (\pi_1', \pi_2')\}$,

where the relation $\Rightarrow_\alpha$ on $Tproc \times Tproc$ is defined similarly to the homonymous relation on *Pterm*, after organizing $Tproc \times Tproc$ as a transition system. Namely, first define the relation $\to_\alpha$ on $Tproc \times Tproc$ by $(\pi_1, \pi_2) \to_\alpha (\pi_1', \pi_2')$ iff one of the following holds:
- $\pi_1 = \pi_1'$ and $(\alpha, \pi_2') \in \pi_2$;
- $\pi_2 = \pi_2'$ and $(\alpha, \pi_1') \in \pi_1$;

[2] Note that the τ-coalgebras are essentially our ϵ-monoidal one-step coalgebras (inhabited by compact processes) from Section 2.

- $\alpha = \tau$ and there exists $a \in Act$ with $(a, \pi_1') \in \pi_1$ and $(\overline{a}, \pi_2') \in \pi_2$. Then define $\Rightarrow_\alpha$ to be the relational composition $(\rightarrow_\tau)^* \circ \rightarrow_\alpha \circ (\rightarrow_\tau)^*$, where $(\rightarrow_\tau)^*$ is the reflexive-transitive closure of $\rightarrow_\tau$.

The unique algebra morphism $g : Pterm \rightarrow Tproc$ represents a fully-abstract compositional denotational semantics for MCCS. Note that saying that g is fully abstract is the same as saying that $g = f$.

Jan Rutten's approach – the processes as terms paradigm [21]. (This approach was generalized to a categorical setting in [26].) Since the results of [21] apply to strong bisimilarity of transition systems, we first modify MCCS into one whose strong bisimilarity is the (weak) bisimilarity of MCCS. This can be done in a standard way. Let WMCCS be the system obtained from MCCS by adding the rules

$$\frac{\cdot}{P \xrightarrow{\tau} P}(\mathsf{TauRf}) \quad \frac{P \xrightarrow{\tau} P' \quad P' \xrightarrow{\alpha} P''}{P \xrightarrow{\alpha} P''}(\mathsf{TauTrL}) \quad \frac{P \xrightarrow{\alpha} P' \quad P' \xrightarrow{\tau} P''}{P \xrightarrow{\alpha} P''}(\mathsf{TauTrR})$$

Then any two process terms are bisimilar in MCCS iff they are weakly bisimilar in WMCCS. Thus in what follows, one can focuss on giving denotational semantics for WMCCS under *strong* bisimilarity.

We take again *Proc* to be the greatest set X such that $X = \mathcal{P}(Eact \times X)$, which with the identity function forms the final coalgebra for the functor $F(X) = \mathcal{P}(Eact \times X)$. Then since WMCCS yields an F-coalgebra in the standard way, there exists a unique F-coalgebra morphism $f : Pterm \rightarrow Proc$ whose kernel is the strong bisimilarity on $Pterm$ – f represents the operational denotational semantics. (Since bisimilarity is a congruence on pterms, this semantics is also a posteriori compositional.) In order to obtain true compositionality, one extends the syntax of process terms so as to incorporate (semantic) processes as constants. Namely, let *Epterm*, the set of *extended process terms*, ranged over by R, S, be given by the following grammar:
$P ::= 0 \mid \alpha.P \mid P|Q \mid \pi$
One also extends WMCCS with transitions given by the semantic structure of processes. namely, let EWMCCS be the system with rules having the same format as the ones of WMCCS (but where of course the term metavariables range over extended process terms rather than process terms), and additionally one axiom of the form $\pi \xrightarrow{\alpha} \pi'$ for each π, α, π' such that $(\alpha, \pi') \in \pi$ (we shall call such axioms "semantics axioms"). Now, EWMCCS also yields an F-coalgebra; let $f' : Epterm \rightarrow Proc$ be the unique coalgebra morphism with indicated type – note that f' is itself an operational denotational semantics for EWMCCS. Interestingly, f' can be used to provide a compositional denotational semantics for WMCCS, as follows. Define the operations:

- **Zero** : *Proc*,
  Zero $= f'0$.
- **Pref** : $Eact \times Proc \rightarrow Proc$,
  Pref$(\alpha, \pi) = f'(\alpha.\pi)$.
- **Par** : $Proc \times Proc \rightarrow Proc$,
  Par$(\pi_1, \pi_2) = f'(\pi_1|\pi_2)$.

So each semantic operation on *Proc* places its arguments in the context of the syntactic version of the operation (according to the extended syntax) and then records, via f', the behavior in the extended system WMCCS. Let $g : Pterm \to Proc$ be the unique algebra morphism with indicated type. Because the system WMCCS is in inductive tyft/tyxt format [21], it turns out that $f = g$, in other words, that g represents a fully-abstract denotational semantics for WMCCS under strong bisimilarity.

As it is, this compositional semantics (obtained by instantiating the very general technique from [21] to this particular case) does not look very appealing, as it does not seem to give much information about the action of the semantic operations on the argument processes. All the concrete information that we can straightforwardly recover from it is by unfolding the definition of the universal arrow f':

- $\mathbf{Zero} = f'0 = \{(\alpha, f'R) : \vdash_{EWMCCS} 0 \xrightarrow{\alpha} R\}$.
- $\mathbf{Pref}(\alpha, \pi) = f'(\alpha.\pi) = \{(\alpha', f'R) : \vdash_{EWMCCS} \alpha.\pi \xrightarrow{\alpha'} R\}$.
- $\mathbf{Par}(\pi_1, \pi_2) = f'(\pi_1|\pi_2) = \{(\alpha, f'R) : \vdash_{EWMCCS} \pi_1|\pi_2 \xrightarrow{\alpha} R\}$.

which illustrates the fact, already transparent in the use of the "operational" map f', that the semantic operators use the (global) syntactic transition system in their definition.

Of course, one needs to do better than that – it should be the case that the semantic operations on processes have definition-like properties with no reference to the syntax (i.e., to the system EWMCCS). By analyzing the structure of the rules of EWMCCS, one can actually achieve this. Namely, for each α, define the relations $\to_{\alpha,1}$ and $\Rightarrow_{\alpha,1}$ on *Proc* and $\to_{\alpha,2}$ and $\Rightarrow_{\alpha,2}$ on $Proc \times Proc$ as follows:

- $\pi \to_{\alpha,1} \pi'$ iff $(\alpha, \pi') \in \pi$.
- $\Rightarrow_{\alpha,1} = (\to_{\tau,1})^* \circ \to_{\alpha,1} \circ (\to_{\tau,1})^*$, where $(\to_{\tau,1})^*$ is the reflexive-transitive closure of $\to_{\tau,1}$.
- $(\pi_1, \pi_2) \to_{\alpha,2} (\pi_1', \pi_2')$ iff one of the following holds:
  — $\pi_1 = \pi_1'$ and $(\alpha, \pi_2') \in \pi_2$;
  — $\pi_2 = \pi_2'$ and $(\alpha, \pi_1') \in \pi_1$;
  — $\alpha = \tau$ and there exists $a \in Act$ such that with $(a, \pi_1') \in \pi_1$ and $(\overline{a}, \pi_2') \in \pi_2$.
- $\Rightarrow_{\alpha,2} = (\to_{\tau,2})^* \circ \to_{\alpha,2} \circ (\to_{\tau,2})^*$, where $(\to_{\tau,2})^*$ is the reflexive-transitive closure of $\to_{\tau,2}$.

(Note that, in $\Rightarrow_2$, we "rediscover" $\Rightarrow$ from Aczel's approach, except that now we work on *Proc* rather than *Tproc*.) One can show that, for all $\pi, \pi_1, \pi_2, \alpha, \alpha', R$, the following hold:

- $\vdash_{EWMCCS} 0 \xrightarrow{\alpha} R$ iff $R = 0$.
- $\vdash_{EWMCCS} \alpha.\pi \xrightarrow{\alpha'} R$ iff
  — either $\alpha = \tau$ and there exists π' such that $R = \pi'$ and $\pi \Rightarrow_{\alpha',1} \pi'$;
  — or $\alpha = \alpha'$ and there exists π' such that $R = \pi'$ and $\pi \Rightarrow_{\alpha,1} \pi'$;

– $\vdash_{EWMCCS}\ \pi_1|\pi_2 \xrightarrow{\alpha,2} R$ iff there exist π_1', π_2' such that $fR = \pi_1'|pi_2'$ and $(\pi_1, \pi_2) \Rightarrow_{\alpha,2} (\pi_1', \pi_2')$.

Consequently, we have

– $\mathbf{Zero} = \{(\alpha, \mathbf{Zero})\}$.
– $\mathbf{Pref}(\alpha, \pi) = \{(\alpha, \pi') : \pi \Rightarrow_{\tau,1} \pi'\} \cup \{(\alpha', \pi') : \alpha = \tau \text{ and } \pi \Rightarrow_{\alpha',1} \pi'\}$.
– $\mathbf{Par}(\pi_1, \pi_2) = \{(\alpha, \mathbf{Par}(\pi_1', \pi_2')) : (\pi_1, \pi_2) \Rightarrow_{\alpha,2} (\pi_1', \pi_2')\}$.

and the above expressions can be taken as purely semantic coalgebraic definitions of operations on *Proc* which yield *Proc* as a fully-abstract compositional model.

Our approach. The previous two approaches to the denotation of weak bisimilarity involved one-step-action final coalgebras. Each time, parallel composition needed information that laid deeper than at one-step depth in the behavior of the process arguments, requiring the construction of some auxiliary relations to explore the processes arbitrarily deep. Our approach is to make all traces of actions explicitly available to processes, so that after focussing on defining the appropriate shuffle operator on traces, parallel composition is defined naturally by directly combining the possible traces of the composites. The definition of parallel composition on traces is the one from the end of Section 2 – we recall it here.

A *trace* is a word of actions, i.e., an element of Act^*. We define the operation of *parallel composition of traces*, $par : Act^* \times Act^* \to \mathcal{P}(Act^*)$, recursively on the sum of the arguments's lengths as follows (recall that $\#$ denotes word concatenation; we also extend $\#$ to sets of words in the obvious way):
$par(\epsilon, \epsilon) = \{\epsilon\}$.

If $(w_1, w_2) \neq (\epsilon, \epsilon)$, then

$$\begin{aligned} par(w_1, w_2) = {} & \{a \# par(w_1', w_2) : w_1 = a \# w_1'\} \cup \\ & \{a \# par(w_1, w_2') : w_2 = a \# w_2'\} \cup \\ & \{par(w_1, w_2') : \exists\, a.\ w_1 = a \# w_1' \text{ and } w_2 = \overline{a} \# w_2'\} \end{aligned} .$$

Our semantic domain is the one from Sections 2 and 3, namely, the final monoidal multi-step coalgebra $(Proc, Proc \hookrightarrow \mathcal{P}(Act^* \times Proc))$. We define the semantic operations on *Proc* as follows (*Parc* being the same as | from the end of Section 2):

– $\mathbf{Zero} : Proc$,
$\mathbf{Zero} = \{(\epsilon, Zero)\}$.
– $\mathbf{Pref} : Eact \times Proc \to Proc$,
$\mathbf{Pref}(\tau, \pi) = \pi$,
$\mathbf{Pref}(a, \pi) = \{(\epsilon, \mathbf{Pref}(a, \pi))\} \cup \{(a \# w, \pi') : (w, \pi') \in \pi\}$.
– $\mathbf{Par} : Proc \times Proc \to Proc$,
$\mathbf{Par}(\pi_1, \pi_2) = \{(w, \mathbf{Par}(\pi_1', \pi_2')) : \exists w_1, w_2.\ w \in par(w_1, w_2), (w_1, \pi_1') \in \pi_1, (w_2, \pi_2') \in \pi_2\}$.

Then one can prove, using in a much simplified manner the technique described in this paper for the π-calculus, that the unique algebra morphism $f : Pterm \to Proc$ has its kernel equal to the (weak) bisimilarity on $Pterm$, and therefore represents a fully-abstract compositional semantics for it.

Compared to the other two mentioned approaches, a virtue of our semantics is essentially the one discusses in Section 2 at the comparison between processes and compact processes – the separation of concerns (first consider trace combinatorics without any reference to processes, and then define process operations), resulting in a cleaner compositionality, in that the problematic parallel composition is defined without reference to any auxiliary transitive closure relation on processes. When faced with defining parallel composition, the other two approaches effectively regard the semantic space and its product with itself as regular transition systems and perform a reachability analysis on them. Instead, in our approach the continuations of the composite are determined transparently by the continuations of the components, because all reachable states are already there in the denotation. Our semantics reflects the following view: in order to understand how some processes can interact silently, one really needs to understand for each of them what sequence of actions it is willing to commit to – making such traces available before hand is a more "honest" and more transparent approach than exploring them on ad-hoc bases.

C.2 On handling names

There is a vast literature on operational and denotational techniques for dealing with name bindings in λ-calculi and process calculi, all implementing in one way or another ideas similar to those of the pioneering work of de Bruijn on representing λ-calculus terms. For the π-calculus, the issue of scope extrusion raises additional difficulties usually employing some notion of level relative to which freshness can be considered. Operational and quasi-operational semantics encodings (partly aimed towards efficiency) were developed in [7, 12, 17], among other places.

More related to our work are the domain-theoretic denotational models developed in [10, 24] for the π-calculus under *strong* bisimilarity. These approaches achieve a presentation of the semantics which is arguably more compact and clearly more structured than ours, by employing the machinery of functor categories. While the domain theoretic setting, requiring finite approximability [1], is not suitable for handling the infinitely branching aspect of weak bisimilarity and so needs to be replaced with something less fancy but more flexible such as coialgebra, the functor category approach is rather independent of the base category (be it of domains or of sets), and thus in principle we could have phrased our semantics in this convenient categorical language – the main requirement for such an approach, namely that our families of semantic operators are uniform with respect to renaming and thus form natural transformations, seems to be true in our case too. However, it is not clear to us yet whether such an approach would bring here similar structuring advantages to the ones brought there – this is because our processes may shift, via traces, not only one level up as for strong

bisimilarity, but an arbitrary number of levels, breaking, as far as we see, the direct applicability of the function space construction for functor categories to the semantics of the operators, and thus the possibility of an index-free treatment.

Apart from the issue regarding the categorical formulation, there is another distinction between our approach and the ones from [10, 24] for strong bisimilarity. There, a process placed in parallel with other processes is swapping its channels *eagerly* to keep up with any change in the channel topology introduced by one of its parallel peers. In our case, the only thing that changes is the interface (configuration), so that a process "remains itself", just that after a change in the channel topology its interaction with the outside world will be filtered differently when needed; in other words, changes in topology are reflected "lazily" in the observable behavior of processes. To illustrate this distinction, let us assume that process π_1 is in parallel with process π_2 and that π_2 receives a new channel – at that very moment, according to the laws of [10] the process π_1 is not the same process any more, but is instantly updated in order to keep up with this global change, namely with the fact that the composite system has improved its knowledge – technically, π_1 performs a shift from level n to level $n + 1$ (which also involves some swapping of channels). According to our laws, it is the configuration, and not the process directly, that acknowledges the presence of a new channel, presence will not be seen in the behavior of π_1 until the moment it needs to send a new channel of its own, case in which what is actually sent is not the next, but the next-next channel. We have opted for this lazy configuration-based approach because in our more complex case the presence of traces of actions makes an alternative eager swapping apparently less tractable.

The models from [13] for the π-calculus under strong and weak bisimilarity are, like ours, coalgebraic (although, as mentioned, compositionality issues are not considered there). The approach there is to consider finite sets a of channels as part of the structure of functor giving by finality the semantic domain. The sets a are to be interpreted as estimates of what a process knows, with behavior that would try to violate the estimates falling into undefinedness. The consideration of sets of names as norms for behavior is common with the approach from [24] and the untyped character of the semantic items is common with our approach. Because of the particular way in which they define the behavior functor, bisimilarity, and not bisimilarity congruence is their main character, unlike in [10, 24] and in our case. (See [9, 25] for a comprehensive comparison between various operational and or/denotational models.)

References

1. S. Abramski. A domain equation for bisimulation. *Inf. Comput.*, 92(2):161–218, 1991.
2. P. Aczel. *Non-Well-Founded Sets*. Stanford, 1988.
3. P. Aczel. Final universes of processes. In *MFPS'93*, pages 1–28, 1993.
4. M. G. Buscemi and U. Montanari. A first order coalgebraic model of pi-calculus early observational equivalence. In *CONCUR*, pages 449–465, 2002.
5. G. L. Cattani and P. Sewell. Models for name-passing processes: interleaving and causal. In *LICS 2000*, pages 322–333, 2000.
6. C. Cîrstea. Semantic constructions for hidden algebra. In *WADT'98*, pages 63–78, 1999.
7. G. L. Ferrari, U. Montanari, and P. Quaglia. A pi-calculus with explicit substitutions. *Theor. Comput. Sci.*, 168(1):53–103, 1996.
8. M. Fiore, G. L. Cattani, and G. Winskel. Weak bisimulation and open maps. In *LICS'99*, pages 67–76, 1999.
9. M. Fiore and S. Staton. Comparing operational models of name-passing process calculi. *Inf. Comput.*, 204(4):524–560, 2006.
10. M. P. Fiore, E. Moggi, and D. Sangiorgi. A fully-abstract model for the π-calculus. In *LICS'96*, pages 43–54, 1996.
11. J. F. Groote and F. Vaandrager. Structured operational semantics and bisimulation as a congruence. *Inf. Comput.*, 100(2):202–260, 1992.
12. D. Hirschkoff. Handling substitutions explicitely in the pi-calculus. Explicit Substitutions: Theory and Applications to Programs and Proofs, 1999.
13. F. Honsell, M. Lenisa, U. Montanari, and M. Pistore. Final semantics for the pi-calculus. In *PROCOMET'98*, pages 225–243, 1998.
14. M. Lenisa. *Themes in Final Semantics*. Dipartimento di Informatica, Universita' di Pisa, TD 6, 1998.
15. R. Milner. *Communicating and mobile systems: the π-calculus*. Cambridge, 2001.
16. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, parts i and ii. *Inf. Comput.*, 100(1):1–77, 1992.
17. U. Montanari and M. Pistore. Structured coalgebras and minimal HD-automata for the π-calculus. *Theor. Comput. Sci.*, 340(3):539–576, 2005.
18. M. Nielsen and A. Chang. Observe behaviour categorically. In *FST&TCS'95*, pages 263–278, 1995.
19. G. Roşu. *Hidden Logic*. PhD thesis, University of California at San Diego, 2000.
20. M. Rössiger. From modal logic to terminal coalgebras. *Theor. Comput. Sci.*, 260(1-2):209–228, 2001.
21. J. J. M. M. Rutten. Processes as terms: Non-well-founded models for bisimulation. *Math. Struct. Comp. Sci.*, 2(3):257–275, 1992.
22. J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.
23. D. Sangiorgi and D. Walker. *The π-calculus. A theory of mobile processes*. Cambridge, 2001.
24. I. Stark. A fully-abstract domain model for the π-calculus. In *LICS'96*, pages 36–42, 1996.
25. S. Staton. *Name-passing process calculi: operational models and structural operational semantics*. PhD thesis, University of Cambridge, 2007.
26. D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *LICS'97*, pages 280–291, 1997.