Appendix

This Appendix contains additional definitions, explanations, examples, and the formal proofs.

A Process Calculi

A.1 The Pi-Calculus

We introduce the pi-calculus as described e.g. in [14, 13]. We introduce three different variants of the monadic pi-calculus. The first is the full synchronous pi-calculus including mixed choice.

Definition 9 (π_{m}). *The set of process terms of the* synchronous pi-calculus (with mixed choice)*, denoted by $\mathcal{P}_{\mathrm{m}}$, is given by*

$$P ::= P_1 \mid P_2 \quad | \quad \checkmark \quad | \quad (\nu n)\, P \quad | \quad !P \quad | \quad \sum_{i \in I} \pi_i.P_i$$

$$\pi ::= \overline{y}\langle z \rangle \quad | \quad y(x) \quad | \quad \tau$$

for some names $n, x, y, z \in \mathcal{N}$ and a finite index set I.

The interpretation of the defined process terms is as usual. *Restriction* $(\nu n)\, P$ restricts the scope of the name n to the definition of P. The *parallel composition* $P_1 \mid P_2$ defines the process in which P_1 and P_2 may proceed independently, possibly interacting using shared links. $!P$ denotes *replication*. The process term $\sum_{i \in I} \pi_i.P_i$ represents *finite guarded choice*; as usual, the sum $\sum_{i \in \{\, 1,\ldots,n \,\}} \pi_i.P_i$ is sometimes written as $\pi_1.P_1 + \ldots + \pi_n.P_n$ and $\mathsf{0}$ abbreviates the empty sum, i.e., where $I = \emptyset$.

The capabilities of the pi-calculus are the (replicated) input prefix $y(x)$, the output prefix $\overline{y}\langle z \rangle$, and the prefix τ, where the capability of a choice-term is the conjunction of the prefixes of all its branches. The input prefix $y(x)$ is used to describe the ability of receiving the value x over link y and, analogously, the output prefix $\overline{y}\langle z \rangle$ describes the ability to send a value z over link y. The prefix τ describes the ability to perform an internal, not observable action. Prefixes and choice are guards, and all their subterms are guarded. Hence, the branches of sums are always guarded.

The definitions of free and bound names are completely standard, i.e., names are bound by restriction and as parameter of input and $\mathsf{n}(P) = \mathsf{fn}(P) \cup \mathsf{bn}(P)$ for all P. We naturally extend substitutions to co-names, i.e., $\forall \overline{n} \in \overline{\mathcal{N}}\ .\ \sigma(\overline{n}) = \overline{\sigma(n)}$ for all substitutions σ.

As usual, the continuation $\mathsf{0}$ is often omitted, so $y(x).\mathsf{0}$ becomes $y(x)$. In addition, for simplicity in the presentation of examples, we sometimes omit an action's object when it does not effectively contribute to the behaviour of a term. Typically, we do this when it would be enough to use a CCS-like example, but the monadic pi-calculus would force us to carry *some* object along that would

never be used on a receiver side, e.g. as in $y(x).\mathsf{0}$, which would be written as $y.\mathsf{0}$ or just y. Moreover, let $(\nu\tilde{x})\,P$ abbreviate the term $(\nu x_1)\ldots(\nu x_n)\,P$.

The expressive power of π_{m} is compared to two of its subcalculi: π_{s}, the pi-calculus with separate choice, and π_{a}, the asynchronous pi-calculus. In π_{s}, both output and input can be used as guards, but within a single choice term either there are no input or no output guards, i.e., we have input- and output-guarded choice, but no mixed choice.

Definition 10 (π_{s}). *The set of process terms of the* pi-calculus with separate choice*, denoted by $\mathcal{P}_{\mathrm{s}}$, is given by*

$$P ::= P_1 \mid P_2 \quad | \quad \checkmark \quad | \quad (\nu n)\,P \quad | \quad !P \quad | \quad \sum_{i\in I} \pi_i^O.P_i \quad | \quad \sum_{i\in I} \pi_i^I.P_i$$

$$\pi^O ::= \overline{y}\langle z\rangle \quad | \quad \tau \qquad \text{and} \qquad \pi^I ::= y(x) \quad | \quad \tau$$

for some names $n, x, y, z \in \mathcal{N}$ and a finite index set I.

As expected, the definitions of π_{s} and π_{m} differ in the definition of choice only.

Asynchronous variants of the pi-calculus were introduced independently by [10] and [3]. In asynchronous communication, a process has no chance to directly determine, i.e., without a hint by another process, whether a value sent by it was already received or not. To model that fact in π_{a}, output actions are not allowed to guard a process different from $\mathsf{0}$. Accordingly, the interpretation of output guards within a choice construct is delicate. Here, we use the standard variant of π_{a}, where choice is not allowed at all. Since π_{a} has no choice, and thus no nullary choice, we include $\mathsf{0}$ as a primitive.

Definition 11 (π_{a}). *The set of process terms of the* asynchronous pi-calculus*, denoted by $\mathcal{P}_{\mathrm{a}}$, is given by*

$$P ::= \mathsf{0} \quad | \quad P_1 \mid P_2 \quad | \quad \checkmark \quad | \quad (\nu n)\,P \quad | \quad !P \quad | \quad \tau.P \quad | \quad \overline{y}\langle z\rangle \quad | \quad y(x).P$$

for some names $n, x, y, z \in \mathcal{N}$.

The *operational semantics* of π_{m}, π_{s}, and π_{a} are jointly given by the transition rules in Figure 3, where the indices m, s, and a refer to rules of π_{m}, π_{s}, and π_{a}, respectively. The structural congruence, denoted by $\equiv$, is given by the rules:

$$P \equiv Q \ \text{ if } P \equiv_\alpha Q \qquad (\nu n)\,\mathsf{0} \equiv \mathsf{0} \qquad P \mid \mathsf{0} \equiv P \qquad P \mid Q \equiv Q \mid P$$

$$P \mid (Q \mid R) \equiv (P \mid Q) \mid R \qquad (\nu n)\,(\nu m)\,P \equiv (\nu m)\,(\nu n)\,P$$

$$P \mid (\nu n)\,Q \equiv (\nu n)\,(P \mid Q) \ \text{ if } n \notin \mathsf{fn}(P) \qquad !P \equiv P \mid !P$$

A.2 The Join-Calculus

Now, we introduce the join-calculus as described e.g. in [7] or [27].

$$\textsc{Tau}_{\mathrm{m,s}} \quad \textstyle\sum_{i\in I} \pi_i.P_i \longmapsto P_i \quad \text{if } \exists i \in I \,.\, \pi_i = \tau$$

$$\textsc{Com}_{\mathrm{m,s}} \quad \begin{array}{c} \textstyle\sum_{i\in I_1} \pi_i.P_i \mid \sum_{j\in I_2} \pi_j.P_j \longmapsto \{\, {}^{z}/_{x} \,\}\, P_i \mid P_j \\ \text{if } \exists i \in I_1 \,.\, \pi_i = y(x) \wedge \exists j \in I_2 \,.\, \pi_j = \overline{y}\langle z\rangle \end{array}$$

$$\textsc{Tau}_{\mathrm{a}} \quad \tau.P \longmapsto P \qquad \textsc{Com}_{\mathrm{a}} \quad y(x)\,.P \mid \overline{y}\langle z\rangle \longmapsto \{\, {}^{z}/_{x} \,\}\, P$$

$$\textsc{Par}_{\mathrm{m,s,a}} \quad \frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \textsc{Res}_{\mathrm{m,s,a}} \quad \frac{P \longmapsto P'}{(\nu n)\, P \longmapsto (\nu n)\, P'}$$

$$\textsc{Cong}_{\mathrm{m,s,a}} \quad \frac{P \equiv Q \quad Q \longmapsto Q' \quad Q' \equiv P'}{P \longmapsto P'}$$

Fig. 3. Reduction Semantics of π_{m}, π_{s}, and π_{a}.

Definition 12 (J). *The set of process terms of the* join-calculus*, denoted by* $\mathcal{P}_{\mathrm{J}}$*, is given by*

$$\begin{array}{llllllll} P & ::= \mathsf{0} & \mid & y\,\langle z\rangle & \mid & P_1 \mid P_2 & \mid & \mathsf{def}\; D \;\mathsf{in}\; P \quad \mid \quad \checkmark \\ J & ::= y\,(x) & \mid & J_1 \mid J_2 & \textit{and} & D & ::= & J \triangleright P \quad \mid \quad D_1 \wedge D_2 \end{array}$$

for some names $x, y, z \in \mathcal{N}$*.*

The interpretation is again as usual. $\mathsf{0}$, $y\,\langle z\rangle$, and $P_1 \mid P_2$ define the empty process, an output capability, and parallel composition similar to π_{a}. A *definition* $\mathsf{def}\; D \;\mathsf{in}\; P$ defines a new receiver on fresh names, where D consists of one or several elementary definitions $J \triangleright P$ connected by $\wedge$, J potentially joins several reception patterns $y\,(x)$ connected by $\mid$, and P is a process. Compared to the pi-calculus, join patterns represent (recurrent) input capabilities that are matched against outputs in order to instantiate and unguard an instance of a guarded subterm. Note that the definition construct $\mathsf{def}\; D \;\mathsf{in}\; P$ unifies the concepts of restriction, input capabilities, and replication of the pi-calculus. In $\mathsf{def}\;\; (J_1 \triangleright P_1) \wedge \ldots \wedge (J_n \triangleright P_n) \;\; \mathsf{in}\; P$ the subterms $P_1, \ldots, P_n$ are guarded while P is an unguarded subterm. Again, we omit an action's object when it does not effectively contribute to the behaviour of a term, e.g. we write $\mathsf{def}\; y\,(x) \triangleright \mathsf{0} \;\mathsf{in}\; y\,\langle z\rangle$ as $\mathsf{def}\; y \triangleright \mathsf{0} \;\mathsf{in}\; y$.

The sets of *received variables* $\mathsf{rv}(\cdot)$ and *defined variables* $\mathsf{dv}(\cdot)$ are inductively defined as:

$$\begin{array}{rclcrcl} \mathsf{rv}(y\,(x)) & \triangleq & \{\, x \,\} & \qquad & \mathsf{rv}(J_1 \mid J_2) & \triangleq & \mathsf{rv}(J_1) \uplus \mathsf{rv}(J_2) \\ \mathsf{dv}(y\,(x)) & \triangleq & \{\, y \,\} & & \mathsf{dv}(J_1 \mid J_2) & \triangleq & \mathsf{dv}(J_1) \cup \mathsf{dv}(J_2) \\ \mathsf{dv}(J \triangleright P) & \triangleq & \mathsf{dv}(J) & & \mathsf{dv}(D_1 \wedge D_2) & \triangleq & \mathsf{dv}(D_1) \cup \mathsf{dv}(D_2) \end{array}$$

By convention, the received variables of composed join patterns have to be pairwise distinct. The bound names $\mathsf{bn}(P)$ of P are the union of the received and defined variables in P. The free names of P are defined by its set of *free variables*,

where $\mathsf{fv}(\cdot)$:

$$\begin{aligned}
\mathsf{fv}(J \triangleright P) &\triangleq \mathsf{dv}(J) \cup (\mathsf{fv}(P) \setminus \mathsf{rv}(J)) \\
\mathsf{fv}(D_1 \wedge D_2) &\triangleq \mathsf{fv}(D_1) \cup \mathsf{fv}(D_2) \\
\mathsf{fv}(y\,\langle z\rangle) &\triangleq \{\, y, z \,\} \\
\mathsf{fv}(\mathsf{def}\ D\ \mathsf{in}\ P) &\triangleq (\mathsf{fv}(P) \cup \mathsf{fv}(D)) \setminus \mathsf{dv}(D) \\
\mathsf{fv}(P_1 \mid P_2) &\triangleq \mathsf{fv}(P_1) \cup \mathsf{fv}(P_2)
\end{aligned}$$

Moreover, [7] define the *core join-calculus* cJ as a subcalculus of J that restricts definitions to the form $\mathsf{def}\, y_1\,(x_1) \mid y_2\,(x_2) \triangleright P_1\ \mathsf{in}\ P_2$, i.e., in the core join-calculus definitions consist of a single elementary definition of exactly two reception patterns.

The operational semantics of the join-calculus is given by an extension of the chemical approach in [1]. The rules operate on so-called solutions $\mathcal{R} \vdash \mathcal{M}$, where $\mathcal{R}$ and $\mathcal{M}$ are multisets. As done in [7], we only mention the elements of the multisets that participate in the rule, separated by commas. The semantics is given by the so-called heating and cooling rules

$$\begin{array}{lrcrcl}
\textsc{Join}_\textsc{j} & & \vdash\ P \mid Q & \rightleftharpoons & \vdash\ P, Q \\
\textsc{And}_\textsc{j} & D \wedge E\ \vdash & & \rightleftharpoons & D, E\ \vdash \\
\textsc{Def}_\textsc{j} & & \vdash\ \mathsf{def}\ D\ \mathsf{in}\ P & \rightleftharpoons & \sigma_{dv}(D)\ \vdash\ \sigma_{dv}(P)
\end{array}$$

and the reduction rule

$$\textsc{Red}_\textsc{j} \quad J \triangleright P\ \vdash\ \sigma_{rv}(J) \quad \longmapsto \quad J \triangleright P\ \vdash\ \sigma_{rv}(P)$$

where σ_{dv} instantiates the defined variables in D to distinct fresh names, and σ_{rv} substitutes the transmitted names for the distinct received variables. Note that the heating and cooling rules describe the underlying structural congruence on processes, i.e., if $P \rightleftharpoons Q$, $Q \longmapsto Q'$, and $Q' \rightleftharpoons P'$ then also $P \longmapsto P'$. In the following, we write $P \equiv Q$ if P and Q differ only by applications of the heating and cooling rules.

A.3 Communicating Sequential Processes (CSP)

The language CSP was introduced by Hoare [28, 29]. We consider two variants of CSP that (instead of arbitrary action events) use in- and output prefixes. The first variant $\mathrm{CSP}_{\mathrm{in}}$ allows input guards in the choice construct.

Definition 13 ($\mathrm{CSP}_{\mathrm{in}}$). *The set of process terms of the* CSP-calculus with input guarded choice*, denoted by $\mathcal{P}_{\mathrm{in}}$, is given by*

$$\begin{aligned}
P &::= 0 \ \mid\ P \setminus n \ \mid\ P_1 \parallel P_2 \ \mid\ \checkmark \ \mid\ y!z \to P \ \mid\ [\,C\,] \ \mid\ \star[\,C\,] \\
C &::= G \ \mid\ G \,\Box\, C \quad \textit{and} \quad G ::= y?\,(x) \to P \ \mid\ \tau \to P
\end{aligned}$$

for some names $n, x, y, z \in \mathcal{N}$.

$P \setminus n$ restricts the name n to P. $P_1 \parallel P_2$ places its subterms in parallel. The process $y!z \to P$ first sends a value z over y and then behaves as P. By convention, the prefix operator $\to$ is right associative. $[\, C \,]$ describes a choice whose branches are separated by $\Box$. In $\mathrm{CSP}_{\mathrm{in}}$ all branches of a choice are either guarded by an input prefix $y?\,(x)$ or the internal action τ. Similar to [26], we define replication as non-deterministic repetition $\star[\, \mathrm{guard}_1 \to P_1 \;\Box\; \ldots \;\Box\; \mathrm{guard}_n \to P_n \,]$.

The capabilities and guards are similar to the pi-calculus. Also the definitions of free and bound names are standard and similar to the pi-calculus. Again, we sometimes omit an action's object when it does not effectively contribute to the behaviour of a term, e.g. as in $y?\,(x) \to \mathsf{0}$, which would be written as $y? \to \mathsf{0}$.

The second variant of CSP that we consider is a subcalculus of $\mathrm{CSP}_{\mathrm{in}}$ that allows only for internal choice.

Definition 14 ($\mathrm{CSP}_{\mathrm{no}}$)**.** *The set of process terms of the* CSP-calculus with only internal choice*, denoted by* $\mathcal{P}_{\mathrm{no}}$*, is given by*

$$\begin{aligned} P &::= \mathsf{0} \;\mid\; P \setminus n \;\mid\; P_1 \parallel P_2 \;\mid\; \checkmark \;\mid\; y!z \to P \;\mid\; y?\,(x) \to P \\ &\quad \mid\; [\, C \,] \;\mid\; \star[\, C \,] \\ C &::= G \;\mid\; G \;\Box\; C \quad \textit{and} \quad G ::= \tau \to P \end{aligned}$$

for some names $n, x, y, z \in \mathcal{N}$

Note that we define the variants $\mathrm{CSP}_{\mathrm{in}}$ and $\mathrm{CSP}_{\mathrm{no}}$ of the CSP-calculus such that they are comparable to the variants CSP_{in} and CSP_{no} used in [26], because we use these variants to restate the separation result of [26] between CSP_{in} and CSP_{no}.

The operational semantics and structural congruence of $\mathrm{CSP}_{\mathrm{in}}$ and $\mathrm{CSP}_{\mathrm{no}}$ can be derived from [29]. In contrast to communications in the pi-calculus, where communication is always between exactly one input and one output guarded process, communication steps in CSP reduce a single output guarded process and arbitrarily many input guarded terms. Moreover, to perform a communication step, all toplevel parallel components have to participate in this communication. Interestingly, this communication mechanism in CSP leads to a separation result in Section D.2, while [16] presents a good and distributability-preserving encoding between the respective counterparts π_{s} without output guarded sums and π_{a} in the pi-calculus.

A.4 Labelled Processes

We assume for each process calculus a so-called *labelling* on the capabilities of processes. The labelling has to ensure that (1) each capability has a label (2) no label occurs more than once in a labelled term, (3) a label disappears only when the corresponding capability is reduced in a reduction step, and (4), once it has disappeared, it will not appear in the execution any more. [4] defines a labelling method to establish such a labelling for processes of the pi-calculus. They derive the labels from the syntax tree of processes. More precisely, they define the labels

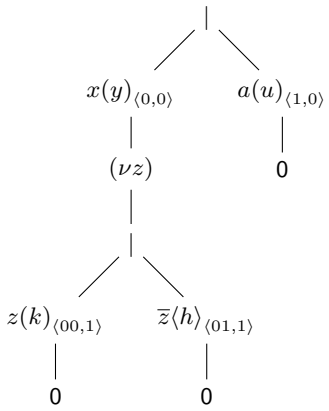

Fig. 4. Tree-representation of a labelled term.

as tupels $\langle s, n\rangle$, where s is a string over $\{0, 1\}$ representing the position of the capability within the parallel structure of the term and n is a natural number that represents the number of guards under which the capability is captured in the term. Labels are assigned to a term P by the function $L_{\langle 0,0\rangle}(P)$, where

$$\begin{array}{ll} L_{\langle s,n\rangle}(0) & = 0 \\ L_{\langle s,n\rangle}(\pi.P) & = \pi_{\langle s,n\rangle}.L_{\langle s,n+1\rangle}(P) \\ L_{\langle s,n\rangle}(P_1 \mid P_2) & = L_{\langle s0,n\rangle}(P_1) \mid L_{\langle s1,n\rangle}(P_2) \\ L_{\langle s,n\rangle}((\nu x)\, P) & = (\nu x)\, L_{\langle s,n\rangle}(P) \\ L_{\langle s,n\rangle}(!P) & = !_{\langle s,n\rangle}P \end{array}$$

As example consider the labelled version

$$x(y)_{\langle 0,0\rangle} \cdot \left((\nu z)\left(z(k)_{\langle 00,1\rangle} \mid \overline{z}\langle h\rangle_{\langle 01,1\rangle}\right)\right) \mid a(u)_{\langle 1,0\rangle}$$

of the term $x(y) \cdot ((\nu z)(z(k) \mid \overline{z}\langle h\rangle)) \mid a(u)$ as visualised in Figure 4.

[4] do not consider choice in the considered variant of the pi-calculus. To capture that operator, we have replace $L_{\langle s,n\rangle}(\pi.P) = \pi_{\langle s,n\rangle}.L_{\langle s,n+1\rangle}(P)$ by

$$L_{\langle s,n\rangle}\left(\sum_{i\in I}\pi_i.P_i\right) = \left(\sum_{i\in I}\pi_i.L_{\langle s,n+1\rangle}(P_i)\right)_{\langle s,n\rangle}$$

Note that there is only a single label for each sum, because we consider a sum as a single capability here. Moreover, [4] do not consider structural congruence and define a labelled semantics which contains the rule

$$\frac{P \xrightarrow{\mu} P'}{!P \xrightarrow{\mu} P' \mid !P}$$

To ensure the above properties on the labelling within sequences of labelled steps, they replace this rule by

$$\frac{P \xrightarrow{\mu} P'}{!_{\langle s,n\rangle}P \xrightarrow{\mu} L_{\langle s0,n+1\rangle}(P') \mid !_{\langle s1,n+1\rangle}P},$$

which ensures fresh labels for the reduced copy of P. To adapt this method to our formalism we have to replace the structural congruence rule $!P \equiv P \mid !P$ by

$$!_{\langle s,n\rangle}P \equiv L_{\langle s0,n+1\rangle}(P') \mid !_{\langle s1,n+1\rangle}P$$

The remaining structural congruence rules do not need special attention, they are simply changed to operate on labelled instead of unlabelled terms such that labels are preserved. Note that, because of the rules for commutativity and associativity of the parallel operator, applications of structural congruence destroy the additional information on the structure on the term that is provided by the labels. However, in contrast to [4] we make no use of these additional informations but use the labelling only to distinguish between different but syntactical equivalent capabilities.

A labelling for the introduced variants of the join-calculus or the CSP-calculus can be obtained in a similar way.

B Encodings and Quality Criteria

As defined above an *encoding* from $\mathcal{L}_{\mathrm{S}}$ into $\mathcal{L}_{\mathrm{T}}$ is a function $[\![\cdot]\!] : \mathcal{P}_{\mathrm{S}} \to \mathcal{P}_{\mathrm{T}}$. We often use $S, S', S_1, \ldots$ to range over $\mathcal{P}_{\mathrm{S}}$ and $T, T', T_1, \ldots$ to range over $\mathcal{P}_{\mathrm{T}}$. We shortly present the five quality criteria of the framework of [8] for language comparison.

The five conditions are divided into two structural and three semantic criteria. The structural criteria include (1) *compositionality* and (2) *name invariance*. The semantic criteria include (3) *operational correspondence*, (4) *divergence reflection*, and (5) *success sensitiveness*. It turns out that we do not need the second criterion to derive the separation results of this paper. Thus, we omit it. Note that a behavioural equivalence $\asymp$ on the target language is assumed for the definition of name invariance and operational correspondence. Its purpose is to describe the abstract behaviour of a target process, where abstract refers to the behaviour of the source term.

Intuitively, an encoding is compositional if the translation of an operator is the same for all occurrences of that operator in a term. Hence, the translation of that operator can be captured by a context that is allowed in [8] to be parametrised on the free names of the respective source term.

Definition 15 (Criterion 1: Compositionality). *The encoding* $[\![\cdot]\!]$ *is* compositional *if, for every operator* $\mathbf{op} : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{S}}$ *of* $\mathcal{L}_{\mathrm{S}}$ *and for every subset of names* N*, there exists a context* $\mathcal{C}^N_{\mathbf{op}}([\cdot]_1, \ldots, [\cdot]_{n+m}) : \mathcal{N}^n \times \mathcal{P}_{\mathrm{S}}^m \to \mathcal{P}_{\mathrm{T}}$ *such that, for all* $x_1, \ldots, x_n \in \mathcal{N}$ *and all* $S_1, \ldots, S_m \in \mathcal{P}_{\mathrm{S}}$ *with* $\mathsf{fn}(S_1) \cup \ldots \cup \mathsf{fn}(S_m) = N$*, it holds that* $[\![\, \mathbf{op}(x_1, \ldots, x_n, S_1, \ldots, S_m) \,]\!] = \mathcal{C}^N_{\mathbf{op}}(x_1, \ldots, x_n, [\![\, S_1 \,]\!], \ldots, [\![\, S_m \,]\!])$.

The first semantic criterion is operational correspondence. It consists of a soundness and a completeness condition. *Completeness* requires that every computation of a source term can be emulated by its translation. *Soundness* requires that every computation of a target term corresponds to some computation of the corresponding source term.

Definition 16 (Criterion 3: Operational Correspondence). *The encoding* $[\![\cdot]\!]$ *satisfies* operational correspondence *if it satisfies:*

Completeness*:* *For all* $S \Longmapsto_{\mathrm{S}} S'$*, it holds* $[\![S]\!] \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]$.
Soundness*:* *For all* $[\![S]\!] \Longmapsto_{\mathrm{T}} T$*, there exists an* S' *such that* $S \Longmapsto_{\mathrm{S}} S'$ *and* $T \Longmapsto_{\mathrm{T}} \asymp [\![S']\!]$.

Note that the definition of operational correspondence relies on the equivalence $\asymp$ to get rid of junk possibly left over within computations of target terms. Sometimes, we refer to the completeness criterion of operational correspondence as *operational completeness* and, accordingly, for the soundness criterion as *operational soundness*. The next criterion concerns the role of infinite computations in encodings.

Definition 17 (Criterion 4: Divergence Reflection). *The encoding* $[\![\cdot]\!]$ reflects divergence *if, for every* S, $[\![S]\!] \longmapsto^{\omega}_{\mathrm{T}}$ *implies* $S \longmapsto^{\omega}_{\mathrm{S}}$.

The last criterion links the behaviour of source terms to the behaviour of their encodings. With Gorla [8], we assume a *success* operator $\checkmark$ as part of the syntax of both the source and the target language. Since $\checkmark$ cannot be further reduced and $\mathsf{n}(\checkmark) = \mathsf{fn}(\checkmark) = \mathsf{bn}(\checkmark) = \emptyset$, the semantics and structural congruence of a process calculus are not affected by this additional constant operator. The test for reachability of success is standard.

Definition 18 (Success). *A process* $P \in \mathcal{P}$ may lead to success*, denoted as* $P \Downarrow_{\checkmark}$*, if it is reducible to a process containing a top-level unguarded occurrence of* $\checkmark$*, i.e., if* $P \Longmapsto P' \wedge P' \equiv P'' \mid \checkmark$ *for some* P', P''*. Moreover, we write* $P \Downarrow_{\checkmark!}$*, if* P *reaches success in every finite maximal execution.*

Note that we choose may-testing here. However, this choice is not crucial. An encoding preserves the abstract behaviour of the source term if it and its encoding answer the tests for success in exactly the same way.

Definition 19 (Criterion 5: Success Sensitiveness). *The encoding* $[\![\cdot]\!]$ *is* success sensitive *if, for every* S*,* $S \Downarrow_{\checkmark}$ *iff* $[\![S]\!] \Downarrow_{\checkmark}$.

This criterion only links the behaviours of source terms and their literal translations, but not of their derivatives. To do so, Gorla relates success sensitiveness and operational correspondence by requiring that the equivalence on the target language never relates two processes with different success behaviours.

Definition 20 (Success Respecting). $\asymp$ *is* success respecting *if, for every* P *and* Q *with* $P \Downarrow_{\checkmark}$ *and* $Q \not\Downarrow_{\checkmark}$*, it holds that* $P \not\asymp Q$.

By [8] a "good" equivalence $\asymp$ is often defined in the form of a barbed equivalence (as described e.g. in [15]) or can be derived directly from the reduction semantics and is often a congruence, at least with respect to parallel composition. For the separation results presented in this paper, we require only that $\asymp$ is a success respecting reduction bisimulation.

Definition 21 ((Weak) Reduction Bisimulation). *The equivalence $\asymp$ is a* (weak) reduction bisimulation *if, for every $T_1, T_2 \in \mathcal{P}_{\mathrm{T}}$ such that $T_1 \asymp T_2$, for all $T_1 \Longrightarrow_{\mathrm{T}} T_1'$ there exists a T_2' such that $T_2 \Longrightarrow_{\mathrm{T}} T_2'$ and $T_1' \asymp T_2'$.*

In this case, a good encoding respects also the ability to reach success in all finite maximal executions.

Lemma 6. *For all success respecting reduction bisimulations $\asymp \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$ and all terms $T_1, T_2 \in \mathcal{P}_{\mathrm{T}}$ such that $T_1 \asymp T_2$, it holds $T_1 \Downarrow_{\surd !}$ iff $T_2 \Downarrow_{\surd !}$.*

Proof. Let us assume the opposite, i.e., there is some success respecting bisimulation $\asymp \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$ and two terms $T_1, T_2 \in \mathcal{P}_{\mathrm{T}}$ such that $T_1 \asymp T_2$ and $T_1 \Downarrow_{\surd !}$ but not $T_2 \Downarrow_{\surd !}$. Then, for all $T_1' \in \mathcal{P}_{\mathrm{T}}$ with $T_1 \Longrightarrow_{\mathrm{T}} T_1'$, we have $T_1' \Downarrow_{\surd}$ but there exists some $T_2' \in \mathcal{P}_{\mathrm{T}}$ such that $T_2 \Longrightarrow_{\mathrm{T}} T_2'$ and $T_2' \not\Downarrow_{\surd}$.

Since $\asymp$ is a bisimulation (Definition 21), $T_1 \asymp T_2$ and $T_2 \Longrightarrow_{\mathrm{T}} T_2'$ imply that there exists some $T_1'' \in \mathcal{P}_{\mathrm{T}}$ such that $T_1 \Longrightarrow_{\mathrm{T}} T_1''$ and $T_2' \asymp T_1''$. Because $\asymp$ is success respecting (Definition 20), $T_2' \asymp T_1''$ and $T_2' \not\Downarrow_{\surd}$ imply $T_1'' \not\Downarrow_{\surd}$. This violates the requirement that $T_1 \Downarrow_{\surd !}$, i.e., contradicts the assumption that for all $T_1' \in \mathcal{P}_{\mathrm{T}}$ with $T_1 \Longrightarrow_{\mathrm{T}} T_1'$ we have $T_1' \Downarrow_{\surd}$. We conclude that $T_1 \Downarrow_{\surd !}$ iff $T_2 \Downarrow_{\surd !}$. □

Moreover, in this case success sensitiveness preserves also the ability to reach success in all finite maximal executions.

Lemma 7. *For all operationally sound and success sensitive encodings $[\![\cdot]\!]$ with respect to some success respecting equivalence $\asymp \subseteq \mathcal{P}_{\mathrm{T}} \times \mathcal{P}_{\mathrm{T}}$ and for all $S \in \mathcal{P}_{\mathrm{S}}$, if $S \Downarrow_{\surd !}$ then $[\![S]\!] \Downarrow_{\surd !}$.*

Proof. Assume the opposite, i.e., there is an encoding that satisfies the criteria operational soundness and success sensitiveness, $\asymp$ is success respecting, and there is some $S \in \mathcal{P}_{\mathrm{S}}$ such that for all $S' \in \mathcal{P}_{\mathrm{S}}$ with $S \Longrightarrow_{\mathrm{S}} S'$ we have $S' \Downarrow_{\surd}$, i.e., $S \Downarrow_{\surd !}$, but there is some $T \in \mathcal{P}_{\mathrm{T}}$ such that $[\![S]\!] \Longrightarrow_{\mathrm{T}} T$ and $T \not\Downarrow_{\surd}$.

Since $[\![\cdot]\!]$ is operationally sound (Definition 16), $[\![S]\!] \Longrightarrow_{\mathrm{T}} T$ implies that there exists some $S'' \in \mathcal{P}_{\mathrm{S}}$ and some $T' \in \mathcal{P}_{\mathrm{T}}$ such that $S \Longrightarrow_{\mathrm{S}} S''$ and $T \Longrightarrow_{\mathrm{T}} T' \asymp [\![S'']\!]$. By Definition 18, then $T \not\Downarrow_{\surd}$ and $T \Longrightarrow_{\mathrm{T}} T'$ imply $T' \not\Downarrow_{\surd}$. Since $\asymp$ respects success (Definition 20), $T' \asymp [\![S'']\!]$ and $T' \not\Downarrow_{\surd}$ imply $[\![S'']\!] \not\Downarrow_{\surd}$. Because $[\![\cdot]\!]$ is success sensitive (Definition 19), then also $S'' \not\Downarrow_{\surd}$, which contradicts the assumption that $S \Downarrow_{\surd !}$. We conclude that if $S \Downarrow_{\surd !}$ then $[\![S]\!] \Downarrow_{\surd !}$. □

C Distributability

Above we state that two executions of a term P are distributable iff P is distributable into two subterms such that each performs one of these executions. Here, we prove this relationship between Definition 3 and Definition 6.

Lemma 8. *Let $\mathcal{L} = \langle\, \mathcal{P}, \longmapsto \,\rangle$ be a process calculus, $P \in \mathcal{P}$, and $A_1, \ldots, A_n$ a set of executions of P. The executions $A_1, \ldots, A_n$ are pairwise distributable within P iff P is distributable into $P_1, \ldots, P_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of P_i, i.e., during A_i only capabilities of P_i are reduced.*

Proof. Let $\equiv$ be the structural congruence of $\mathcal{L}$.

Assume that the set of executions $A_1, \ldots, A_n$ are pairwise distributable in P. By Definition 6 no pair of executions A_i and A_j with $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$ reduces the same not distributable capability. Moreover, since for all $1 \leq i \leq n$ the sequence of steps A_i is an execution of P, i.e., $P \longmapsto P_{i,1} \longmapsto \ldots \longmapsto P_{i,m}$ for some $P_{i,1}, \ldots, P_{i,m} \in \mathcal{P}$, none of these executions reduces a capability produced, i.e., unguarded, by a step in one of the other executions in the set $\{\, A_1, \ldots, A_n \,\}$. Thus, whenever an execution A_i reduces some capability that was guarded in P, then A_i also reduces the guarding capability. Hence, we can choose $P_1, \ldots, P_n$ such that, for all $1 \leq i \leq n$, P_i is an unguarded subterm of P' or can be separated in P' by the chemical approach with $P' \equiv P$ and P_i contains at least all capabilities reduced in A_i. Note that to ensure that all guarded subterms and constants of P are contained in at least one of the terms $P_1, \ldots, P_n$, as it is required by the last condition of Definition 3, some of these terms may contain subterms that are not reduced by one of the executions $A_1, \ldots, A_n$. Since different executions A_i and A_j with $1 \leq i \leq n$, $1 \leq j \leq n$, and $i \neq j$ reduce the same capability only if it is recurrent and distributable, by Definition 3, the terms $P_1, \ldots, P_n$ are distributable in P.

Now, assume that P is distributable into n terms $P_1, \ldots, P_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of P_i, i.e., during A_i only capabilities of P_i are reduced. Then, by Definition 3, no capability with the same label occurs twice in $P_1, \ldots, P_n$. Hence, since A_i reduces only capabilities in P_i, no two executions in $A_1, \ldots, A_n$ reduces the same capability. Thus, by Definition 6, then all executions in $\{\, A_1, \ldots, A_n \,\}$ are pairwise distributable in P. □

Moreover, we prove Lemma 1.

Proof (Proof of Lemma 1). Let $\mathcal{L}_\mathrm{S} = \langle\, \mathcal{P}_\mathrm{S}, \longmapsto_\mathrm{S} \,\rangle$ and let $\mathcal{L}_\mathrm{T} = \langle\, \mathcal{P}_\mathrm{T}, \longmapsto_\mathrm{T} \,\rangle$ be two process calculi.

Let us assume that the set of executions $A_1, \ldots, A_n$ is pairwise distributable within S. Then, by Lemma 8, S is distributable into n terms $S_1, \ldots, S_n \in \mathcal{P}$ such that, for all $1 \leq i \leq n$, A_i is an execution of S_i, i.e., during A_i only capabilities of S_i are reduced. Because $[\![\cdot]\!]$ preserves distributability, by Definition 4, there are some $T_1, \ldots, T_n \in \mathcal{P}_\mathrm{T}$ that are distributable within $[\![S]\!]$ such that $T_i \asymp [\![S_i]\!]$ for all $1 \leq i \leq n$. Let us fix some arbitrary $i \in \{\, 1, \ldots, n \,\}$. By operational completeness in Definition 16, all sequences of steps of S_i are emulated by its encoding, i.e., $S_i \Longmapsto_\mathrm{S} S_i'$ implies $[\![S_i]\!] \Longmapsto_\mathrm{S} \asymp [\![S_i']\!]$. Because $\asymp$ is some reduction bisimulation, $T_i \asymp [\![S_i]\!]$ implies that also T_i has to emulate the executions of S_i independent of the other encoded subterms, i.e., $[\![S_i]\!] \Longmapsto_\mathrm{S} \asymp [\![S_i']\!]$ implies $T_i \Longmapsto_\mathrm{S} \asymp [\![S_i']\!]$. We conclude that for all $1 \leq i \leq n$ the term T_i emulates the sequence of steps A_i. Then, again by Lemma 8, all these emulations are pairwise distributable within $[\![S]\!]$. □

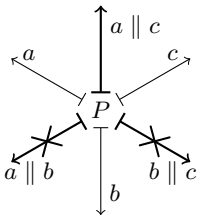

Fig. 5. Visualisation of the Synchronisation Pattern M.

D Synchronisation Pattern

Within this section we present the proof of our main results.

D.1 The Synchronisation Pattern M in π_a and J

Proof (Proof of Lemma 2). Two steps are in conflict, if performing one step disables the other step. To do so the first step has to consume something necessary to perform the other step. In the join-calculus, it is not possible to consume input capabilities, i.e., definitions. Hence, the only way for a step to disable a former alternative step is to consume one of its necessary outputs. In the join-calculus, communication is allowed only on defined variables, i.e., to consume an output message the channel of that message has to be defined in a definition. Note that defining the syntactical representation of a name twice in different definitions results in two different names. Thus, if the first step consumes an output necessary to perform the second step, then both steps share a defined name. Because of that, both steps must use the same definition, i.e., are not distributable. We conclude that for each list of alternative steps $S = [s_1, \ldots, s_n]$, where for all $1 \leq i < n$ the step s_i is in conflict with the step s_{i+1}, all steps in S use exactly the same definition. Thus, all pairs of steps in the set $S = \{ s_1, \ldots, s_n \}$ are pairwise local. □

Figure 5 visualises the synchronisation pattern M as conditions on a state P in a step transition system. Note that a, b, and c are not labels. They serve just to distinguish different steps. Moreover, $x \parallel y$ refer to the parallel execution of x and y, given a step semantics. The following example visualises a local M in the join-calculus.

Example 2 (Local M *in the join-calculus).* Consider the J-term

$$P = \mathsf{def}\, x\,(z) \mid y\,(z') \triangleright z\,\langle z' \rangle \;\mathsf{in}\; (x\,\langle u \rangle \mid x\,\langle v \rangle \mid y\,\langle u \rangle \mid y\,\langle v \rangle)\,.$$

To show that P is an M, we can for example choose:

- $a : P \longmapsto u\,\langle u \rangle \mid \mathsf{def}\, x\,(z) \mid y\,(z') \triangleright z\,\langle z' \rangle \;\mathsf{in}\; (x\,\langle v \rangle \mid y\,\langle v \rangle)$,
- $b : P \longmapsto u\,\langle v \rangle \mid \mathsf{def}\, x\,(z) \mid y\,(z') \triangleright z\,\langle z' \rangle \;\mathsf{in}\; (x\,\langle v \rangle \mid y\,\langle u \rangle)$, and

– $c : P \longmapsto v\,\langle v\rangle \mid \mathsf{def}\; x\,(z) \mid y\,(z') \triangleright z\,\langle z'\rangle \;\mathsf{in}\; (x\,\langle u\rangle \mid y\,\langle u\rangle)$.

We observe that $u\,\langle u\rangle$, $u\,\langle v\rangle$, and $v\,\langle v\rangle$ are pairwise different. Moreover, the steps a and c are parallel, but b disables a as well as c, because it consumes $x\,\langle u\rangle$ necessary for a and $y\,\langle v\rangle$ necessary for c. Since P does only contain a single definition, all its steps are local. Hence, P is a local M in the join-calculus. And, since P is in fact a cJ-term, it is also a local M in the core join-calculus.

We show that all M in the join-calculus are local.

Proof (Proof of Lemma 3). Assume the opposite, i.e., assume there is a non-local M in the join-calculus. Let us denote the corresponding J-term as P. By Definition 7, P can perform three alternative steps a, b, and c such that a and c are distributable but b is in conflict with both a and c. By Lemma 2, all conflicts in the join-calculus are local. Thus, all three steps a, b, and c are pairwise local, which contradicts the assumption that a and c are distributable. □

As mentioned in Section 4.2, we use the following example as counterexample to show that no good encoding from π_{a} into J can preserve distributability.

Example 3 (Running Counterexample). The non-local M

$$S = (\overline{y}\langle u\rangle \mid y(x)\,.\overline{x}) \mid (\overline{y}\langle v\rangle \mid y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark) \tag{E1}$$

reaches success iff S performs both of the distributable steps a and c, where

Step a**:** $S \longmapsto S_a$ with $S_a = \overline{u} \mid \overline{y}\langle v\rangle \mid y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$ and $S_a \Downarrow_{\checkmark!}$,
Step b**:** $S \longmapsto S_b$ with $S_b = y(x)\,.\overline{x} \mid \overline{y}\langle v\rangle \mid \overline{u} \mid u.v.v.\checkmark$ and $S_b \not\Downarrow_{\checkmark}$, and
Step c**:** $S \longmapsto S_c$ with $S_c = \overline{y}\langle u\rangle \mid y(x)\,.\overline{x} \mid \overline{v} \mid \overline{v} \mid u.v.v.\checkmark$ and $S_c \Downarrow_{\checkmark!}$.

To show that no good and distributability-preserving encoding can emulate E1, we use the fact that two distributable reductions in the join-calculus cannot reduce the same defined variable.

Lemma 9. *Let $P \in \mathcal{P}_{\mathrm{J}}$ and let A and C denote two distributable executions of P. Then the set of defined variables of all outputs reduced in A and all outputs reduced in C are disjoint.*

Proof. Without loss of generality let us assume that there are no name clashes in P. Let D_A denote the set of defined variables of all outputs reduced in A, and let D_C denote the corresponding set for C. Let us assume A and C are distributable but $D_A \cap D_C \neq \emptyset$. Then there is some defined name y such that an output on channel y is reduced in one step s_a of A, and an output on channel y is reduced in one step s_c of C. Since for each defined name there is exactly one definition in the join-calculus, there is exactly one definition defining y. Because each step that reduces an output on channel y has in the join-calculus to use this definition, by Definition 5, s_a and s_c are not distributable. Hence, by Definition 6, A and C are not distributable, which contradicts our assumption. □

Hence the encoding has to split up the conflict in the counterexample.

Proof (Proof of Lemma 4). By operational completeness (Definition 16), all three steps of S have to be emulated in $[\![S]\!]$, i.e., there exists $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ such that $[\![S]\!] \Longmapsto T_a \asymp [\![S_a]\!]$, $[\![S]\!] \Longmapsto T_b \asymp [\![S_b]\!]$, and $[\![S]\!] \Longmapsto T_c \asymp [\![S_c]\!]$. Because S has no infinite execution and $[\![\cdot]\!]$ reflects divergence (Definition 17), $[\![S]\!]$ has no infinite execution. By success sensitiveness (Definition 19), Lemma 6 and 7, and because $\asymp$ is success respecting (Definition 20), we have $T_a \Downarrow_{\checkmark !}$, $T_b \not\Downarrow_{\checkmark}$, $T_c \Downarrow_{\checkmark !}$, and $T_a \not\asymp T_b \not\asymp T_c$. We conclude that, for all $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ such that $T_a \asymp [\![S_a]\!]$, $T_b \asymp [\![S_b]\!]$, and $T_c \asymp [\![S_c]\!]$ and for all sequences $A : [\![S]\!] \Longmapsto T_a$, $B : [\![S]\!] \Longmapsto T_b$, and $C : [\![S]\!] \Longmapsto T_c$, there is a conflict between a step of A and a step of B, and there is a conflict between a step of B and a step of C. Note, that since $T_b \not\Downarrow_{\checkmark}$ but $T_a \Downarrow_{\checkmark !}$ and $T_c \Downarrow_{\checkmark !}$, the conflict between a and b (or b and c) has to be translated into a conflict of A and B (or B and C). It is not possible, that the emulation of b disables all ways to reach success after T_a or T_c is reached.

Because $[\![\cdot]\!]$ preserves distributability (Definition 4) and because of Lemma 1, the distributable steps a and c of S have to be translated into distributable executions, i.e., there is at least one A and one C such that these two executions are distributable. By Lemma 8, this implies that $[\![S]\!]$ is distributable into $T_1, T_2 \in \mathcal{P}_{\mathrm{J}}$ such that A is an execution of T_1 and C is an execution of T_2. By Lemma 2, the conflicts between A, B, and C are such that B and A as well as B and C compete for some output but, by Lemma 9, A and C do not reduce the same outputs. Hence, the two conflicts cannot be ruled out in a single step. Moreover, the reduction steps of A that lead to the conflicting step with B and the reduction steps of C that lead to the conflicting step with B are distributable, because A and C are distributable. We conclude, that there is at least one emulation of b, i.e., one execution $B : [\![S]\!] \Longmapsto T_b \asymp [\![S_b]\!]$, starting with two distributable executions such that one is (in its last step) in conflict with the emulation of a in $A : [\![S]\!] \Longmapsto T_a \asymp [\![S_a]\!]$ and the other one is in conflict with the emulation of c in $C : [\![S]\!] \Longmapsto T_c \asymp [\![S_c]\!]$. In particular this means that also the two steps of B that are in conflict with a step in A and a step in C are distributable.

Hence, there is no possibility to ensure that these two conflicts are decided consistently, i.e., there is a maximal execution of $[\![S]\!]$ that emulates A but not C as well as a maximal execution of $[\![S]\!]$ that emulates C but not A. □

Finally, we show that there cannot exist a good and distributability-preserving encoding.

Proof (Proof of Theorem 1). Assume the opposite. Then there is a good and distributability-preserving encoding of the S given by E1. By the proof of Lemma 4, there is a maximal execution of $[\![S]\!]$ in that a but not c is emulated and success is reached, i.e., there is an execution such that the emulation of a leads to success without the emulation of c.

For encodings as described above, there exists a context $\mathcal{C} : \mathcal{P}_{\mathrm{J}}^2 \to \mathcal{P}_{\mathrm{J}}$—the combination of the surrounding context and the context introduced by compositionality (Definition 15)—such that $[\![S]\!] = \mathcal{C}([\![S_1]\!], [\![S_2]\!])$, where $S_1 = \overline{y}\langle u \rangle \mid$

$y(x)\,.\overline{x}$ and $S_2 = \overline{y}\langle v\rangle \mid y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$. Let $S_2' = y(x)\,.\,(\overline{x} \mid \overline{x}) \mid u.v.v.\checkmark$. Since $\mathsf{fn}(S_2) = \mathsf{fn}(S_2')$, also $S_1 \mid S_2'$ has to be translated by the same context, i.e., $[\![\, S_1 \mid S_2' \,]\!] = \mathcal{C}\,([\![\, S_1 \,]\!], [\![\, S_2' \,]\!])$. Note that S and $S_1 \mid S_2'$ differ only by a capability necessary for step c, but step a and b are still possible. We conclude, that if $\mathcal{C}\,([\![\, S_1 \,]\!], [\![\, S_2 \,]\!])$ reaches some $T_a \Downarrow_{\checkmark!}$ without the emulation of c, then $\mathcal{C}\,([\![\, S_1 \,]\!], [\![\, S_2' \,]\!])$ reaches at least some state T_a' such that $T_a' \Downarrow_{\checkmark}$. Hence, $[\![\, S_1 \mid S_2' \,]\!] \Downarrow_{\checkmark}$ but $(S_1 \mid S_2') \not\Downarrow_{\checkmark}$ which contradicts success sensitiveness. □

D.2 Distributability of CSP

In the following, we show how the proof method behind the above separation result can be transferred to other process calculi. Accordingly, we consider two variants of CSP introduced in Section A.3. First we replace the source language π_{a} by $\mathrm{CSP}_{\mathrm{in}}$—a variant of CSP with input and output guards and input guarded choice. Afterwards we replace the target language J by $\mathrm{CSP}_{\mathrm{no}}$—a subcalculus of $\mathrm{CSP}_{\mathrm{in}}$, where choice is only internally branching. Note that these two languages were already compared in [26]. Here, we use them rather to explain how the separation result above is transferred than to prove new results. For simplicity, we consider only compositional encodings in the following, but the results hold as well for combinations of an inner compositional encoding surrounded by a fixed context parametrised on the free names of the source terms as considered by Theorem 1.

Changing the source language is often the easier task, because it usually suffices to show that the new source language is expressive enough to provide the counterexample with the properties required by the absolute result. In the present case, we have to show that $\mathrm{CSP}_{\mathrm{in}}$ contains an M similar to E1 and to recycle the argumentation in the proof of Theorem 1, thereby adapting it to the new source language. We gain the absolute result and Lemma 4 for free, because its proofs do not use any information about the source language except that it provides E1.

Example 4 (Non-Local M *in* $\mathrm{CSP}_{\mathrm{in}}$*).* Consider

$$S = S_1 \parallel (S_2 \parallel S_3) \tag{E2}$$

with $S, S_1, S_2, S_3 \in \mathcal{P}_{\mathrm{in}}$, where $S_1 = [\,(\tau \to \mathsf{0}) \;\square\; (b? \to \mathsf{0})\,]$, $S_2 = b! \to \mathsf{0}$, and $S_3 = [\,(b? \to \mathsf{0}) \;\square\; (\tau \to \checkmark)\,]$. S can perform three different alternative steps modulo structural congruence:

Step a**:** $S \longmapsto S_a$ with $S_a = \mathsf{0} \parallel (S_2 \parallel S_3)$
Step b**:** $S \longmapsto S_b$ with $S_b = \mathsf{0} \parallel (\mathsf{0} \parallel \mathsf{0})$
Step c**:** $S \longmapsto S_c$ with $S_c = S_1 \parallel (S_2 \parallel \checkmark)$

If the first step is either a or c then S can perform the respective other step as second step. Moreover, the steps a and c are parallel and distributable but b is in conflict with a and c. In case b is not performed, any maximal execution of S has two steps and leads to success. Hence, $S_a \Downarrow_{\checkmark!}$, $S_b \not\Downarrow_{\checkmark}$, and $S_c \Downarrow_{\checkmark!}$.

Since E2 and E1 have the same properties, we can show a separation result between CSP and J similar to Theorem 1.

Theorem 3. *There exists no good and distributability-preserving encoding from* $\mathrm{CSP}_{\mathrm{in}}$ *into* J.

Proof. Assume the opposite. Because S of Example 4 and E1 have the same properties, Lemma 4 holds also for S. Thus, there is a good and distributability-preserving encoding of S and there is a maximal execution of $[\![\ S\ ]\!]$ in that a but not c is emulated and success is reached, i.e., there is an execution such that the emulation of a leads to success without the emulation of c.

Let $S_3' = [\ (b? \to \mathsf{0})\ \ \square\ \ (\tau \to \mathsf{0})\ ]$. Because of compositionality (cf. Definition 15) and since $\mathsf{fn}(S_3) = \mathsf{fn}(S_3')$, the terms $[\![\ S\ ]\!]$ and $[\![\ S_1 \parallel (S_2 \parallel S_3')\ ]\!]$ differ only by the encoding of S_3. Note that $S \Downarrow_{\checkmark}$ and $(S_1 \parallel (S_2 \parallel S_3')) \not\Downarrow_{\checkmark}$, but the possibilities to perform the steps a, b, and c remain unchanged. We conclude, that if $[\![\ S\ ]\!]$ reaches some $T_a \Downarrow_{\checkmark !}$ without the emulation of c, then $[\![\ S_1 \parallel (S_2 \parallel S_3')\ ]\!]$ reaches as least some state T_a' such that $T_a' \Downarrow_{\checkmark}$. Hence, $[\![\ S_1 \parallel (S_2 \parallel S_3')\ ]\!] \Downarrow_{\checkmark}$ but $(S_1 \parallel (S_2 \parallel S_3')) \not\Downarrow_{\checkmark}$ which contradicts success sensitiveness. □

In case of the target language, we have to adapt the proof of Lemma 4. Therefore, we have first to revise the absolute result. To do so, we show that, because of the restrictive communication mechanism, without guards in choices all conflicts in $\mathrm{CSP}_{\mathrm{no}}$ are between τ-steps of a single choice only. Since choice is not distributable, all conflicts are local.

Lemma 10. *All conflicts in* $\mathrm{CSP}_{\mathrm{no}}$ *are between* τ*-steps and are local.*

Proof. Two steps are in conflict, if performing one step disables the other step by the consumption of a capability necessary to perform the other step. In $\mathrm{CSP}_{\mathrm{no}}$ a reduction step is either a τ-step or it reduces all unguarded capabilities of some subject. The later case is possible only if all parallel processes have an unguarded prefix with that subject and if there are no two unguarded output prefixes on this subject. Since all unguarded capabilities of some subject are reduced, an alternative step is either again a τ-step or a step on another subject. Without guards in choice it is not possible to remove an output or input prefix of subject y in a step on subject x. Thus, the only chance for conflicts is between τ-steps. The only way a τ-step may consume something necessary for an alternative step is within a choice. Since in $\mathrm{CSP}_{\mathrm{no}}$ only internal choice is allowed, i.e., all branches of a choice are guarded by τ, all conflicts in $\mathrm{CSP}_{\mathrm{no}}$ are between two τ-steps reducing the same internal choice. Since choice is not distributable, such steps are always local. □

As a consequence, all M in $\mathrm{CSP}_{\mathrm{no}}$ are also local, because of the conflict between b and a or c. Following the line of argumentation in Section 4.2, we show next that each good encoding of an M has to split up the conflicts of b with the steps a and c. It turns out that to adapt the proof of Lemma 4 it suffices to replace the argument on the absolute result in Lemma 2 by our new absolute result above.

Lemma 11. *Any good and distributability-preserving encoding from* π_{a} *(or from* $\mathrm{CSP}_{\mathrm{in}}$*) into* $\mathrm{CSP}_{\mathrm{no}}$ *has to split up the conflicts in* S *given by E1 (or by E2) of* b *with* a *and* c *such that there exists a maximal execution in* $[\![S]\!]$ *in which* a *is emulated but not* c*, and vice versa.*

Proof. The proof of Lemma 11 is similar to the proof of Lemma 4 above. It suffice to replace the sentences

> By Lemma 2, the conflicts between A, B, and C are such that B and A as well as B and C compete for some output but, by Lemma 9, A and C do not reduce the same outputs. Hence, the two conflicts cannot be ruled out in a single step.

by

> By Lemma 10, the conflicts between A, B, and C are such that B and A as well as B and C compete for some τ-capabilities within the same choice. Because the choice operator is not distributable but A and C are, the two conflicts cannot be ruled out in a single step.

Moreover, in case of E2, replace $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{J}}$ by $T_a, T_b, T_c \in \mathcal{P}_{\mathrm{in}}$. □

In this case we gain the argumentation in the proof of Theorem 1 and Theorem 3 for free.

Theorem 4. *There exists no good and distributability-preserving encoding from* π_{a} *(or* $\mathrm{CSP}_{\mathrm{in}}$*) into* $\mathrm{CSP}_{\mathrm{no}}$*.*

Proof. In case of π_{a}, the proof of Theorem 4 is similar to the proof of Theorem 1. It suffice to replace Lemma 4 by Lemma 11.

Else if the source language is $\mathrm{CSP}_{\mathrm{in}}$, the proof is similar to the proof of Theorem 3. Again it suffice to replace Lemma 4 by Lemma 11. □

D.3 The Synchronisation Pattern ⋆

Above we stated that in π_{s} each step between two distributable subprocesses reduces only outputs in one subprocess and only inputs in the other. We now prove this statement.

Lemma 12. *For all* $P \in \mathcal{P}_{\mathrm{s}}$ *and for all* $P_1, P_2 \in \mathcal{P}_{\mathrm{s}}$ *that are distributable within* P*, a reduction step between* P_1 *and* P_2 *either reduces only output guards in* P_1 *and only input guards in* P_2*, or vice versa.*

Proof. By the reduction semantics of π_{s} in Figure 3, the derivation of each reduction step results from exactly one axiom, i.e., there are no branches in derivation trees of reduction steps in π_{s}. Moreover, a step between two distributable processes, i.e., a step that uses capabilities of two parallel composed processes, cannot result from $\textsc{Tau}_{\mathrm{m,s}}$. By the Axiom $\textsc{Com}_{\mathrm{m,s}}$ an output guard within a sum and an input guard of another sum are reduced, but no other output or

input guards are reduced outside the mentioned two sums. Remember that in π_{s} it is not allowed to place input and output guards within the same sum. Hence, if the step reduces an output guard in P_1 it has to reduce input guards in P_2 but can neither reduce also input guards in P_1 nor output guards in P_2. The same holds if we swap the roles of P_1 and P_2. □

Because of that there is no $\star$ in π_{s}.

Proof (Proof of Lemma 5). Assume the opposite, i.e., assume there is some $P \in \mathcal{P}_{\mathrm{s}}$ such that $a : P \longmapsto P_a$, $b : P \longmapsto P_b$, $c : P \longmapsto P_c$, $d : P \longmapsto P_d$, and $e : P \longmapsto P_e$ for some $P_a, P_b, P_c, P_d, P_e \in \mathcal{P}_{\mathrm{s}}$ that are pairwise different such that a is in conflict with b, b is in conflict with c, c is in conflict with d, d is in conflict with e, e is in conflict with a, and all pairs of steps in $\{ a, b, c, d, e \}$ that are not in conflict are parallel in P. Note that a communication step in π_{s} always reduces a sum of output guarded subterms and a sum of input guarded subterms. Accordingly, let i_x be the capability of the sum of input guards reduced by step $x \in \{ a, b, c, d, e \}$ in P and o_x be the capability of the sum of output guards reduced by step x, respectively.

Since a and c are distributable in P, by Lemma 8, P is distributable into the terms $P_1, P_2 \in \mathcal{P}_{\mathrm{s}}$ such that a is a step of P_1 and c is a step of P_2, i.e., there exists $P_1', P_2' \in \mathcal{P}_{\mathrm{s}}$ and a sequence of names $\tilde{x}$ such that $P \equiv (\nu\tilde{x})\,(P_1 \mid P_2)$, $P_a \equiv (\nu\tilde{x})\,(P_1' \mid P_2)$, and $P_c \equiv (\nu\tilde{x})\,(P_1 \mid P_2')$. Because b is in conflict with a and c, it reduces one capability in P_1 and one capability in P_2, i.e., b is a communication between P_1 and P_2. By Lemma 12, b reduces either only input guards or only output guards in P_1. Let us assume that b reduces only output guards in P_1. Since b is in conflict with a, it reduces an unguarded output guard in $o_a = o_b$, i.e., a and b compete for outputs in the same sum. Then, again by Lemma 12, the conflict between b and c comes from a competition for the capability $i_b = i_c$ in P_2. b and d are distributable, but c is in conflict with b and d. We know that the conflict between b and c comes from the competition for the capability in $i_b = i_c$. By the same argumentation as before, then c and d compete for the capability in $o_c = o_d$. Furthermore, d and e compete for the capability in $i_d = i_e$. And thus, e and a compete for $o_e = o_a$. But then e and a as well as a and b compete for capabilities in o_a. Because e and b are distributable and sums are not distributable, they cannot reduce the same output guarded sum. This is a conflict. (By the way, even if it would have been possible to distribute the output guarded sum, we could apply Lemma 12 once more to derive the conflict as in the second case.)

Now, in order to capture the other case, let us assume that a and b compete for $i_a = i_b$. Thus, b and c compete for $o_b = o_c$, c and d compete for $i_c = i_d$, d and e compete for $o_d = o_e$, and e and a compete for $i_e = i_a$. By Lemma 12, if a is in conflict with e and b, it is not possible that a reduces an input guard of e as well as of b, i.e., again this is a conflict. □

The following example shows that π_{m}, in contrast to π_{s}, can express the synchronisation pattern $\star$. We use this example as running counterexample in the following.

Example 5 ($\star$ in π_{m}). Consider a term $S \in \mathcal{P}_{\mathrm{m}}$ such that

$$S = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5 \tag{E3}$$

for some $S_1, \ldots, S_5 \in \{\, \mathsf{0}, \checkmark \,\}$. Then, S can perform the steps

Step *a*: $S \longmapsto S_a$ with $S_a = \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid S_5$,
Step *b*: $S \longmapsto S_b$ with $S_b = S_1 \mid \overline{c} + d.S_3 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5$,
Step *c*: $S \longmapsto S_c$ with $S_c = \overline{a} + b.S_1 \mid S_2 \mid \overline{d} + e.S_4 \mid \overline{e} + a.S_5$,
Step *d*: $S \longmapsto S_d$ with $S_d = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid S_3 \mid \overline{e} + a.S_5$, and
Step *e*: $S \longmapsto S_e$ with $S_e = \overline{a} + b.S_1 \mid \overline{b} + c.S_2 \mid \overline{c} + d.S_3 \mid S_4$.

By Definition 8, S is a non-local $\star$.

Unfortunately, the same cyclic dependencies between the conflicts in $\star$ that are used in the proof of Lemma 5 prevent us from initialising $S_1, \ldots, S_5$ such that $S_x \Downarrow_{\checkmark !}$ and $S_y \not\Downarrow_{\checkmark}$ for each pair of conflicting steps x and y. Note that in the proof of Lemma 4 we use the properties $S_a \Downarrow_{\checkmark !}$, $S_b \not\Downarrow_{\checkmark}$, and $S_c \Downarrow_{\checkmark !}$ to ensure that the conflict of b with a and c has to be translated into a conflict of $B : [\![\, S \,]\!] \Longmapsto T_b \asymp [\![\, S_b \,]\!]$ with $A : [\![\, S \,]\!] \Longmapsto T_a \asymp [\![\, S_a \,]\!]$ and $C : [\![\, S \,]\!] \Longmapsto T_c \asymp [\![\, S_c \,]\!]$. Here, we use compositionality and the fact that initialising S_i for $1 \leq i \leq 5$ by either $\checkmark$ or $\mathsf{0}$ has no consequence on the surrounding contexts in the encoding, to show that the encoding has to preserve also the conflicts in $E3$.

Lemma 13. *Any good and distributability-preserving encoding $[\![\cdot]\!]$ from π_{m} into π_{s} has to translate the conflicts in S given by E3 into conflicts of the corresponding emulations.*

Proof. By operational completeness (Definition 16), all five steps of S have to be emulated in $[\![\, S \,]\!]$, i.e., there exists $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathrm{s}}$ such that $[\![\, S \,]\!] \Longmapsto T_x \asymp [\![\, S_x \,]\!]$ for all $x \in \{\, a, b, c, d, e \,\}$. Because $[\![\cdot]\!]$ preserves distributability, for each pair of steps x and y that are parallel in S, the emulations $X : [\![\, S \,]\!] \Longmapsto T_x$ and $Y : [\![\, S \,]\!] \Longmapsto T_y$ such that $T_x \asymp [\![\, S_x \,]\!]$ and $T_y \asymp [\![\, S_y \,]\!]$ are distributable. Note that X and Y refer to the upper case variants of x and y, respectively.

In Example 5 we do not initialise $S_1, \ldots, S_5$. Now, we consider all variants of S, where $S_1, \ldots, S_5 \in \{\, \mathsf{0}, \checkmark \,\}$, i.e., each of these terms is either chosen to be empty or to present an unguarded occurrence of success. Since $\mathsf{n}(\checkmark) = \mathsf{n}(\mathsf{0}) = \emptyset$ and because of compositionality (Definition 15), the encodings of these variants of S differ only by the encodings of $S_1, \ldots, S_5$. The remaining operators and, hence, the remaining term has to be translated in exactly the same way. Accordingly, the encoding of a term $S_1, \ldots, S_n$ cannot influence the emulation of the steps of S.

Thus, for each triple of steps $x, y, z \in \{\, a, b, c, d, e \,\}$ in S such that y is in conflict with x and z but x and z are parallel, we can choose $S_{\mathrm{f}(x)} = \checkmark = S_{\mathrm{f}(z)}$

and initialise all other terms in $\{ S_1, \ldots, S_5 \}$ by $\mathsf{0}$, where

$$\mathrm{f}(x) = \begin{cases} 1, & \text{if } x = b \\ 2, & \text{if } x = c \\ 3, & \text{if } x = d \\ 4, & \text{if } x = e \\ 5, & \text{if } x = a \end{cases}$$

for all $u \in \{ a, b, c, d, e \}$, such that $S_{\mathrm{f}(x)} \Downarrow_{\checkmark!}$ and $S_{\mathrm{f}(z)} \Downarrow_{\checkmark!}$ but $S_{\mathrm{f}(y)} \not\Downarrow_{\checkmark}$. Then, also $S_x \Downarrow_{\checkmark!}$ and $S_z \Downarrow_{\checkmark!}$ but $S_y \not\Downarrow_{\checkmark}$. Now we can proceed as in the proof of Lemma 4. Because S has no infinite execution and $[\![\cdot]\!]$ reflects divergence, $[\![S]\!]$ has no infinite execution. By success sensitiveness, Lemma 6 and 7, and because $\asymp$ is success respecting, we have $T_x \Downarrow_{\checkmark!}$, $T_y \not\Downarrow_{\checkmark}$, $T_z \Downarrow_{\checkmark!}$, and $T_x \not\asymp T_y \not\asymp T_z$. We conclude that, for all $T_x, T_y, T_z \in \mathcal{P}_{\mathrm{J}}$ such that $T_x \asymp [\![S_x]\!]$, $T_y \asymp [\![S_y]\!]$, and $T_z \asymp [\![S_z]\!]$ and for all sequences $X : [\![S]\!] \Longmapsto T_x$, $Y : [\![S]\!] \Longmapsto T_y$, and $Z : [\![S]\!] \Longmapsto T_z$, there is a conflict between a step of X and a step of Y, and there is a conflict between a step of Y and a step of Z. □

Similar to Section 4.2, we show that each good encoding of the counterexample requires that a conflict has to be distributed.

Lemma 14. *Any good and distributability-preserving encoding $[\![\cdot]\!]$ from π_{m} into π_{s} has to split up a least one of the conflicts in S given by E3 such that there exists a maximal execution in $[\![S]\!]$ that emulates only one source term step.*

Proof. By operational completeness (Definition 16), all five steps of S have to be emulated in $[\![S]\!]$, i.e., there exists $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathrm{s}}$ such that $X : [\![S]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ for all $x \in \{ a, b, c, d, e \}$, where X is the upper case variant of x. By Lemma 13, for all $T_a, T_b, T_c, T_d, T_e \in \mathcal{P}_{\mathrm{s}}$ and all $x \in \{ a, b, c, d, e \}$ such that $T_x \asymp [\![S_x]\!]$, there is a conflict between a step of the following pairs of emulations: A and B, B and C, C and D, D and E, and E and A.

Since $[\![\cdot]\!]$ preserves distributability (Definition 4) and by Lemma 4, each pair of distributable steps in S has to be translated into emulations that are distributable within $[\![S]\!]$. Let $X, Y, Z \in \{ A, B, C, D, E \}$ be such that X and Z are distributable within $[\![S]\!]$ but Y is in conflict with X as well as Z. By Lemma 8, this implies that $[\![S]\!]$ is distributable into $T_1, T_1 \in \mathcal{P}_{\mathrm{s}}$ such that X is an execution of T_1 and Z is an execution of T_2. Since Y is in conflict with X and Z and because all three emulations are executions of $[\![S]\!]$, there is one step of Y that is in conflict with one step of X and there is one (possibly the same) step of Y that is in conflict with one step of Z. Moreover, since X and Z are distributable, if a single step of Y is in conflict with X as well as Z then this step is a communication between T_1 and T_2.

Assume that for all such combinations X, Y, and Z, the conflicts between Y and X or Z are ruled out by a single step of Y, i.e., both conflicts are ruled out by a communication step between some capabilities of X and some capabilities of Z. By Lemma 12, then this step reduces only input guards in one of the executions X and Z and only output guards in the respective other, i.e., X and Y

compete either only for input or only for output guards and Y and Z compete for the respective other kind of guards. Without loss of generality let us assume that A and B compete for some output guards and, thus, B and C compete for some input guards, C and D compete for some output guards, D and E compete for some input guards, E and A compete for some output guards, and A and B compete for some input guards. This is a contradiction, because, by Lemma 12, A and B cannot compete for both input and output guards.

We conclude that there is at least one triple of emulations X, Y, and Z such that the conflict of Y with X and with Z results from two different steps in Y. Because X and Z are distributable, the reduction steps of X that leads to the conflicting step with Y and the reduction steps of Z that leads to the conflicting step with Y are distributable. We conclude, that there is at least one emulation of y, i.e., one execution $Y : [\![S]\!] \Longmapsto T_y \asymp [\![S_y]\!]$, starting with two distributable executions such that one is (in its last step) in conflict with the emulation of x in $X : [\![S]\!] \Longmapsto T_x \asymp [\![S_x]\!]$ and the other one is in conflict with the emulation of z in $Z : [\![S]\!] \Longmapsto T_z \asymp [\![S_z]\!]$. In particular this means that also the two steps of Y that are in conflict with a step in X and a step in Z are distributable. Hence, there is no possibility to ensure that these two conflicts are decided consistently, i.e., there is a maximal execution of $[\![S]\!]$ that emulates X but neither Y nor Z.

In the set $\{ A, B, C, D, E \}$ there are—apart from X, Y, and Z—two remaining executions. One of them, say X', is in conflict with X and the other one, say Z', is in conflict with Z. Since X is emulated successfully, X' cannot be emulated. Moreover, note that Y and Z' are distributable. Thus, also Z' and the partial execution of Y that leads to the conflict with Z are distributable. Moreover, also the step of Y that already rules out Z cannot be in conflict with a step of Z'. Thus, although the successful completion of Z is already ruled out by the conflict with Y, there is some step of Z left, that is in conflict with one step in Z'. Hence, the conflict between Z and Z' cannot be ruled out by the partial execution described so fare that leads to the emulation of X but forbids to complete the emulations of X', Y, and Z. Thus, it cannot be avoided that Z wins this conflict, i.e., that also Z' cannot be completed. We conclude that there is a maximal execution of $[\![S]\!]$ such that only one of the five source term steps of S is emulated. □

Since each maximal execution of E3 consists of exactly two distributable steps, Lemma 14 violates the requirements on a good encoding.

Proof (Proof of Theorem 2). Assume the opposite, i.e., there is a good and distributability-preserving encoding $[\![\cdot]\!]$ from π_{m} into π_{s}, and, thus, also of S given by E3. Since S has no infinite execution and because $[\![\cdot]\!]$ is divergence reflecting, $[\![S]\!]$ has no infinite execution. By Lemma 14 there exists a maximal execution in $[\![S]\!]$ in which only one source term step is emulated. Let us denote this step by $x \in \{ a, b, c, d, e \}$. Hence, $[\![S]\!] \Longmapsto T_x \Longmapsto T$ with $T \not\longmapsto$ for some $T \in \mathcal{P}_{\mathrm{s}}$, because there is no infinite execution. Moreover, by operational soundness, $T_x \asymp T$, because after the emulation of x no other step is emulated. Note that since we do not fix $S_1, \ldots, S_5$ in Example 5 and by the argumentation

in the Lemma 13 and 14, the above conditions hold for all variants of S such that $S_1, \ldots, S_5 \in \{\, \mathsf{0}, \checkmark \,\}$. Let us consider the case that $S_{\mathrm{f}(x)} = \mathsf{0} = S_{\mathrm{f}(y)}$ but all other terms in $\{\, S_1, \ldots, S_5 \,\}$ are equal to $\checkmark$, where f is the function defined in the proof of Lemma 13 and y is a step that is parallel to x within S. Then, $S_x \Downarrow_{\checkmark}$ but $S_x \not\Downarrow_{\checkmark!}$, i.e., the step x may lead to success (in case the next step is not y) or it does not lead to success (in case the next step is y). By success sensitiveness and because $\asymp$ is success sensitive, then also $T_x \Downarrow_{\checkmark}$ and $T \Downarrow_{\checkmark}$ but $T_x \not\Downarrow_{\checkmark!}$ and $T \not\Downarrow_{\checkmark!}$. But this contradicts the property that $T \mapsto\!\!\!\!\!\!\rightarrow$, because a term in π_{s} that cannot perform a step either already has an unguarded occurrence of success or can never reach some. We conclude that there cannot be such an encoding. □

References

26. L. Bougé. On the Existence of Symmetric Algorithms to Find Leaders in Networks of Communicating Sequential Processes. *Acta Informatica*, 25(4):179–201, 1988.
27. C. Fournet. *The Join-Calculus: a Calculus for Distributed Mobile Programming.* PhD thesis, L'École Polytechnique, 1998.
28. C. A. R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.
29. C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall International Series in Computer Science, 2004. Electronic version of Communicating Sequential Processes, first published in 1985.