Encoding CSP into CCS *

Meike Hatzel
TU Berlin

Christoph Wagner
TU Berlin

Kirstin Peters†
TU Dresden

Uwe Nestmann
TU Berlin

We study encodings from CSP into asynchronous CCS with name passing and matching, so in fact, the asynchronous π-calculus. By doing so, we discuss two different ways to map the multi-way synchronisation mechanism of CSP into the two-way synchronisation mechanism of CCS. Both encodings satisfy the criteria of Gorla except for compositionality, as both use an additional top-level context. Following the work of Parrow and Sjödin, the first encoding uses a centralised coordinator and establishes a variant of weak bisimilarity between source terms and their translations. The second encoding is decentralised, and thus more efficient, but ensures only a form of coupled similarity between source terms and their translations.

1 Introduction

In the context of a scientific meeting on Expressiveness in Concurrency and Structural Operational Semantics (SOS), likely very little needs to be said about the process algebras (or process calculi) CSP and CCS. Too many papers have been written since their advent in the 70's to be mentioned in our own paper; it is instructive, though, and recommended to appreciate Jos Baeten's historical overview [1], which also places CSP and CCS in the context of other process algebras like ACP and the many extensions by probabilities, time, mobility, etc. Here, we just select references that help to understand our motivation.

Differences. From the beginning, although CSP [8] and CCS [11] were intended to capture, describe and analyse reactive and interactive concurrent systems, they were designed following rather different philosophies. Tony Hoare described this nicely in his position paper [9] as follows: "A primary goal in the original design of CCS was to discover and codify a minimal set of basic primitive agents and operators ... and a wide range of useful operators which have been studied subsequently are all definable in terms of CCS primitives." and "CSP was more interested in this broader range of useful operators, independent of which of them might be selected as primitive." So, at their heart, the two calculi use two different synchronisation mechanisms, one (CCS) using binary, i.e., two-way, handshake via matching actions and co-actions, the other (CSP) using multiway synchronisation governed by explicit synchronisation sets that are typically attached to parallel composition. Another difference is the focus on Structural Operational Semantics in CCS, and the definition of behavioural equivalences on top of this, while CSP emphasised a trace-based denotational model, enhanced with failures, and the question on how to design models such that they satisfy a given set of laws of equivalence.

Comparisons. From the early days, researchers were interested in more or less formal comparisons between CSP and CCS. This was carried out by both Hoare [9] and Milner [12] themselves, where they concentrate on the differences in the underlying design principles. But also other researchers joined the game, but with different analysis tools and comparison criteria.

For example, Brookes [3] contributed a deep study on the relation between the underlying abstract models, synchronisation trees for CCS and the failures model of CSP. Quite differently, Lanese and Montanari [10] used the power to transform graphs as a measure for the expressiveness of the two calculi.

*Supported by the DFG via project "Synchronous and Asynchronous Interaction in Distributed Systems".
†Supported by the German Federal and State Governments via the Excellence Initiative (Institutional Strategy).

S. Crafa and D. Gebler (Eds.): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics (EXPRESS/SOS 2015)
EPTCS 190, 2015, pp. 61–75, doi:10.4204/EPTCS.190.5

© Hatzel, Wagner, Peters & Nestmann
This work is licensed under the Creative Commons Attribution License.

Yet completely differently, Parrow and Sjödin [16,21] tried to find an algorithm to implement—best in a fully distributed fashion—the multiway synchronisation operator of CSP (and its variant LOTOS [2]) using the supposedly simpler two-way synchronisation of CCS. They came up with two candidates—a reasonably simple centralised synchroniser, and a considerably less simple distributed synchroniser[1]—and proved that the two are not weakly bisimilar, but rather coupled similar, which is only slightly weaker. Coupled simulation is a notion that Parrow and Sjödin invented for just this purpose, but it has proved afterwards to be often just the right tool when analysing the correctness of distribution- and divergence-sensitive encodings that involve partial commitments (whose only effect is to gradually perform internal choices) [15].

The probably most recent comparison between CSP and CCS was provided by van Glabbeek [5]. As an example for his general framework to analyse the relative expressive power of calculi, he studied the existence of syntactical translations from CSP into CCS, for which a common semantical domain is provided via labelled transition systems (LTS) derived from respective sets of SOS rules. The comparison is here carried out by checking whether a CSP term and its translation into CCS are distinguishable with respect to a number of equivalences defined on top of the LTS. The concrete results are: (1) there is a translation that is correct up to trace equivalence (and contains deadlocks), and (2) there is no translation that is correct up to weak bisimilarity equivalence that also takes divergence into account.

Contribution. Given van Glabbeek's negative result, and given Parrow and Sjödin's algorithm, we set out to check whether we can define a syntactical encoding from CSP into CCS—using Parrow and Sjödin's ideas—that is correct up to coupled similarity.[2] We almost managed. In this paper, we report on our current results along these lines: (1) Our encoding target is an asynchronous variant of CCS, but enhanced with name-passing and matching, so it is in fact an asynchronous π-calculus; we kept mentioning CCS in the title of this paper, as it clearly emphasises the origin and motivation of this work. But, we could *not* do without name-passing. (2) We exhibit one encoding that is not distributability-preserving (so, it represents a centralised solution), but is correct up to weak bisimilarity and does not introduce divergence. This does not contradict van Glabbeek's results, but suggests that van Glabbeek's framework implies some form of distributability-preservation. (3) We exhibit another encoding that *is* distributability-preserving and divergence-reflecting, but is only correct up to coupled similarity.

Overview. We introduce the considered variants of CSP and CCS in § 2. There we also introduce the criteria—that are (variants of) the criteria in [6] and [20]—modulo which we prove the quality of the considered encodings. In § 3 we introduce the inner layer of our two encodings. It provides the main machinery to encode synchronisations of CSP. We complete this encoding with an outer layer that is either a centralised (§ 4) or a decentralised coordinator (§ 5). In § 6 we discuss the two encodings. Missing proofs and some additional informations can be found in [7].

2 Technical Preliminaries

A process calculus $(\mathscr{P}, \longmapsto)$ consists of a set $\mathscr{P}$ of processes (syntax) and a reduction relation $\longmapsto \subseteq \mathscr{P}^2$ (semantics). Let $\mathscr{N}$ be the countably-infinite set of names. $\tau \notin \mathscr{N}$ denotes an internal unobservable action. We use $a, b, x, \ldots$ to range over names and $P, Q, \ldots$ to range over processes. We use $\alpha, \beta \ldots$ to range over $\mathscr{N} \cup \{\tau\}$. $\tilde{a}$ denotes a sequence of names. Let $\mathrm{fn}(P)$ and $\mathrm{bn}(P)$ denote the sets of free names and bound names occurring in P, respectively. Their definitions are completely standard. We use $\sigma, \sigma', \sigma_1, \ldots$

[1] Recently [4], a slight variant of the protocol behind this algorithm was used to implement the distributed compiler DLC for a substantial subset of LNT (successor of LOTOS New Technology) that yields reasonably efficient C code.

[2] The idea and a first draft of the encoding were developed by Nestmann and van Glabbeek during a stay at NICTA, Sydney.

to range over substitutions. A substitution is a mapping $[^{x_1}/_{y_1}, \ldots, ^{x_n}/_{y_n}]$ from names to names. The application of a substitution on a term $P[^{x_1}/_{y_1}, \ldots, ^{x_n}/_{y_n}]$ is defined as the result of simultaneously replacing all free occurrences of y_i by x_i for $i \in \{ 1, \ldots, n \}$. For all names in $\mathcal{N} \setminus \{ y_1, \ldots, y_n \}$ the substitution behaves as the identity mapping. The relation $\longmapsto$ as defined in the semantics below defines the reduction steps processes can perform. We write $P \longmapsto P'$ if $(P, P') \in \longmapsto$ and call P' a *derivative* of P. Let $\Longmapsto$ denote the reflexive and transitive closure of $\longmapsto$. P is *divergent* if it has an infinite sequence of steps $P \longmapsto^{\omega}$. We use *barbs* or *observables* to distinguish between processes with different behaviours. We write $P{\downarrow_\alpha}$ if P has a barb α, where the predicate $\cdot{\downarrow_{\cdot}}$ can be defined differently for each calculus. Moreover P has a weak barb α, if P may reach a process with this barb, i.e., $P{\Downarrow_\alpha} \triangleq \exists P'.\, P \Longmapsto P' \wedge P'{\downarrow_\alpha}$.

As source calculus we use the following variant of CSP [8].

Definition 1. The processes $\mathscr{P}_{\text{CSP}}$ are given by

$$P ::= P \|_A P \quad | \quad \text{DIV} \quad | \quad \text{STOP} \quad | \quad P \sqcap P \quad | \quad P/b \quad | \quad f(P) \quad | \quad X \quad | \quad \mu X \cdot P \quad | \quad \textstyle\sum_{i \in \mathscr{I}} a_i \to P$$

where $X \in \mathscr{X}$ is a process variable, $A \subseteq \mathcal{N}$, and $\mathscr{I}$ is a finite index set.

$P \|_A Q$ is the parallel composition of P and Q, where P and Q can proceed independently except for actions $a \in A$, on which they have to synchronise. DIV describes *divergence*. STOP denotes *inaction*. *Internal choice* $P \sqcap Q$ reduces to either P or Q within a single internal step. *Concealment* P/b hides an action b and masks it as τ. *Renaming* $f(P)$ for some $f : \mathcal{N} \to \mathcal{N}$ extended by $f(\tau) = \tau$ behaves as P, where a is replaced by $f(a)$ for all $a \in \mathcal{N}$. *Recursion* $\mu X \cdot P$ describes a process behaving like P with every occurrence of X being replaced by $\mu X \cdot P$. *External choice* $\sum_{i \in \mathscr{I}} a_i \to P_i$ offers a selection of one of the *action prefixes* $a_i \to \cdot$ followed by the corresponding continuation P_i, so it may perform any a_i and then behave like P_i. Note that we enforce action prefixes to be syntactically part of an external choice construct. As usual, we use $M \,\square\, N$ to denote binary external choice.

The CSP semantics is given by the following rules, using labelled steps $\xrightarrow{\alpha}$ to define $\longmapsto$:

$$\frac{E \xrightarrow{b} E'}{E/b \xrightarrow{\tau} E'/b} \qquad \frac{E \xrightarrow{\alpha} E' \quad (\alpha \neq b)}{E/b \xrightarrow{\alpha} E'/b} \qquad \frac{E \xrightarrow{\alpha} E'}{f(E) \xrightarrow{f(\alpha)} f(E')} \qquad \frac{M_j \xrightarrow{a} M_j' \quad (j \in \mathscr{I})}{\sum_{i \in \mathscr{I}} M_i \xrightarrow{a} M_j'}$$

$$(a \to E) \xrightarrow{a} E \qquad \mu X \cdot E \xrightarrow{\tau} E[\mu X \cdot E / X]$$

$$\frac{E \xrightarrow{\alpha} E' \quad (\alpha \notin A)}{E \|_A F \xrightarrow{\alpha} E' \|_A F} \qquad \frac{F \xrightarrow{\alpha} F' \quad (\alpha \notin A)}{E \|_A F \xrightarrow{\alpha} E \|_A F'} \qquad \frac{E \xrightarrow{a} E' \quad F \xrightarrow{a} F' \quad (a \in A)}{E \|_A F \xrightarrow{a} E' \|_A F'} \qquad \frac{P \xrightarrow{\tau} P'}{P \longmapsto P'}$$

$$\text{DIV} \xrightarrow{\tau} \text{DIV} \qquad E \sqcap F \xrightarrow{\tau} E \qquad E \sqcap F \xrightarrow{\tau} F$$

A barb of CSP is the possibility of a term, to perform an action, i.e., $P{\downarrow_a} \triangleq \exists P'.\, P \xrightarrow{a} P'$. Following the definition of distributability in [20] a CSP term P is distributable into $P_1, \ldots, P_n$ if $P_1, \ldots, P_n$ are unguarded subterms of P such that every action prefix in P occurs in exactly one of the $P_1, \ldots, P_n$, where different but equally-named action prefixes are distinguished and unguarded occurrences of $\mu X \cdot P'$ may result in several copies of P' within the $P_1, \ldots, P_n$.

As target calculus we use an asynchronous variant of CCS [11] with name-passing and matching.

Definition 2. The processes $\mathscr{P}_{\text{CCS}}$ are given by

$$P ::= P \mid P \quad | \quad (\nu \tilde{c}) P \quad | \quad * \underline{c}(\tilde{x}).P \quad | \quad \underline{c}(\tilde{x}).P \quad | \quad \overline{c}\langle \tilde{x} \rangle \quad | \quad [c = z] P \quad | \quad \mathbf{0}$$

$P \mid Q$ is the parallel composition of P and Q, where P and Q can either proceed independently or synchronise on matching channels names. $(\nu\tilde{c})\,P$ restricts the visibility of actions using names in $\tilde{c}$ to P. $\underline{c}(\tilde{x}).P$ denotes input on channel c. $\overline{c}\langle\tilde{x}\rangle$ is output on channel c. Since there is no continuation, we interpret this calculus as asynchronous. We use $*\underline{c}(\tilde{x}).P$ to denote *replicated input* on channel c with the continuation P. $[x=y]\,P$ is the matching operator, if $x=y$ then P is enabled. $\mathbf{0}$ denotes inaction.

The CCS semantics is given by following transition rules:

$$\frac{P \longmapsto P'}{P \mid Q \longmapsto P' \mid Q} \qquad \frac{P \longmapsto P'}{(\nu\tilde{c})\,P \longmapsto (\nu\tilde{c})\,P'} \qquad \frac{P \equiv P' \quad P' \longmapsto Q' \quad Q' \equiv Q}{P \longmapsto Q}$$

$$*\underline{c}(\tilde{x}).P \mid \overline{c}\langle\tilde{y}\rangle \longmapsto *\underline{c}(\tilde{x}).P \mid P[\tilde{y}/\tilde{x}] \qquad \overline{c}\langle\tilde{y}\rangle \mid \underline{c}(\tilde{x}).Q \longmapsto P \mid Q[\tilde{y}/\tilde{x}]$$

where $\equiv$ denotes structural congruence given by the rules: $P \mid 0 \equiv P$, $P \mid Q \equiv Q \mid P$, $P \mid (Q \mid R) \equiv (P \mid Q) \mid R$, $(\nu\tilde{a})\,\mathbf{0} \equiv \mathbf{0}$, $P \mid (\nu\tilde{a})\,Q \equiv (\nu\tilde{a})(P \mid Q)$ if $\mathrm{bn}(\tilde{a}) \notin \mathrm{fn}(P)$, and $[x=x]\,P \equiv P$. As discussed in [20], a CCS term P is distributable into $P_1,\ldots,P_n$ if $P \equiv (\nu\tilde{x})(P_1 \mid \ldots \mid P_n)$.

Simulation Relations. The semantics of a process is usually considered modulo some behavioural equivalence. For many calculi, *the* standard reference equivalence is some form of weak bisimilarity. In the context of encodings, the source and target language often differ in their relevant obervables, i.e., barbs. In this case, it is advantageous to use a variant of reduction bisimilarity. With Gorla [6], we add a *success* operator $\checkmark$ to the syntax of both CSP and CCS. Since $\checkmark$ cannot be further reduced, the semantics is left unchanged in both cases. The test for the reachability of success is standard in both languages, i.e., $P\downarrow_{\checkmark} \triangleq \exists P'.\, P \equiv \checkmark \mid P'$. To obtain a non-trivial equivalence, we require that the bisimulation respects success and the reachability of barbs. We use the standard definition of barbs in CSP, i.e., action prefixes. Our encoding function will translate all source terms into closed terms, thus the standard definition of CCS barbs would not provide any information. Instead we use a notion of translated barb ($\cdot\Downarrow_{\llbracket\cdot\rrbracket}$) that reflects how the encoding function translates source term barbs. Its definition is given in Section 3.

Definition 3 (Bisimulation)**.** A relation $\mathscr{R} \subseteq \mathscr{P}^2$ is a *(success-sensitive, [translated-]barb-respecting, weak, reduction) bisimulation* if, whenever $(P,Q) \in \mathscr{R}$, then:

- $P \longmapsto P'$ implies $\exists Q'.\, Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$
- $Q \longmapsto Q'$ implies $\exists P'.\, P \Longmapsto P' \wedge (P',Q') \in \mathscr{R}$
- $P\Downarrow_{\checkmark}$ iff $Q\Downarrow_{\checkmark}$
- P and Q reach the same (translated) barbs, where we use $\cdot\Downarrow_a$ for CSP and $\cdot\Downarrow_{\llbracket a\rrbracket}$ for CCS

Two terms $P,Q \in \mathscr{P}$ are *bisimilar*, denoted as $P \stackrel{\centerdot}{\approx} Q$, if there exists a bisimulation that relates P and Q.

We use the symbol $\stackrel{\centerdot}{\approx}$ to denote either bisimilarity on our target language CCS or on the disjoint union of CSP and CCS that allows us to describe the relationship between source terms and their translations. In the same way we define a corresponding variant of coupled similarity.

Definition 4 (Coupled Simulation)**.** A relation $\mathscr{R} \subseteq \mathscr{P}^2$ is a *(success-sensitive, [translated-]barb-respecting, weak, reduction) coupled simulation* if, whenever $(P,Q) \in \mathscr{R}$, then:

- $P \longmapsto P'$ implies $\exists Q'.\, Q \Longmapsto Q' \wedge (P',Q') \in \mathscr{R}$ and $\exists Q''.\, Q \Longmapsto Q'' \wedge (Q'',P') \in \mathscr{R}$
- $P\Downarrow_{\checkmark}$ iff $Q\Downarrow_{\checkmark}$
- P and Q reach the same (translated) barbs, where we use $\cdot\Downarrow_a$ for CSP and $\cdot\Downarrow_{\llbracket a\rrbracket}$ for CCS

Two terms $P,Q \in \mathscr{P}$ are *coupled similar*, denoted as $P \stackrel{\centerdot}{\approx}_{\mathrm{cs}} Q$, if there exists a coupled simulation that relates P and Q in both directions.

Encodings and Quality Criteria. We consider two different translations from (the above-defined variant of) CSP into (the above-defined variant of) CCS with name passing and matching. In this context, we

refer to CSP terms as *source terms* $\mathcal{P}_S$ and to CCS terms as *target terms* $\mathcal{P}_T$. Encodings often translate single source steps into a sequence or pomset of target steps. We call such a sequence or pomset a *simulation* of the corresponding source term step. Moreover, we assume for each encoding the existence of a so-called renaming policy φ, i.e., a mapping of names from the source into vectors of target term names.

To analyse the quality of encodings and to rule out trivial or meaningless encodings, Gorla [6] provide a general framework comprising five quality criteria, which have afterwards been used in many papers. In addition to our above-mentioned definition of process calculus, whough, Gorla requires the target calculus to be equipped with a notion of behavioural equivalence $\asymp$ on target terms. Its purpose is to describe the 'abstract' behaviour of a target process, where 'abstract' refers to an observer at the source level. In [6], the equivalence $\asymp$ is often defined as a barbed equivalence (cf. [13]) or can be derived directly from the reduction semantics, and it typically is a congruence, at least with respect to parallel composition. Bisimilarity and coupled similarity are such relations on CCS terms. The criteria are:

(1) *Compositionality*: The translation of an operator op is the same for all occurrences of that operator in a term, i.e., it can be captured by a context $\mathcal{C}_{\text{op}}$ such that $\text{enc}(\text{op}(x_1,\ldots,x_n,S_1,\ldots,S_m)) = \mathcal{C}_{\mathbf{op}}^N(x_1,\ldots,x_n,\text{enc}(S_1),\ldots,\text{enc}(S_m))$ for $\text{fn}(S_1)\cup\ldots\cup\text{fn}(S_m) = N$.
(2) *Name Invariance*: The encoding does not depend on particular names, i.e., for every S and σ, it holds that $\text{enc}(\sigma(S)) \equiv \sigma'(\text{enc}(S))$ if σ is injective and $\text{enc}(\sigma(S)) \asymp \sigma'(\text{enc}(S))$ otherwise, where σ' is such that $\varphi(\sigma(n)) = \sigma'(\varphi(n))$ for every $n \in \mathcal{N}$.
(3) *Operational Correspondence*: Every computation of a source term can be simulated by its translation, i.e., $S \Longmapsto_S S'$ implies $\text{enc}(S) \Longmapsto_T \asymp \text{enc}(S')$ (completeness), and every computation of a target term corresponds to some computation of the corresponding source term (soundness, compare to Section 5).
(4) *Divergence Reflection*: The encoding does not introduce divergence, i.e., $\text{enc}(S) \longmapsto_T^\omega$ implies $S \longmapsto_S^\omega$.
(5) *Success Sensitiveness*: A source term and its encoding answer the tests for success in exactly the same way, i.e., $S \Downarrow_\checkmark$ iff $\text{enc}(S) \Downarrow_\checkmark$.

Our encodings will satisfy all of these criteria except for compositionality, because both encodings consists of two layers. [20] shows that the above criteria do not ensure that an encoding preserves distribution and proposes an additional criterion for the preservation of distributability.

Definition 5 (Preservation of Distributability)**.** An encoding $\text{enc}(\cdot)$ *preserves distributability* if for every S and for all terms $S_1,\ldots,S_n$ that are distributable within S there are some $T_1,\ldots,T_n$ that are distributable within $\text{enc}(S)$ such that $T_i \asymp \text{enc}(S_i)$ for all $1 \leq i \leq n$.

Here, because of the choice of the source and the target language, an encoding preserves distributability if for each sequence of distributable source term steps their simulations are pairwise distributable. In both languages two alternative steps of a term are in *conflict* with each other if—for CSP—they reduce the same action-prefix or—for CCS—they either reduce the same input using two outputs or they reduce the same output using two [replicated] inputs. Two alternative steps that are not in conflict are *distributable*.

3 Translating the CSP Synchronisation Mechanism

CSP and CCS—or the π-calculus—differ fundamentally in their communication and synchronisation mechanisms. In CSP there is only a single kind of action $c \to \cdot$, where c is a name. Synchronisation is implemented by the parallel operator $\cdot \|_A \cdot$ that in CSP is augmented with a set of names A containing the names that need to be synchronised at this point. By nesting parallel operators arbitrarily many actions on the same name can be synchronised. In CCS there are two different kinds of actions: inputs $\underline{c}$ and

outputs $\overline{c}$. Again synchronisation is implemented by the parallel operator, but in CCS only a single input and a single matching output can ever be synchronised within one step.

To encode the CSP communication and synchronisation mechanisms in CCS with name passing we make use of a technique already used in [17, 19] to translate between different variants of the π-calculus. CSP actions are translated into action announcements augmented with a lock indicating, whether the respective action was already used in the simulation of a step. The other operators of CSP are then translated into handlers for these announcements and locks. The translation of sum combines several actions under the same lock and thus ensures that only one term of the sum can ever be used. The translation of the parallel operator combines announcements of actions that need to be synchronised into a single announcement under a fresh lock, whose value is determined by the combination of the respective underlying locks at its left and right side. Announcements of actions that do not need to be synchronised are simply forwarded. A second layer—containing either a centralised or a decentralised coordinator—then triggers and coordinates the simulation of source term steps.

Action announcements are of the form $\overline{\mathsf{a}}\langle c, \mathsf{r}, \mathsf{l}, \mathsf{r}' \rangle$: c is the translation of the source term action. r is used to trigger the computation of the Boolean value of l. The lock l evaluates to $\top$ as long as the respective translated action was not successfully used in the simulation of a step. r' is used to guard the encoded continuation of the respective source term action. In the case of a successful simulation attempt involving this announcement, an output $\overline{\mathsf{r}'}\langle \top \rangle$ allows to unguard the encoded source term continuation and ensures that all following evaluations of l return $\bot$. The message $\overline{\mathsf{r}'}\langle \bot \rangle$ indicates an aborted simulation attempt and allows to restore l for later simulation attempts. Once a lock becomes $\bot$, all request for its computation return $\bot$.

Abbreviations. We introduce some abbreviations to simplify the presentation of the encodings. We use

$$[x \in A]\, P \triangleq \prod_{a \in A} [x = a]\, P$$

to test, whether an action belongs to the set of synchronised actions in the encoding of the parallel operator. As already done in [14, 15] we use Boolean-valued locks to ensure that every translation of an action is only used once to simulate a step. *Boolean locks* are channels on which only the Boolean values $\top$ (true) or $\bot$ (false) are transmitted. An unguarded output over a Boolean lock with value $\top$ represents a positive instantiation of the respective lock, whereas an unguarded output sending $\bot$ represents a negative instantiation. At the receiving end of such a channel, the Boolean value can be used to make a binary decision, which is done here within an IF-*construct*. This construct and according instantiations of locks are implemented as in [14, 15] using restriction and the order of transmitted values.

$$\overline{\mathsf{l}}\langle \top \rangle \triangleq \underline{\mathsf{l}}(t, f).\overline{t} \qquad \overline{\mathsf{l}}\langle \bot \rangle \triangleq \underline{\mathsf{l}}(t, f).\overline{f}$$

$$\underline{\mathsf{l}}(b).\text{IF } b \text{ THEN } P \text{ ELSE } Q \triangleq (\nu t, f)\left(\overline{\mathsf{l}}\langle t, f \rangle \mid \underline{t}.P \mid \underline{f}.Q\right)$$

We observe that the Boolean values $\top$ and $\bot$ are realised by a pair of links without parameters. Both cases of the IF-construct operate as guard for its subterms P and Q. The renaming policy φ reserves the names t and f to implement the Boolean values $\top$ and $\bot$.

The Algorithm. The encoding functions introduce some fresh names, that are reserved for special purposes. In Table 1 we list the reserved names $\mathscr{R}$ and provide a hint on their purpose. Moreover we reserve the names $\{ x_i \mid i \in \mathbb{N} \}$ and assume an injective mapping $\varphi' : \mathscr{X} \to \{ x_i \mid i \in \mathbb{N} \}$ that maps process variables of CSP to distinct names. The renaming policy φ for our encodings is then a function that reserves the names in $\mathscr{R} \cup \{ x_i \mid i \in \mathbb{N} \}$ and translates every source term name into three target term names. More precisely, choose $\varphi : \mathscr{N} \to \mathscr{N}^3$ such that:

1. No name is mapped onto a reserved name, i.e., $\varphi(n) \cap (\mathscr{R} \cup \{ x_i \mid i \in \mathbb{N} \}) = \emptyset$ for all $n \in \mathscr{N}$.
2. No two different names are mapped to overlapping sets of names, i.e., $\varphi(n) \cap \varphi(m) = \emptyset$ for all $n, m \in \mathscr{N}$ with $n \neq m$.

reserved names	purpose
a, a'	announce the ability to perform an action
c, c_{L}, c_{R}, z	(translated) source term channel, channel from the left/right of a parallel operator
l, l_{L}, l_{R}	lock, lock from the left/right of a parallel operator
l'	re-instantiate a positive sum lock
r, r_{L}, r_{R}	request the computation of the value of a lock
r', r'_i, r'_{L}, r'_{R}	simulate a source term step and unguard the corresponding continuations
n	order left announcements for the same channel that need to be synchronised
s, s'	distribute right announcements that need to be synchronised
b	Boolean value ($\bot$ or $\top$)
τ	fresh name used to announce τ-steps that result from concealment
once	used by the centralised encoding to avoid overlapping simulation attempts
m	fresh names used to encode internal choice
d	fresh names used to encode divergence
t, f	used to encode Boolean values

Table 1: Reserved Names.

We naturally extend the renaming policy to sets of names, i.e., $\varphi(X) \triangleq \{ \varphi(x) \mid x \in X \}$ if $X \subseteq \mathcal{N}$. Let $((x_1, \ldots, x_n)).i \triangleq x_i$ denote the projection of a n-tuple to ith element, if $1 \leq i \leq n$. Moreover $(X).i \triangleq \{ (x).i \mid x \in X \}$ for a set X of n-tuples and $1 \leq i \leq n$.

The inner part of our two encodings is presented in Figure 1. The most complex case is the translation of the parallel operator $\llbracket P \|_A Q \rrbracket$ that is based on the following four steps:

Step 1: Action announcements for channels $c \notin A$

In the case of actions on channels $c \notin A$—that do not need to be synchronised here—the encoding of the parallel operator acts like a forwarder and transfers action announcements of both its subtrees further up in the parallel tree. Two different restrictions of the channel for action announcements a from the left side $\llbracket P \rrbracket$ and the right side $\llbracket Q \rrbracket$, allow to trace action announcements back to their origin as it is necessary in the following case. In the present case we use a' to bridge the action announcement over the restrictions on a.

Step 2: Action announcements for channels $c \in A$

Actions $c \in A$ need to be synchronised, i.e., can be performed only if both sides of the parallel operator cooperate on this action. Simulating this kind of synchronisation is the main purpose of the encoding of the parallel operator. The renaming policy φ translates each source term name into three target term names. The first target term name is used as reference to the original source term name and transferred in announcements. The other two names are used to simulate the synchronisation of the parallel operator in CSP. Announcements from the left are translated to outputs on the respective second name and announcements from the right to the respective third name. Restriction ensures that these outputs can only be computed by the current parallel operator encoding. The translations of the announcements into different outputs for different source term names allows us to treat announcements of different names concurrently using the term $\mathsf{Synch}(c)$, where c is a source term name.

Step 3: The term $\mathsf{Synch}(c)$

In $\mathsf{Synch}(c)$ all announcements for the same source term name c from the left are ordered in order to combine each left and each right announcement on the same name. Several such announcements

$$
\begin{aligned}
\llbracket P \|_A Q \rrbracket &\triangleq \big(\nu \mathsf{a}', (\varphi(A)).2, (\varphi(A)).3\big) \Big(\\
&\quad (\nu \mathsf{a})\Big(\llbracket P \rrbracket \mid *\underline{\mathsf{a}}(c,\tilde{x}).\Big([c \in (\varphi(A)).1]\,\overline{(\varphi(c)).2}\langle \tilde{x}\rangle \mid [c \notin (\varphi(A)).1]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\Big)\Big) \\
&\quad (\nu \mathsf{a})\Big(\llbracket Q \rrbracket \mid *\underline{\mathsf{a}}(c,\tilde{x}).\Big([c \in (\varphi(A)).1]\,\overline{(\varphi(c)).3}\langle \tilde{x}\rangle \mid [c \notin (\varphi(A)).1]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\Big)\Big) \\
&\quad \mid \textstyle\prod_{c\in A} \mathsf{Synch}(c) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle \tilde{x}\rangle\Big) \\
\mathsf{Synch}(c) &\triangleq (\nu \mathsf{n})\Big(\overline{\mathsf{n}}\langle (\varphi(c)).3\rangle \\
&\quad \mid *\underline{\mathsf{n}}(\mathsf{s})\Big(\underline{(\varphi(c)).2}(\mathsf{r}_{\mathrm{L}},\mathsf{l}_{\mathrm{L}},\mathsf{r'}_{\mathrm{L}}).\Big(\,(\nu \mathsf{s}')\,\Big(\\
&\qquad *\underline{\mathsf{s}}(\mathsf{r}_{\mathrm{R}},\mathsf{l}_{\mathrm{R}},\mathsf{r'}_{\mathrm{R}}).\big((\nu \mathsf{r},\mathsf{l},\mathsf{r}')\big(\overline{\mathsf{a}}\big\langle (\varphi(c)).1,\mathsf{r},\mathsf{l},\mathsf{r}'\big\rangle \mid \mathsf{Sim}\big) \mid \overline{\mathsf{s}'}\big\langle \mathsf{r}_{\mathrm{R}},\mathsf{l}_{\mathrm{R}},\mathsf{r'}_{\mathrm{R}}\big\rangle\big) \\
&\qquad \mid (\nu \mathsf{s})\big(\overline{\mathsf{n}}\langle \mathsf{s}\rangle \mid *\underline{\mathsf{s}'}(\tilde{x}).\overline{\mathsf{s}}\langle \tilde{x}\rangle\big)\,\Big)\Big)\Big)\Big) \\
\mathsf{Sim} &\triangleq (\nu \mathsf{l}')\,\Big(\overline{\mathsf{l}'} \mid *\underline{\mathsf{l}'}.\Big(\underline{\mathsf{r}}.\Big(\overline{\mathsf{r}_{\mathrm{L}}} \mid \underline{\mathsf{l}_{\mathrm{L}}}(b).\Big(\text{IF } b \text{ THEN } \Big(\overline{\mathsf{r}_{\mathrm{R}}} \mid \underline{\mathsf{l}_{\mathrm{R}}}(b).\Big(\text{IF } b \\
&\qquad \text{THEN } \big(\overline{\mathsf{l}}\langle \top\rangle \mid \underline{\mathsf{r}'}(b).\big(\overline{\mathsf{r'}_{\mathrm{L}}}\langle b\rangle \mid \overline{\mathsf{r'}_{\mathrm{R}}}\langle b\rangle \mid \text{IF } b \text{ THEN } *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \bot\rangle \text{ ELSE } \overline{\mathsf{l}'}\big)\big) \\
&\qquad \text{ELSE } \big(\overline{\mathsf{l}}\langle \bot\rangle \mid \overline{\mathsf{r'}_{\mathrm{L}}}\langle \bot\rangle\big) \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \bot\rangle\Big)\Big) \\
&\quad \text{ELSE } \big(\overline{\mathsf{l}}\langle \bot\rangle \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \bot\rangle\big)\Big)\Big)\Big)\Big) \\
\llbracket \textstyle\sum_{i\in\mathscr{I}} c_i \to P_i \rrbracket &\triangleq (\nu \mathsf{r},\mathsf{l},\mathsf{r'}_1,\ldots,\mathsf{r'}_n)\,\Big(\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \top\rangle \\
&\quad \mid \textstyle\prod_{i\in\mathscr{I}} \big(\overline{\mathsf{a}}\langle (c_i).1,\mathsf{r},\mathsf{l},\mathsf{r'}_i\rangle \mid *\underline{\mathsf{r'}_i}(b).\text{IF } b \text{ THEN } \big(\llbracket P_i \rrbracket \mid *\underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \bot\rangle\big) \text{ ELSE } \underline{\mathsf{r}}.\overline{\mathsf{l}}\langle \top\rangle\big)\Big) \\
\llbracket (P)/z \rrbracket &\triangleq (\nu \mathsf{a}')\big((\nu \mathsf{a},z)\big(\llbracket P \rrbracket \mid *\underline{\mathsf{a}}(c,\tilde{x}).\big([c=z]\,\overline{\mathsf{a}'}\langle \tau,\tilde{x}\rangle \mid [c\neq z]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\big)\big) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle \tilde{x}\rangle\big) \\
\llbracket f(P) \rrbracket &\triangleq (\nu \mathsf{a}')\,\big((\nu \mathsf{a},z)\,\big(\llbracket P \rrbracket \mid *\underline{\mathsf{a}}(c,\tilde{x})\,\big(\textstyle\prod_{z/x\in f} [c=(\varphi(x)).1]\,\overline{\mathsf{a}'}\langle (\varphi(z)).1,\tilde{x}\rangle \\
&\quad \mid [c\notin \mathrm{dom}(f)]\,\overline{\mathsf{a}'}\langle c,\tilde{x}\rangle\,\big)\big) \mid *\underline{\mathsf{a}'}(\tilde{x}).\overline{\mathsf{a}}\langle \tilde{x}\rangle\big) \\
\llbracket \mathrm{DIV} \rrbracket &\triangleq (\nu \mathsf{d})\big(\overline{\mathsf{d}} \mid *\underline{\mathsf{d}}.\overline{\mathsf{d}}\big) \\
\llbracket \mu X \cdot P \rrbracket &\triangleq (\nu \varphi'(X))\Big(\overline{\varphi'(X)} \mid *\underline{\varphi'(X)}.\llbracket P \rrbracket\Big) \\
\llbracket X \rrbracket &\triangleq \overline{\varphi'(X)} \\
\llbracket P \sqcap Q \rrbracket &\triangleq (\nu \mathsf{m})(\underline{\mathsf{m}}.\llbracket P \rrbracket \mid \underline{\mathsf{m}}.\llbracket Q \rrbracket \mid \overline{\mathsf{m}}) \\
\llbracket \mathrm{STOP} \rrbracket &\triangleq \mathbf{0} \\
\llbracket \checkmark \rrbracket &\triangleq \checkmark
\end{aligned}
$$

where $\notin (\varphi(A)).1$ is short for $\in (\mathrm{fn}(P)\cup\mathrm{fn}(Q))\setminus(\varphi(A)).1$, $\notin \mathrm{dom}(f)$ is short for $\in \mathrm{fn}(P)\setminus\mathrm{dom}(f)$, and $\neq z$ is short for $\in \mathrm{fn}(P)\setminus\{\,z\,\}$.

Figure 1: An encoding from CSP into CCS with value passing (inner part).

may result from underlying parallel operators, sums with similar summands, and junk left over from already simulated source term steps. For each left announcement a fresh instance of s is generated and restricted. The names s and s' are used to transfer right announcements to the respective next left announcement, where s' is used to bridge over the restriction on s. This way each right announcement will eventually be transferred to each left announcement on the same name. Note that this kind of forwarding is not done concurrently but in the source language a term $P \|_A Q$ also cannot perform two steps on the same name $c \in A$ concurrently. After combining a left and a right announcement on the same source term name a fresh set of auxiliary variables $\mathsf{r}, \mathsf{l}, \mathsf{r}'$ is generated and a corresponding announcement is transmitted. The term Sim reacts to requests regarding this announcement and is used to simulate a step on the synchronised action.

Step 4: The term Sim

If a request reaches Sim it starts questioning the left and the right side. First the left side is requested to compute the current value of the lock of the action. Only if $\top$ is returned, the right side is requested to compute its lock as well. This avoids deadlocks that would result from blindly requesting the computation of locks in the decentralised encoding. If the locks of both sides are still valid the fresh lock l returns $\top$ else $\perp$ is returned. For each case Sim ensures that subsequently requests will obtain an answer by looping with $\overline{\mathsf{l}'}$ or returning $\perp$ to all requests, respectively. The messages $\overline{\mathsf{r}'_{\mathrm{L}}}\langle\perp\rangle$ and $\overline{\mathsf{r}'_{\mathrm{R}}}\langle\perp\rangle$ cause the respective underlying subterms on the left and the right side to do the same, whereas $\overline{\mathsf{r}'_{\mathrm{L}}}\langle\top\rangle$ and $\overline{\mathsf{r}'_{\mathrm{R}}}\langle\top\rangle$ cause the unguarding of encoded continuations as result of a successful simulation of a source term synchronisation step.

Basic Properties and Translated Observables. The protocol introduced by the encoding function in Figure 1 (and its outer parts introduced later) simulates a single source term step by a sequence of target term steps. Most of these steps are merely pre- and post-processing steps, i.e., they do not participate in decisions regarding the simulation of conflicting source term steps but only prepare and complete simulations. Accordingly we distinguish between *auxiliary steps*—that are pre- and post-processing steps—and *simulation steps*—that mark a point of no return by deciding which source term step is simulated. Note that the points of no return and thus the definition of auxiliary and simulation steps is different in the two variants of our encoding.

Auxiliary steps do not influence the choice of source terms steps that are simulated. Moreover they operate on restricted channels, i.e., are unobservable. Accordingly they do not change the state of the target term modulo the considered reference relations $\dot{\approx}$ and $\dot{\approx}_{\mathrm{cs}}$. We introduce some auxiliary lemmata to support this claim.

The encoding $\llbracket \cdot \rrbracket$ translates source term barbs c into free announcements with $(\varphi(c)).1$ as first value and a lock l as third value that computes to $\top$. The two coordinators, i.e., outer encodings, we introduce later, restrict the free a-channel of $\llbracket \cdot \rrbracket$.

Definition 6 (Translated Barbs)**.** Let $T \in \mathscr{P}_{\mathrm{T}}$ such that $\exists S.\ \llbracket S \rrbracket \Longmapsto_{\mathrm{T}} T$, $\exists S.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$, or $\exists S.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T$. T has a translated barb c, denoted by $T \downarrow_{\llbracket c \rrbracket}$, if

- there is an unguarded output $\overline{\mathsf{a}}\langle(\varphi(c)).1, \mathsf{r}, \mathsf{l}, \mathsf{r}'\rangle$—on a free channel a in the case of $\llbracket \cdot \rrbracket$ or the outermost variant of a in the case of the later introduced encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$—in T or
- such an announcement was consumed to unguard an IF-construct testing l and this construct is still not resolved in T

such that all locks that are necessary to instantiate l are positively instantiated.

Analysing the encoding function in Figure 1 we observe that an encoded source term has a translated barb iff the corresponding source term has the corresponding source term barb.

$$[\![P]\!] \triangleq (\nu \mathsf{a}, \mathsf{once})\big(\llbracket P \rrbracket \mid \overline{\mathsf{once}} \mid *\underline{\mathsf{once}}.\underline{\mathsf{a}}\big(c,\mathsf{r},\mathsf{l},\mathsf{r}'\big).\big(\bar{\mathsf{r}} \mid \underline{\mathsf{l}}(b).\big(\overline{\mathsf{once}} \mid \text{IF } b \text{ THEN } \overline{\mathsf{r}'}\langle\top\rangle\big)\big)\big)$$

Figure 2: A **centralised** encoding from CSP into CCS with value passing.

Observation 7. For all $S \in \mathscr{P}_{\mathrm{S}}$, it holds $S{\downarrow_c}$ iff $\llbracket S \rrbracket {\downarrow_{\llbracket c \rrbracket}}$.

All instances of success in the translation result from success in the source. More precisely the only way to obtain ✓ in the translation is by $\llbracket \checkmark \rrbracket \triangleq \checkmark$.

Observation 8. For all $S \in \mathscr{P}_{\mathrm{S}}$, it holds $S{\downarrow_\checkmark}$ iff $\llbracket S \rrbracket {\downarrow_\checkmark}$.

The encoding propagates announcements through the translated parallel structure. In the translation of parallel operators it combines all left and right announcements w.r.t. to the same channel name, if this channel needs to be synchronised. Therefore we copy announcements. We use locks carrying a Boolean value to indicate whether an announcement was already used to simulate a source term step. These locks carry $\top$ in the beginning and are swapped to $\bot$ as soon as the announcement was used. In each state there is at most one positive instantiation of each lock and as soon as a lock is instantiated negatively it never becomes positive again.

Lemma 9. *Let $T \in \mathscr{P}_{\mathrm{T}}$ such that $\exists S.\ \llbracket S \rrbracket \Longmapsto_{\mathrm{T}} T$. Then for each variant l of the names $\mathsf{l}, \mathsf{l}_{\mathrm{L}}, \mathsf{l}_{\mathrm{R}}$*

1. *there is at most one positive instantiation of l in T,*
2. *if there is a positive instantiation of l in T then there is no other instantiation of l in T,*
3. *if there is a negative instantiation of l in T then no derivative of T contains a positive instantiation of l.*

4 The Centralised Encoding

Figure 1 describes how to translate CSP actions into announcements augmented with locks and how the other operators are translated to either forward or combine these announcements and locks. With that $\llbracket \cdot \rrbracket$ provides the basic machinery of our encoding from CSP into CCS with name passing and matching. However it does not allow to simulate any source term step. Therefore we need a second (outer) layer that triggers and coordinates the simulation of source term steps. We consider two ways to implement this coordinator: a centralised and a decentralised coordinator. The centralised coordinator is depicted in Figure 2.

The channel once is used to ensure that simulation attempts of different source term steps cannot overlap each other. For each simulation attempt exactly one announcement is consumed. The coordinator then triggers the computation of the respective lock that was transmitted in the announcement. This request for the computation of the lock is propagated along the parallel structure induced by the translations of parallel operators until—in the leafs—encodings of sums are reached. There the request for the computation yields the transmission of the current value of the respective lock. While being transmitted back to the top of the tree, different locks that refer to synchronisation in the source terms are combined. If the computation of the lock results with $\top$ at the top of the tree, the respective source term step is simulated. Else the encoding aborts the simulation attempt and restores the consumed informations about the values of the respective locks. In both cases a new instance of $\overline{\mathsf{once}}$ allows to start the next simulation attempt. Accordingly only some post-processing steps can overlap with a new simulation attempt.

As we prove below, the points of no return in the centralised encoding can result from the consumption of action announcements by the outer encoding in Figure 2 if the corresponding lock computes to

$\top$. Moreover the encoding of internal choice and divergence introduces simulation steps, namely all steps on variants of the channels m, d, and $\varphi'(X)$. All remaining steps of the centralised encoding are auxiliary.

Definition 10 (Auxiliary and Simulation Steps)**.** A step $T \longmapsto_{\mathrm{T}} T'$ such that $\exists S \in \mathscr{P}_{\mathrm{S}}.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ is called a *simulation step*, denoted by $T \overset{\mapsto}{\longmapsto} T'$, if $T \longmapsto T'$ is a step on the outermost channel a and the computation of the value of the received lock l will return $\top$ or it is a step on a variant of m, d, or $\varphi'(X)$.

Else the step $T \longmapsto_{\mathrm{T}} T'$ is called an *auxiliary step*, denoted by $T \overset{\cdot}{\longmapsto} T'$.

Let $\overset{\cdot}{\Longmapsto}$ denote the reflexive and transitive closure of $\overset{\cdot}{\longmapsto}$ and let $\overset{\mapsto}{\Longmapsto} \triangleq \overset{\cdot}{\Longmapsto}\overset{\mapsto}{\longmapsto}\overset{\cdot}{\Longmapsto}$. Auxiliary steps do not change the state modulo $\dot{\approx}$.

Lemma 11. *$T \overset{\cdot}{\longmapsto} T'$ implies $T \dot{\approx} T'$ for all target terms T, T'.*

By distinguishing auxiliary and simulation steps, we can prove a condition stronger than operational correspondence, namely that each source term step is simulated by exactly one simulation step.

Lemma 12. *For all S, S', it holds $S \longmapsto_{\mathrm{S}} S'$ iff $\exists T.\ [\![S]\!] \overset{\mapsto}{\Longmapsto} T \wedge [\![S']\!] \dot{\approx} T$.*

This direct correspondence between source term steps and the points of no return of their translation allows us to prove a variant of operational correspondence that is significantly stricter than the variant proposed in [6].

Definition 13 (Operational Correspondence)**.**

An encoding $\mathrm{enc}(\cdot) : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is *operationally corresponding* w.r.t. $\dot{\approx} \subseteq \mathscr{P}_{\mathrm{T}}^2$ if it is:

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ implies $\exists T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T \wedge [\![S']\!] \dot{\approx} T$

Sound: $\forall S, T.\ [\![S]\!] \Longmapsto_{\mathrm{T}} T$ implies $\exists S'.\ S \Longmapsto_{\mathrm{S}} S' \wedge [\![S']\!] \dot{\approx} T$

The 'if'-part of Lemma 12 implies operational completeness w.r.t. $\dot{\approx}$ and the 'only-if'-part contains the main argument for operational soundness w.r.t. $\dot{\approx}$. Hence $[\![\cdot]\!]$ is operationally corresponding w.r.t. to $\dot{\approx}$.

Theorem 1. *The encoding $[\![\cdot]\!]$ is operationally corresponding w.r.t. to $\dot{\approx}$.*

To obtain divergence reflection we show that there is no infinite sequence of only auxiliary steps. Then divergence reflection follows from the combination of this fact and Lemma 12.

Theorem 2. *The encoding $[\![\cdot]\!]$ reflects divergence.*

The encoding function ensures that $[\![S]\!]$ has an unguarded occurrence of $\checkmark$ iff S has such an unguarded occurrence. Operational correspondence ensures that S and $[\![S]\!]$ also answer the question for the reachability of $\checkmark$ in the same way.

Theorem 3. *The encoding $[\![\cdot]\!]$ is success sensitive.*

In a similar way we can prove that a source term reaches a barb iff its translation reaches the respective translated barb.

Theorem 4. *For all S, c, it holds $S \Downarrow_c$ iff $[\![S]\!] \Downarrow_{[\![c]\!]}$.*

As proved in [18], Theorem 1, the fact that $\dot{\approx}$ is success sensitive and respects (translated) barbs, Theorem 3, and Theorem 4 imply that for all S it holds S and $[\![S]\!]$ are (success sensitive, (translated) barb respecting, weak, reduction) bisimilar, i.e., $S \dot{\approx} [\![S]\!]$. Bisimilarity is a strong relation between source terms and their translation. On the other hand, because of efficiency, distributability preserving encodings are more interesting. Because of once the encoding $[\![\cdot]\!]$ obviously does not preserve distributability. As discussed in [16] bisimulation often forbids distributed encodings. Instead they propose coupled simulation as a relation that still provides a strong connection between source terms and their translations but is more flexible. Following the approach in [16] we consider a decentralised coordinator next.

$$(\!|P|\!) \triangleq (\nu \mathsf{a})\big(\llbracket P \rrbracket \mid *\underline{\mathsf{a}}\big(c,\mathsf{r},\mathsf{l},\mathsf{r}'\big).\big(\overline{\mathsf{r}} \mid \underline{\mathsf{l}}(b).\text{IF } b \text{ THEN } \overline{\mathsf{r}'}\langle\top\rangle\big)\big)$$

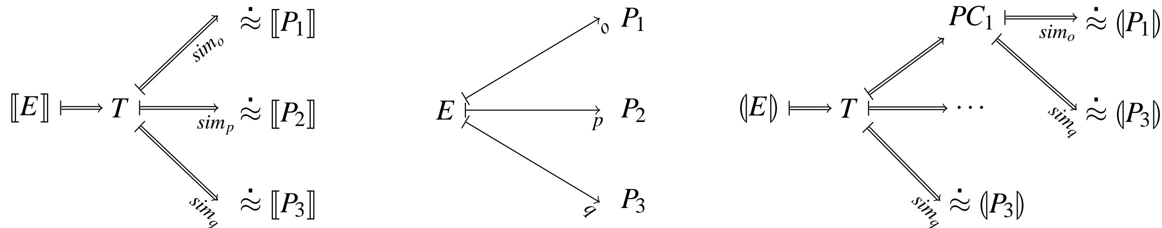
Figure 3: A **decentralised** encoding from CSP into CCS with value passing.

5 The Decentralised Encoding

Figure 3 presents a decentralised variant of the coordinator in Figure 2. The only difference between the centralised and the decentralised version of the coordinator is that the latter can request to check different locks concurrently. Technically $\llbracket\cdot\rrbracket$ and $(\!|\cdot|\!)$ differ only by the use of once. As a consequence the steps of different simulation attempts can overlap and even (pre-processing) steps of simulations of conflicting source term steps can interleave to a certain degree. Because of this effect, $(\!|\cdot|\!)$ does not satisfy the version of operational correspondence used above for $\llbracket\cdot\rrbracket$, but $(\!|\cdot|\!)$ satisfies weak operational correspondence that was proposed in [6] as part of a set of quality criteria.

Since several announcements can be processed concurrently by the decentralised coordinator, here all consumptions of announcements are auxiliary steps. Instead the consumption of positive instantiations of locks can mark a point of no return. In contrast to $\llbracket\cdot\rrbracket$ not every point of no return in $(\!|\cdot|\!)$ unambiguously marks a simulation of a single source term step, because in contrast to $\llbracket\cdot\rrbracket$ the encoding $(\!|\cdot|\!)$ introduces *partial commitments* [17, 19].

Consider the example $E = (o \to P_1 \,\square\, p \to P_2) \parallel_{\{\,o,p\,\}} (o \to P_3 \,\square\, p \to P_4 \,\square\, q \to P_5)$.



In the example, two sides of a parallel operator have to synchronise on either action p, or action o, or action q happens without synchronisation. In the centralised encoding $\llbracket E \rrbracket$ the use of once ensures that different simulation attempts cannot overlap. Thus, only after finishing the simulation of a source term step, the simulation of another source term step can be invoked. As a consequence each state reachable from encoded source terms can unambiguously be mapped to a single state of the source term. This allows us to use a stronger version of operational correspondence and, thus, to prove that source terms and their translations are bisimilar. The corresponding 1-to-1 correspondence between source terms and their translations is visualised by the first two graphs above, where $T \dot\approx \llbracket E \rrbracket$.

The decentralised encoding $(\!|E|\!)$ introduces partial commitments. Assume the translation of a source term that offers several alternative ways to be reduced. Then some encodings—as our decentralised one—do not always decide on which of the source term steps should be simulated next. More precisely a partial commitment refers to a state reachable from the translation of a source term in that already some possible simulations of source term steps are ruled out, but there is still more than a single possibility left.

In the decentralised encoding announcements can be processed concurrently and parts of different simulation attempts can interleave. The only blocking part of the decentralised encoding are conflicting attempts to consume the same positive instantiation of a lock. In the presented example above there are

two locks; one for each side of the parallel operator. The simulations of the step on o and p need both of these locks, whereas to simulate the step on q only a positive instantiation of the right lock needs to be consumed. By consuming the positive instantiation of the left lock in an attempt to simulate the step on o, the simulation of the step on p is ruled out, but the simulation of the step on q is still possible. Since either the simulation of the step on o or the simulation of the step on q succeeds, the simulation of the step on p is not only blocked but ruled out. But the consumption of the instantiation of the left lock does not unambiguously decide between the remaining two simulations. The intermediate state that results from consuming the instantiation of the left lock and represents a partial commitment is visualised in the right graph above by the state PC_1.

Partial commitments forbid a 1-to-1 mapping between the states of a source term and its translations by a bisimulation. But, as shown in [16], partial commitments do not forbid to relate source terms and their translations by coupled similarity.

Whether the consumption of a positive instantiation of a lock is an auxiliary step—does not change the state of the term modulo $\dot{\approx}$—, is a partial commitment, or unambiguously marks a simulation of a single source term step depends on the surrounding term, i.e., cannot be determined without the context. For simplicity we consider all steps that reduce a positive instantiation of a lock as simulation steps. Also steps on variants of the channels m, d, and $\varphi'(X)$ are simulation steps, because they unambiguously mark a simulation of a single source term step. All remaining steps of the decentralised encoding are auxiliary.

Definition 14 (Auxiliary and Simulation Steps)**.** A step $T \longmapsto_{\mathrm{T}} T'$ such that $\exists S \in \mathscr{P}_{\mathrm{S}}.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T$ is called a *simulation step*, denoted by $T \overset{\mapsto}{\longmapsto} T'$, if $T \longmapsto T'$ reduces a positive instantiation of a lock or is a step on a variant of m, d, or $\varphi'(X)$.

Else the step $T \longmapsto_{\mathrm{T}} T'$ is called an *auxiliary step*, denoted by $T \overset{\cdot}{\longmapsto} T'$.

Again let $\overset{\cdot}{\Longmapsto}$ denote the reflexive and transitive closure of $\overset{\cdot}{\longmapsto}$ and let $\overset{\mapsto}{\Longmapsto} \triangleq \overset{\cdot}{\Longmapsto}\overset{\mapsto}{\longmapsto}\overset{\cdot}{\Longmapsto}$. Since auxiliary steps do not introduce partial commitments, they do not change the state modulo $\dot{\approx}$. The proof of this lemma is very similar to the centralised case.

Lemma 15. *$T \overset{\cdot}{\longmapsto} T'$ implies $T \dot{\approx} T'$ for all target terms T, T'.*

In contrast to the centralised encoding, the simulation of a source term step in the decentralised encoding can require more than a single simulation step and a single simulation step not unambiguously refers to the simulation of a particular source term step. The partial commitments described above forbid operational correspondence, but the weaker variant proposed in [6] is satisfied. We call this variant weak operational correspondence.

Definition 16 (Weak Operational Correspondence)**.**

An encoding $\mathrm{enc}(\cdot) : \mathscr{P}_{\mathrm{S}} \to \mathscr{P}_{\mathrm{T}}$ is *weakly operationally corresponding* w.r.t. $\dot{\approx}_{\mathrm{cs}} \subseteq \mathscr{P}_{\mathrm{T}}^2$ if it is:

Complete: $\forall S, S'.\ S \Longmapsto_{\mathrm{S}} S'$ implies $\exists T.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T \wedge (\!|S'|\!) \dot{\approx}_{\mathrm{cs}} T$

Weakly Sound: $\forall S, T.\ (\!|S|\!) \Longmapsto_{\mathrm{T}} T$ implies $\exists S', T'.\ S \Longmapsto_{\mathrm{S}} S' \wedge T \Longmapsto_{\mathrm{T}} T' \wedge (\!|S'|\!) \dot{\approx}_{\mathrm{cs}} T'$

The only difference to operational correspondence is the weaker variant of soundness that allows for T to be an intermediate state that does not need to be related to a source term directly. Instead there has to be a way from T to some T' such that T' is related to a source term.

Theorem 5. *The encoding $(\!|\cdot|\!)$ is weakly operational corresponding w.r.t. to $\dot{\approx}$.*

As in the encoding $[\![\cdot]\!]$, there is no infinite sequence of only auxiliary steps in $(\!|S|\!)$. Moreover each simulation of a source term requires only finitely many simulation steps (to consume the respective positive instantiations of locks). Thus $(\!|\cdot|\!)$ reflects divergence.

Theorem 6. *The encoding $(\!|\cdot|\!)$ reflects divergence.*

The encoding function ensures that $(\!|S|\!)$ has an unguarded occurrence of $\checkmark$ iff S has such an unguarded occurrence. Operational correspondence again ensures that S and $(\!|S|\!)$ also answer the question for the reachability of $\checkmark$ in the same way.

Theorem 7. *The encoding $(\!|\cdot|\!)$ is success sensitive.*

Similarly, a source term reaches a barb iff its translation reaches the respective translated barb.

Theorem 8. *For all S, c, it holds $S \Downarrow_c$ iff $(\!|S|\!) \Downarrow_{\llbracket c \rrbracket}$.*

Weak operational correspondence does not suffice to establish a bisimulation between source terms and their translations. But, as proved in [18], Theorem 5, the fact that $\dot{\approx}$ is success sensitive and respects (translated) observables, Theorem 7, and Theorem 8 imply that $\forall S.\ S$ and $\llbracket S \rrbracket$ are (success sensitive, (translated) barbs respecting, weak, reduction) coupled similar, i.e., $S \dot{\approx}_{\text{cs}} (\!|S|\!)$.

It remains to show, that $(\!|\cdot|\!)$ indeed preserves distributability. Therefore we prove that all blocking parts of the encoding $(\!|\cdot|\!)$ refer to simulations of conflicting source term steps.

Theorem 9. *The encoding $(\!|\cdot|\!)$ preserves distributability.*

6 Conclusions

We introduced two encodings from CSP into asynchronous CCS with name passing and matching. As in [16] we had to encode the multiway synchronisation mechanism of CSP into binary communications and, similarly to [16], we did so first using a centralised controller that was then modified into a decentralised controller. By doing so we were able to transfer the observations of [16] to the present case:

1. The centralised solution allows to prove a stronger connection between source terms and their translations, namely by bisimilarity. Our decentralised solution does not relate source terms and their translations that strongly and we doubt that any decentralised solution can do so.
2. Nonetheless, decentralised solutions are possible as presented by the second encoding and they still relate source terms and their translations in an interesting way, namely by coupled similarity.

Thus as in [16] we observed a trade-off between *centralised* but *bisimilar* solutions on the one-hand side and *decentralised* but only *coupled similar* solutions on the other side.

More technically we showed here instead a trade-off between centralised but *operationally corresponding* solutions on the one-hand side and *weakly operationally corresponding* but decentralised solutions on the other side. The mutual connection between operational correspondence and bisimilarity as well as between weak operational correspondence and coupled similarity is proved in [18].

Both encodings make strict use of the renaming policy and translate into closed terms. Hence the criterion *name invariance* is trivially satisfied in both cases. Moreover we showed that both encodings are *success-sensitive*, *reflect divergence*, and even *respect barbs* w.r.t. to the standard source term (CSP) barbs and a notion of translated barbs on the target. The centralised encoding $\llbracket \cdot \rrbracket$ additionally satisfies a variant of *operational correspondence* that is stricter than the variant proposed in [6]. The decentralised encoding $(\!|\cdot|\!)$ satisfies *weak operational correspondence* as proposed in [6] and *distributability preservation* as proposed in [20]. Thus both encodings satisfy all of the criteria proposed in [6] except for compositionality. However in both cases the inner part is obviously compositional and the outer part only adds a fixed context.

References

[1] J. C. M. Baeten (2005): *A Brief History of Process Algebra*. *Theor. Comput. Sci.* 335(2-3), pp. 131–146, doi:10.1016/j.tcs.2004.07.036.

[2] E. Brinksma (1985): *A tutorial on LOTOS*. In: *Proc. of PSTV*, pp. 171–194.

[3] S. D. Brookes (1983): *On the Relationship of CCS and CSP*. In: *Proc. of ICALP*, *LNCS* 154, pp. 83–96, doi:10.1007/BFb0036899.

[4] H. Evrard & F. Lang (2015): *Automatic Distributed Code Generation from Formal Models of Asynchronous Concurrent Processes*. In: *Proc. of PDP*, IEEE, pp. 459–466, doi:10.1109/PDP.2015.96.

[5] R. van Glabbeek (2012): *Musings on Encodings and Expressiveness*. In: *Proc. of EXPRESS/SOS*, *EPTCS* 89, pp. 81–98, doi:10.4204/EPTCS.89.7.

[6] D. Gorla (2010): *Towards a Unified Approach to Encodability and Separation Results for Process Calculi*. *Information and Computation* 208(9), pp. 1031–1053, doi:10.1016/j.ic.2010.05.002.

[7] M. Hatzel, C. Wagner, K. Peters & U. Nestmann (2015): *Encoding CSP into CCS (Extended Version)*. Technical Report. Available at `http://arxiv.org/abs/1508.01127`.

[8] C. A. R. Hoare (1978): *Communicating Sequential Processes*. *Communications of the ACM* 21(8), pp. 666–677, doi:10.1145/359576.359585.

[9] C. A. R. Hoare (2006): *Why ever CSP?* *Electronic Notes in Theoretical Computer Science* 162(0), pp. 209–215, doi:10.1016/j.entcs.2006.01.031.

[10] I. Lanese & U. Montanari (2006): *Hoare vs Milner: Comparing Synchronizations in a Graphical Framework With Mobility*. *Electronic Notes in Theoretical Computer Science* 154(2), pp. 55 – 72, doi:10.1016/j.entcs.2005.03.032.

[11] R. Milner (1980): *A calculus of communicating systems*. Springer, doi:10.1007/3-540-10235-3.

[12] R. Milner (1986): *Process Constructors and Interpretations (Invited Paper)*. In: *IFIP Congress*, pp. 507–514.

[13] R. Milner & D. Sangiorgi (1992): *Barbed Bisimulation*. In: *Proc. of ICALP*, *LNCS* 623, pp. 685–695, doi:10.1007/3-540-55719-9_114.

[14] U. Nestmann (1996): *On Determinacy and Nondeterminacy in Concurrent Programming*. Ph.D. thesis, Universität Erlangen-Nürnberg.

[15] U. Nestmann & B. C. Pierce (2000): *Decoding Choice Encodings*. *Information and Computation* 163(1), pp. 1–59, doi:10.1006/inco.2000.2868.

[16] J. Parrow & P. Sjödin (1992): *Multiway synchronization verified with coupled simulation*. In: *Proc. of CONCUR*, *LNCS* 630, Springer, pp. 518–533, doi:10.1007/bfb0084813.

[17] K. Peters (2012): *Translational Expressiveness*. Ph.D. thesis, TU Berlin.

[18] K. Peters & R. van Glabbeek (2015): *Analysing and Comparing Encodability Criteria*. In: *Proc. of EXPRESS/SOS*, EPTCS 190, pp. 46–60, doi:10.4204/EPTCS.190.4.

[19] K. Peters & U. Nestmann (2012): *Is it a "Good" Encoding of Mixed Choice?* In: *Proc. of FoSSaCS*, *LNCS* 7213, pp. 210–224, doi:10.1007/978-3-642-28729-9_14.

[20] K. Peters, U. Nestmann & U. Goltz (2013): *On Distributability in Process Calculi*. In: *Proc. of ESOP*, *LNCS* 7792, Springer, pp. 310–329, doi:10.1007/978-3-642-37036-6_18.

[21] P. Sjödin (1991): *From LOTOS Specifications to Distributed Implementations*. Ph.D. thesis, Department of Computer Science, Uppsala University. Available as Report DoCS 91/31.