From CCS to CSP: the m-among-n Synchronisation Approach

Gerard Ekembe Ngondi, Vasileios Koutavas, Andrew Butterfield
Trinity College Dublin, Lero - the SFI Software Research Centre
gerard.ekembe, vkoutav, andrew.butterfield @tcd.ie

We present an alternative translation from CCS to an extension of CSP based on m-among-n synchronisation (called CSPmn). This translation is correct up to strong bisimulation. Unlike the g-star renaming approach ([4]), this translation is not limited by replication (viz., recursion with no nested parallel composition). We show that m-among-n synchronisation can be implemented in CSP based on multiway synchronisation and renaming.

1 Introduction

In [4], the authors present a translation from CCS [1] into CSP [22, 20], *ccs*2*csp*, which is correct up to strong bisimulation (cf. [10]). This means that a CCS process is strong bisimilar to its CSP translation. *ccs*2*csp* has been implemented in Haskell (cf. [23]), which allows using the model-checker FDR [7] for analysing translated CCS terms. In the course of the same work, the authors have proposed an alternative translation, $ccs2csp_2$, correct up to failure equivalence. Both translations differ in the translation of the prefix term $\tau.P$, translated into $(tau \to ccs2csp(P))\backslash_{csp}\{tau\}$ in the first case, and $ccs2csp_2(P)$ in the second case.

In this paper we present yet a third alternative, $ccs2csp_3$, achieved by first extending CSP with m-among-n synchronisation [9], from which we can derive multiway (or n-among-n) synchronisation, the default CSP synchronisation mechanism, and binary syncronisation (used in CCS). Then, we translate CCS parallel composition into the binary version of CSP parallel operator. The resulting translation is correct up to strong bisimulation.

The translations in [4] were achieved by hard coding binary synchronisation into CCS before going to CSP. Using a renaming function, g^*, the translations generated unique pairs of indices between any two pairs of complementary prefixes in a parallel composition, e.g., $(a,\bar{a}) \mapsto \{(a_{12},\bar{a}_{12}),(a_{13},\bar{a}_{13})\}$. This effectively made synchronising prefix pairs unique. Although these indices were generated in CCS, the g^*-renaming approach shows how to enforce binary synchronisation even in CSP: given a CSP process $P \underset{a}{\parallel} Q \underset{a}{\parallel} R$, to ensure binary synchronisations on a, assign unique indices to a accordingly, through renaming. E.g., $P[\{a_{12},a_{13}\}/a] \parallel Q[a_{12}/a] \parallel R[a_{13}/a]$ ensures that pairs of processes (P,Q) and (P,R) can synchronise respectively, but not (Q,R). This approach, which we call the Gstar approach, has been encoded in the translation tool and the resulting CSP terms can be analysed in FDR immediately.

m-among-n synchronisation [9] demands adding new rules to CSP, hence it would require updating FDR first. In other words, the CSP terms resulting from our new translation, $ccs2csp_3$, cannot immediately be analysed in FDR. Nonetheless, function g^* implements binary synchronisation, hence, can be taken for an implementation of 2-among-n synchronisation.

The Gstar approach does *not* allow translating recursive terms with nested parallelism (or replication). That is because function g^* needs to generate every synchronisation index so the translation can

V. Castiglioni and C. A. Mezzina (Eds): Combined Workshop on Expressiveness in Concurrency and Structural Operational Semantics 2022 (EXPRESS/SOS 2022).
EPTCS 368, 2022, pp. 60–74, doi:10.4204/EPTCS.368.4

terminate. With m-among-n synchronisation, we need only one index to separate interleaving from synchronisation, i.e., we map every CCS name unto two CSP events, e.g., $a \mapsto \{a, a_S\}$, where a_S is the synchronisation event. Therefore, this new translation is not limited by parallel under recursion.

Our main contribution in this paper hence is a new translation from CCS into CSP which is correct up to strong bisimulation, is not limited by parallel under recursion, but cannot be immediately analysed with FDR. As a byproduct, we define m-among-n synchronisation for CSP processes. We call the corresponding extension CSPmn. We show that CSPmn preserves CSP axioms by defining m-among-n sysnchronisation in terms of both multiway synchronisation and renaming. The translation from CSPmn into CSP is limited by parallel under recursion as it requires generating unique indices for all possible combinations of synchronising processes.

2 Correct Translation, CCS(Tau), CSP, CCS-to-CSP

2.1 Correct Translations

A correct translation of one language into another is a mapping from the valid expressions in the first language to those in the second, that preserves their meaning (for some definition of meaning). Below we recall the two main definitions of correctness from [10].

Let $\mathscr{L} = (\mathbb{T}_{\mathscr{L}}, [\![\,]\!]_{\mathscr{L}})$ denote a language as a pair of a set $\mathbb{T}_{\mathscr{L}}$ of valid expressions in $\mathscr{L}$ and a surjective mapping $[\![\,]\!]_{\mathscr{L}} : \mathbb{T}_{\mathscr{L}} \to \mathscr{D}_{\mathscr{L}}$ from $\mathbb{T}_{\mathscr{L}}$ to some set of meanings $\mathscr{D}_{\mathscr{L}}$. Candidate instances of $[\![\,]\!]_{\mathscr{L}}$ are *traces* and *failures* (cf. [14, 21]).

Definition 1 (Correct Translation up to Semantic Equivalence [10])**.** *A translation $\mathsf{T} : \mathbb{T}_{\mathscr{L}} \to \mathbb{T}_{\mathscr{L}'}$ is correct up to a semantic equivalence $\approx$ on $\mathscr{D}_{\mathscr{L}} \cup \mathscr{D}_{\mathscr{L}'}$ when $[\![E]\!]_{\mathscr{L}} \approx [\![\mathsf{T}(E)]\!]_{\mathscr{L}'}$ for all $E \in \mathbb{T}_{\mathscr{L}}$.*

Operational correspondence allows matching the transitions of two processes, which can help determine the appropriate relation (semantic equivalence) between a term and its translation. Let the operational semantics of $\mathscr{L}$ be defined by the labelled transition system $(\mathbb{T}_{\mathscr{L}}, Act_{\mathscr{L}}, \to_{\mathscr{L}})$, where $Act_{\mathscr{L}}$ is the set of labels and $E \xrightarrow{\lambda}_{\mathscr{L}} E'$ defines transitions with $E, E' \in \mathbb{T}_{\mathscr{L}}$ and $\lambda \in Act_{\mathscr{L}}$.

Definition 2 (Labelled Operational Correspondence, [8, 19])**.** *Let $\mathsf{T} : \mathbb{T}_{\mathscr{L}} \to \mathbb{T}_{\mathscr{L}'}$ be a mapping from the expressions of a language $\mathscr{L}$ to those of a language $\mathscr{L}'$, and let $\mathsf{f} : Act_{\mathscr{L}} \to Act_{\mathscr{L}'}$ be a mapping from the labels of $\mathscr{L}$ to those of $\mathscr{L}'$. A translation $\langle \mathsf{T}, \mathsf{f} \rangle$ is operationally corresponding w.r.t. a semantic equivalence $\approx$ on $\mathscr{D}_{\mathscr{L}} \cup \mathscr{D}_{\mathscr{L}'}$ if it is:*

- *Sound: $\forall E, E' : E \xrightarrow{\lambda}_{\mathscr{L}} E'$ imply that $\exists F : \mathsf{T}(E) \xrightarrow{\mathsf{f}(\lambda)}_{\mathscr{L}'} F$ and $F \approx \mathsf{T}(E')$*
- *Complete: $\forall E, F : \mathsf{T}(E) \xrightarrow{\lambda'}_{\mathscr{L}'} F$ imply that $\exists E' : E \xrightarrow{\lambda}_{\mathscr{L}} E'$ and $F \approx \mathsf{T}(E') \wedge \lambda' = \mathsf{f}(\lambda)$*

The previous two definitions coincide when the semantic equivalence $\approx$ is strong bisimulation (Def.3) and f is the identity.

2.2 CCS, CCSTau

CCS. CCS (Calculus of Communicating Systems) [17, 1] is a process algebra that allows reasoning about concurrent systems. CCS represents programs as *processes*, whose behaviour is determined by

Table 1: SOS rules for CCS

$$Prefix : \alpha.P \xrightarrow{\alpha} P \qquad Sum : \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad Par : \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$Com : \frac{P \xrightarrow{\bar{a}} P' \quad Q \xrightarrow{a} Q'}{P|Q \xrightarrow{\tau} P'|Q'} \qquad Res : \frac{P \xrightarrow{\alpha} P' \quad \alpha \notin B}{P \upharpoonright B \xrightarrow{\alpha} P' \upharpoonright B} \qquad Rec : \frac{P[\mu X.P/X] \xrightarrow{\alpha} P'}{\mu X.P \xrightarrow{\alpha} P'}$$

rules specifying their possible execution steps. The syntax of CCS processes is defined by the following BNF:

$$CCS ::= 0 \,|\, \alpha.P \,|\, P+Q \,|\, P|Q \,|\, P \upharpoonright B \,|\, \mu X.P$$
$$\alpha ::= \tau \,|\, \bar{a} \,|\, a$$

Let $\mathscr{N}$ denote an infinite set of *names*; let $a,b,c,...$ range over $\mathscr{N}$. Let $\overline{\mathscr{N}} = \{\bar{a}|a \in \mathscr{N}\}$ denote the set of conames. Let $\overline{\overline{a}} = a$. Let $\mathscr{L} = \mathscr{N} \cup \overline{\mathscr{N}}$ denote the set of all possible labels. The set of labels of a process P is denoted by $\mathscr{L}(P)$ ([17, Def.2, p52]). Let τ denote the silent or invisible action. Let $Act = \mathscr{N} \cup \overline{\mathscr{N}} \cup \{\tau\}$ denote the set of all possible actions that a process can perform. Let $\alpha, \beta, ..$ range over Act. The SOS semantics of CCS are given in Table 1.

Informally: 0 (or *NIL*) is the process that performs no action. $\alpha.P$ is the process that performs an action α and then behaves like P. $P+Q$ is the process that behaves either like P or like Q. $P|Q$ is the process that executes P and Q in parallel: if both P and Q can engage in an action a then, their execution corresponds to interleaving, e.g. $a.0|a.0 \equiv a.a.0$; if P can engage in action a, Q in the complementary action $\bar{a}$, then, either P and Q interleave on a or they synchronise and the result of synchronisation is the invisible action τ, e.g. $a.0|\bar{a}.0 \equiv a.\bar{a}.0 + \bar{a}.a.0 + \tau.0$. $P \upharpoonright B$ is the process that cannot engage in actions in B except for synchronisation, e.g., $(a.0|\bar{a}.0) \upharpoonright \{a\} \equiv \tau.0$, $(a.0) \upharpoonright \{a\} \equiv 0$. $\mu X.P$ is the process that executes P recursively.

Equivalence based on bisimulations is the preferred choice for discriminating among CCS processes. We will use strong bisimulation to prove the correctness of our translation.

Definition 3 (Strong Bisimulation [21, 17])**.** *A strong bisimulation is a symmetric binary relation $\mathscr{R}$ on processes satisfying the following: $P\mathscr{R}Q$ and $P \xrightarrow{\alpha} P'$ imply that*

$$\exists Q' : Q \xrightarrow{\alpha} Q' \wedge P'\mathscr{R}Q'$$

P is strong bisimilar to Q, written $P \sim Q$, if $P\mathscr{R}Q$ for some strong bisimulation $\mathscr{R}$.

CCSTau. CCSTau [4] extends CCS with visible synchronisations, viz., the result of synchronisation on a pair $(a,\bar{a})$ is the visible action $\tau[a,\bar{a}]$ instead of the visible action τ. This makes it easier to guarantee that when two processes synchronise in CCS(Tau), their CSP translation also synchronises. The syntax of CCSTau processes is defined by the following grammar:

$$\begin{aligned} P,Q,R &::= 0 \mid \alpha.P \mid P+Q \mid P|_{\tau}Q \mid P \restriction B \mid \mu X.P \mid P\backslash_{\tau}B \mid X \\ \alpha &::= \tau \mid \overline{a} \mid a \\ \beta &::= \alpha \mid \tau[a|\overline{a}] \end{aligned}$$

The parallel operator in CCSTau is denoted $|_{\tau}$. CCSTau also defines a hiding operator, denoted $\backslash_{\tau}$, which can hide all actions including $\tau[a,\bar{a}]$ actions. The restriction operator behaves as in CCS, does not apply to $\tau[a,\bar{a}]$ actions. Rules for these operators are given hereafter:

$$Par: \frac{P \xrightarrow{\beta} P'}{P|_{\tau}Q \xrightarrow{\beta} P'|_{\tau}Q} \qquad Com: \frac{P \xrightarrow{\overline{a}} P' \quad Q \xrightarrow{a} Q'}{P|_{\tau}Q \xrightarrow{\tau[\overline{a}|a]} P'|_{\tau}Q'}$$

$$Res: \frac{P \xrightarrow{\beta} P' \quad \beta = \tau[\overline{a}|a] \text{ or } \beta \notin B}{P \restriction B \xrightarrow{\beta} P' \restriction B}$$

$$Hide: \frac{P \xrightarrow{\beta} P' \quad \beta \notin B}{P\backslash_{\tau}B \xrightarrow{\beta} P'\backslash_{\tau}B} \qquad \frac{P \xrightarrow{\beta} P' \quad \beta \in B}{P\backslash_{\tau}B \xrightarrow{\tau} P'\backslash_{\tau}B}$$

All other CCS operators are also CCSTau operators.

CCS-to-CCSTau. Translation function $c2ccs\tau$ [4] translates CCS processes into CCSTau, is correct up to strong bisimulation. For any CCS process P other than CCS-parallel operator, $c2ccs\tau(P) = P$. For the parallel operator: [1]

$$c2ccs\tau(P|Q) \mathrel{\widehat{=}} (c2ccs\tau(P)|_{\tau}c2ccs\tau(Q))\backslash_{\tau}\{\tau[a|\overline{a}] \mid a \in \mathscr{L}(P), \bar{a} \in \mathscr{L}(Q)\} \qquad \text{(c2ccs}\tau\text{-par-def)}$$

2.3 CSP

CSP (Communicating Sequential Processes) [14, 22] is a process algebra that allows reasoning about concurrent systems. In CSP, a (concurrent) program is represented as a *process*, whose behaviour is entirely determined by the possible actions of the program, represented as *events*. The set of events that a process P can possibly perform is denoted by $\mathscr{A}(P)$. Event τ denotes invisible actions, hidden from the environment; event ✓ denotes successful termination, by opposition say to deadlock and abortion. Both denotational and operational semantics have been defined for CSP processes, in terms of traces. The syntax of some CSP processes is defined by the following BNF:

$$\begin{aligned} CSP &::= SKIP \mid STOP \mid \alpha \rightsquigarrow P \mid P \sqcap Q \mid P \,\square\, Q \mid P \underset{B}{\parallel} Q \mid f(P) \mid P \setminus B \mid \mu X.P \\ \alpha &::= a \mid a?x \mid a!m \end{aligned}$$

The SOS semantics of CSP processes are given in Table 2. Informally: *SKIP* is the process that refuses to engage in any event, terminates immediately, and does not diverge. *STOP* is the process that is unable to interact with its environment. $\alpha \rightsquigarrow P$ is the process that first engages in event α then behaves like P. $P \,\square\, Q$ is the process that behaves like P or Q, where the choice is decided by the environment.

[1] The set of labels of a CCS process P, $\mathscr{L}(P)$, corresponds to the set of events $\mathscr{A}(Q)$ for a CSP process Q.

Table 2: SOS rules for CSP [22]

$$Prefix: (a \rightsquigarrow P) \xrightarrow{a} P \qquad Skip:\ SKIP \xrightarrow{\checkmark} STOP$$

$$IntChoice: P_1 \sqcap P_2 \xrightarrow{\tau} P_1 \qquad P_1 \sqcap P_2 \xrightarrow{\tau} P_2$$

$$ExtChoice: \frac{P_1 \xrightarrow{a} P'}{P_1 \Box P_2 \xrightarrow{a} P'} \qquad \frac{P_1 \xrightarrow{\tau} P'}{P_1 \Box P_2 \xrightarrow{\tau} P' \Box P_2}$$

$$IfacePar: \frac{P_1 \xrightarrow{a} P' \quad [a \notin B^{\checkmark}]}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P' \underset{B}{\parallel} P_2} \qquad \frac{P_1 \xrightarrow{a} P_1' \quad P_2 \xrightarrow{a} P_2' \quad [a \in B^{\checkmark}]}{P_1 \underset{B}{\parallel} P_2 \xrightarrow{a} P_1' \underset{B}{\parallel} P_2'}$$

$$Hide: \frac{P \xrightarrow{a} P' \quad [a \notin B]}{P \setminus B \xrightarrow{a} P' \setminus B} \qquad \frac{P \xrightarrow{a} P' \quad [a \in B]}{P \setminus B \xrightarrow{\tau} P' \setminus B}$$

$$FwdRen: \frac{P \xrightarrow{a} P'}{f(P) \xrightarrow{f(a)} f(P')} \qquad \frac{P \xrightarrow{\tau} P'}{f(P) \xrightarrow{\tau} f(P')}$$

$$Rec: \frac{P \xrightarrow{\mu} P' \quad [N = P]}{N \xrightarrow{\mu} P'}$$

$P \underset{B}{\parallel} Q$ behaves like the parallel execution of P and Q where the latter must both synchronise on the set of events B. When $B = \{\}$, we say that P and Q interleave, denoted by $P \,|||\, Q$; if $B = \mathscr{A}(P) \cap \mathscr{A}(Q)$ we also write $P \parallel Q$. $f(P)$ engages in $f(a)$ whenever P engages in a. $P \setminus B$ is the process that engages in all events of P except those in B. $\mu X.P$ is the process that executes P recursively.

Equivalence based on (enriched versions of) traces is the preferred choice for distinguishing CSP processes. We kindly refer the reader to [14, 22] for details.

2.4 CCS-to-CSP Translation

Notation. Given two functions, say f_1 and f_2, $f_1 \circ f_2$ denotes functional composition, viz., $f_1(f_2)$.

In this section, we present *ccs2csp* [4], the translation from CCS-to-CSP, correct up to strong bisimulation.

Definition 4 (*ccs2csp* [4])**.** *Let P be a CCS process. Then:*

$$\begin{aligned}
ccs2csp(P) &\mathrel{\widehat{=}} ai2a \circ (t2csp \circ c2ccs\tau(P))\setminus_{csp}\{a_{ij} | a_{ij} \in \mathscr{A}\big(t2csp(c2ccs\tau(P))\big)\}\\
t2csp(P) &\mathrel{\widehat{=}} (tl \circ conm \circ g^*_{\{\}} \circ ix(P))\setminus_{csp}\{tau\}\\
g^*_S &\mathrel{\widehat{=}} \{\tau \mapsto \tau, a_i \mapsto \{a_i\} \cup \{a_{ij} | \bar{a}_j \in S, i < j\} \cup \{a_{ji} | \bar{a}_j \in S, j < i\}\}\\
conm &\mathrel{\widehat{=}} \{\tau \mapsto \tau, a_i \mapsto a_i, \bar{a}_i \mapsto \bar{a}_i, a_{ij} \mapsto a_{ij}, \bar{a}_{ij} \mapsto a_{ij}\}\\
ai2a &\mathrel{\widehat{=}} \{a_i \mapsto a\}
\end{aligned}$$

where *ix* generates unique indexed prefixes such that a name b maps to a set of indexed names $b_i, i \geq 1$; g^* generates unique double-indexed names for every pair of synchronising names; *conm* renames every synchronising coname into the corresponding name (so they can synchronise in CSP); and *tl* translates CCS operators into corresponding CSP operators. We kindly refer the reader to [4] for details.

Example 1 ([4]). *The translation of CCS binary synchronisation into CSP can be illustrated succinctly as follows:*

$$\begin{aligned}
& ccs2csp(a.0|\bar{a}.0) && \text{(ccs2csp-def)}\\
=& ai2a \circ t2csp\big(c2ccs\tau(a.0|\bar{a}.0)\big)\backslash_{csp}\{a_{ij}|..\} && \text{(c2ccs}\tau\text{-par-def)}\\
=& ai2a \circ t2csp\big((a.0|_T\,\bar{a}.0)\backslash_T\{\tau[a|\bar{a}]\}\big)\backslash_{csp}\{a_{ij}|..\} && \text{(t2csp-def)}\\
=& ai2a \circ tl \circ conm \circ g^*(\{\}, ix\big((a.0|_T\,\bar{a}.0)\backslash_T\{\tau[a|\bar{a}]\}\big))\backslash_{csp}\{tau\}\backslash_{csp}\{a_{ij}|..\} && \text{(ix-def)}\\
=& ai2a \circ tl \circ conm \circ g^*\big((a_1.0|_T\,\bar{a}_2.0)\big)\backslash_{csp}\{tau\}\backslash_{csp}\{a_{ij}|..\} && \text{(gstar-def)}\\
=& ai2a \circ tl \circ conm\big((a_1.0+a_{12}.0)|_T\,(\bar{a}_2.0+\bar{a}_{12}.0)\big)\backslash_{csp}\{tau\}\backslash_{csp}\{a_{12}\} && \text{(conm-def)}\\
=& ai2a \circ tl\big((a_1.0+a_{12}.0)|_T\,(\bar{a}_2.0+a_{12}.0)\big)\backslash_{csp}\{tau,a_{12}\} && \text{(tl-def)}\\
=& ai2a \circ \big((a_1 \,\Box\, a_{12} \rightsquigarrow STOP) \underset{\{a_{12}\}}{\|} (\bar{a}_2 \,\Box\, a_{12} \rightsquigarrow STOP)\big)\backslash_{csp}\{tau,a_{12}\} && \text{(ai2a-def)}\\
=& \big((a \,\Box\, a_{12} \rightsquigarrow STOP) \underset{\{a_{12}\}}{\|} (\bar{a} \,\Box\, a_{12} \rightsquigarrow STOP)\big)\backslash_{csp}\{tau,a_{12}\}
\end{aligned}$$

In CCS, a name can be used both for interleaving and for synchronisation. This is reflected in the translation above by generating indexed names a_1 and $\bar{a}_2$ for interleaving; then for the synchronisation pair $(a_1,\bar{a}_2)$, a unique synchronisation name a_{12} is generated. More generally, there will be as many a_{ij} synchronisation names as there are of synchronisation on name a.

In the next section, we extend CSP with m-among-n synchronisation, then derive 2-among-n (binary) synchronisation. In the end, we will be able to translate CCS binary synchronisation into CSP binary synchronisation.

3 CSP plus m-among-n Synchronisation

Multiway synchronisation in CSP is *maximal*, viz., all processes that can synchronise *must* synchronise. This is also called the *maximal (or n-ary) coordination* paradigm ([9]): if n processes are ready to synchronise on event a, then all n processes must synchronise together. Can we generalise this to allow only m-among-n ($2 \leq m \leq n$) processes to synchronise instead? If the answer is yes then binary synchronisation can be defined as 2-among-n coordination and n-ary synchronisation as n-among-n coordination. Garavel and Sighireanu [9] define m/n coordination for the language E-LOTOS.

First, let us generalise CSP (n-ary) interface parallel operator ([22]).

$$IndxIfacePar: \frac{P_j \xrightarrow{a} P' \quad [a \notin B^{\checkmark}, k \neq j]}{\underset{B}{\|} P_i \xrightarrow{a} (\underset{B}{\|} P_k) \underset{B}{\|} P'} \qquad \frac{P_1 \xrightarrow{a} P_1' \quad \ldots \quad P_n \xrightarrow{a} P_n' \quad [a \in B^{\checkmark}]}{\underset{B}{\|} P_i \xrightarrow{a} \underset{B}{\|} P_i'}$$

Definition 5 ($a\#m$ clause [9]). *Let $I = \{1,..,n\}, n \in \mathbf{N}, n \geq 2$. Let m be a natural number in the range $2,..,n$ associated to an a-event such that a clause a#m denotes that m processes are allowed to synchronise on event a at once. Each clause #m is optional: if omitted, m has default value n.*

The rules for m/n indexed interface paralell composition are given hereafter.[2]

$$M/N\text{–}IndxIfacePar : \frac{P_j \xrightarrow{a} P' \quad [a\#m \notin B^{\checkmark} \times \{2,..,n\}, k \neq j]}{\underset{B\times\{2,..,n\}}{\|} P_i \xrightarrow{a} (\underset{B\times\{2,..,n\}}{\|} P_k) \underset{B\times\{2,..,n\}}{\|} P'}$$

$$\frac{P_1 \xrightarrow{a} P_1' \ldots P_n \xrightarrow{a} P_n' \quad [a\#m \in B^{\checkmark} \times \{2,..,n\}, j \in J, k \neq j]}{\underset{B\times\{2,..,n\}}{\|} P_i \xrightarrow{a} \underset{\{J\subseteq I | card(J)=m\}}{\sqcap} \left((\underset{B\times\{2,..,n\}}{\|} P_k) \underset{B\times\{2,..,n\}}{\|} (\underset{B\times\{2,..,n\}}{\|} P_j') \right)}$$

We can then derive binary-only synchronisation by imposing that *every* event in set B allows 2(only)-among-n processes to synchronise.

$$2/N\text{–}IndxIfacePar : \frac{P_1 \xrightarrow{a} P_1' \ldots P_n \xrightarrow{a} P_n' \quad [a\#2 \in A^{\checkmark} \times \{2\}, j \in J, k \neq j]}{\underset{B\times\{2\}}{\|} P_i \xrightarrow{a} \underset{\{J\subseteq I | card(J)=2\}}{\sqcap} \left((\underset{B\times\{2\}}{\|} P_k) \underset{B\times\{2\}}{\|} (\underset{B\times\{2\}}{\|} P_j') \right)}$$

Similarly, we derive n-ary-only synchronisation by imposing that *every* event in set B allows n-among-n processes to synchronise. We easily verify that rules N/N-IndxIfacePar and IndxIfacePar (synchronisation) are the same.

$$N/N\text{–}IndxIfacePar : \frac{P_1 \xrightarrow{a} P_1' \ldots P_n \xrightarrow{a} P_n' \quad [a\#n \in B^{\checkmark} \times \{n\}]}{\underset{B\times\{n\}}{\|} P_i \xrightarrow{a} \underset{B\times\{n\}}{\|} P_i'}$$

Correctness of M/N-IndxIfacePar rule. Let us call CSPmn the extension of CSP with m-among-n synchronisation. We argue here that CSPmn is a conservative extension of CSP, i.e., CSPmn preserves the axioms of CSP.

The proof method is suggested to us by function g^* [4]. For binary synchronisation, select process pairs that must synchronise and assign them a unique synchronisation index. E.g.,

$$a \underset{a\#2}{\|} a \underset{a\#2}{\|} a \quad \text{maps to} \quad (a_{12} \,\square\, a_{13}) \,\|\, (a_{12} \,\square\, a_{23}) \,\|\, (a_{13} \,\square\, a_{23})$$

Then, for m processes to synchronise among n, generate a unique index for all possible combinations of m processes among n, e.g.,

$$a \underset{a\#2}{\|} a \underset{a\#2}{\|} a \underset{a\#2}{\|} a \quad \text{maps to} \quad (a_{12} \,\square\, a_{13} \,\square\, a_{14}) \,\|\, (a_{12} \,\square\, a_{23} \,\square\, a_{24}) \,\|\, (a_{13} \,\square\, a_{23} \,\square\, a_{34}) \,\| \\ (a_{14} \,\square\, a_{24} \,\square\, a_{34})$$

$$a \underset{a\#3}{\|} a \underset{a\#3}{\|} a \underset{a\#3}{\|} a \quad \text{maps to} \quad (a_{123} \,\square\, a_{124} \,\square\, a_{134}) \,\|\, (a_{123} \,\square\, a_{124} \,\square\, a_{234}) \,\|\, (a_{123} \,\square\, a_{134} \,\square\, a_{234}) \,\| \\ (a_{124} \,\square\, a_{134} \,\square\, a_{234})$$

$$a \underset{a\#4}{\|} a \underset{a\#4}{\|} a \underset{a\#4}{\|} a \quad \text{maps to} \quad a_{1234} \,\|\, a_{1234} \,\|\, a_{1234} \,\|\, a_{1234}$$

[2]The rules in [9] use a different rule format than CSP rules: they use predicates.

From what precedes, there exists a relational renaming, say G, such that

$$\mathop{\|}_{a\#m,j} P_j \sim \mathop{\|}_{G(a),j} P_j[G(a)/a]$$

We can thus define (CSPmn parallel operator) $\mathop{\|}_{a\#m}$ in terms of both (CSP parallel operator) $\mathop{\|}_{a}$ and (CSP relational renaming) $G(a)$. Therefore, CSPmn is a conservative extension of CSP, viz., preserves CSP axioms (cf. Appendix A for a full proof).

4 CCSTau Transformations

$$\boxed{CCS} \xrightarrow{c2ccs\tau} \boxed{CCSTau} \xrightarrow{g^2}\xrightarrow{conm} \boxed{CCSTau} \xrightarrow{tl_3}\xrightarrow{\backslash\{tau\}}\xrightarrow{\backslash\{a_S\}} \boxed{CSPmn}$$

Figure 1: CCS-to-CSPmn Translation workflow

The different stages of our translation are shown in Fig. 1.

Pairwise vs. Multiway Synchronisation Recall, a CCSTau name has both interleaving and synchronisation semantics. We hence have to generate two distinct CSP events for a single CCS name. Also, it is possible to hide $\tau[a|\bar{a}]$ synchronisation actions in CCSTau (typically, to obtain a CCS process—cf. Def.c2ccsτ-par-def). Then, it will be convenient to ignore them. Let g^2 define the function that generates a synchronisation name for any CCS name.

Definition 6 ($g^2(\alpha)$).

$$g^2(S,\tau) \mathrel{\widehat{=}} \tau \qquad g^2(S,a) \mathrel{\widehat{=}} \{a\} \cup \{a_S \mid \bar{a} \in S\}$$
$$g^2(S,\tau[a|\bar{a}]) \mathrel{\widehat{=}} \{\tau[a,\bar{a}]\} \qquad g^2(S,B) \mathrel{\widehat{=}} \{g^2(S,a) \mid a \in B, \bar{a} \in S\}$$

Given a set of names generated by g^2, a-names denote interleaving, whilst a_S-names denote synchronisation. The application of g^2 to processes is given hereafter.

Definition 7 ($g^2(P)$). *Let P be a CCS process. Let $g^2(P) \mathrel{\widehat{=}} g^2(\{\},P)$.*

$$g^2(S,0) \mathrel{\widehat{=}} 0 \qquad g^2(S,P \upharpoonright B) \mathrel{\widehat{=}} g^2(S,P) \upharpoonright g^2(S,B)$$
$$g^2(S,\alpha.P) \mathrel{\widehat{=}} \sum_{b \in g^2(S,\alpha)} b.g^2(S,P) \qquad g^2(S,P\backslash_{\tau}B) \mathrel{\widehat{=}} g^2(S,P)\backslash_{\tau} g^2(S\cup B,B)$$
$$g^2(S,P+Q) \mathrel{\widehat{=}} g^2(S,P)+g^2(S,Q) \qquad g^2(S,\mu X.P) \mathrel{\widehat{=}} \mu X.g^2(S,P)$$
$$g^2(S,P|_{\tau}Q) \mathrel{\widehat{=}} g^2(S\cup\mathscr{A}(Q),P)|_{\tau}\, g^2(S\cup\mathscr{A}(P),Q) \qquad g^2(S,X) \mathrel{\widehat{=}} X$$

Note the difference between restriction and hiding. Names $g^2(S,B)$ are generated between a process and its environment. Only those names will be restricted, understood that (restricted) B names cannot interact with their environment. Internal synchronisation on B names, however, will not be restricted (until later in CSP). In contrast, for hiding, internal synchronisation on B must be hidden as well, hence we hide names $g^2(S\cup B,B)$ instead.

Example 2. *Let us illustrate the translation of restriction.*

$$
\begin{aligned}
& g^2\big(\{\},(a.0|_T\bar{a}.0)\restriction\{a\}\big) && \text{(g2-def)}\\
&= g^2\big(\{\},a.0|_T\bar{a}.0\big)\restriction g^2(\{\},\{a\}) && \text{(g2-res-def)}\\
&= \big(g^2(\{\bar{a}\},a.0)|_T g^2(\{a\},\bar{a}.0)\big)\restriction\{a\} && \text{(g2-par-def)}\\
&= \big((a.0+a_S.0)|_T(\bar{a}.0+\bar{a}_S.0)\big)\restriction\{a\}
\end{aligned}
$$

Contrast with hiding, which hides both a and a_S. (Recall $\backslash_T\{a\} = \backslash_T\{a,\bar{a}\}$.)

$$
\begin{aligned}
& g^2\big(\{\},(a.0|_T\bar{a}.0)\backslash_T\{a\}\big) && \text{(hide-def)}\\
&= g^2\big(\{\},(a.0|_T\bar{a}.0)\backslash_T\{a,\bar{a}\}\big) && \text{(g2-def)}\\
&= g^2\big(\{\},a.0|_T\bar{a}.0\big)\backslash_T g^2(\{a,\bar{a}\},\{a,\bar{a}\}) && \text{(g2-hide-def)}\\
&= \big(g^2(\{\bar{a}\},a.0)|_T g^2(\{a\},\bar{a}.0)\big)\backslash_T\{a,\bar{a},a_S,\bar{a}_S\} && \text{(g2-par-def, hide-def)}\\
&= \big((a.0+a_S.0)|_T(\bar{a}.0+\bar{a}_S.0)\big)\backslash_T\{a,a_S\}
\end{aligned}
$$

Finally, consider hiding the synchronisation action $\tau[a|\bar{a}]$, this turns out to be vacuous.

$$
\begin{aligned}
& g^2\big(\{\},(a.0|_T\bar{a}.0)\backslash_T\{\tau[a|\bar{a}]\}\big) && \text{(g2-def)}\\
&= g^2\big(\{\},a.0|_T\bar{a}.0\big)\backslash_T g^2(\{a\},\{\tau[a|\bar{a}]\}) && \text{(g2-hide-def)}\\
&= \big(g^2(\{\bar{a}\},a.0)|_T g^2(\{a\},\bar{a}.0)\big)\backslash_T\{\tau[a|\bar{a}]\} && \text{(g2-par-def)}\\
&= \big((a.0+a_S.0)|_T(\bar{a}.0+\bar{a}_S.0)\big)\backslash_T\{\tau[a|\bar{a}]\}
\end{aligned}
$$

Parallel Composition. In CSP, synchronisation pairs $(a_S,\bar{a}_S)$ will not be able to synchronise. We hence update the coname function to translate conames into names.

Definition 8 (*conm*). *conm* $\widehat{=} \{\tau\mapsto\tau, a\mapsto a, \bar{a}\mapsto\bar{a}, a_S\mapsto a_S, \bar{a}_S\mapsto a_S\}$.

Link CCSTau-to-CSPmn In [4], function *tl* translates CCSTau operators into CSP operators, without consideration for differences in their respective alphabets. Hereafter, we define tl_3, to map CCS binary synchronisation into CSPmn binary synchronisation. All other operators are translated as before, viz., $tl_3(P) = tl(P)$ for all process expressions other than parallel composition. Additionally, because of the possibility to hide $\tau[a,\bar{a}]$ synchronisation actions in CCSTau, we translate CCSTau hiding operator also, translation which was not needed for tl.

Definition 9 (tl_3). *Let tau be a CSP event that cannot synchronise.*

$$
\begin{aligned}
tl_3(0) &\mathrel{\widehat{=}} STOP & tl_3(P+Q) &\mathrel{\widehat{=}} tl_3(P)\,\Box\,tl_3(Q)\\
tl_3(\tau.P) &\mathrel{\widehat{=}} tau\rightsquigarrow tl_3(P) & tl_3(P|_TQ) &\mathrel{\widehat{=}} tl_3(P)\underset{\{a\#2|a\in\mathscr{A}(P)\cap\mathscr{A}(Q)\}}{\|}tl_3(Q)\\
tl_3(a.P) &\mathrel{\widehat{=}} a\rightsquigarrow tl_3(P) & tl_3(\mu X.P) &\mathrel{\widehat{=}} \mu X.tl_3(P)\\
tl_3(P\restriction B) &\mathrel{\widehat{=}} tl_3(P)\restriction_{csp} B & tl_3(X) &\mathrel{\widehat{=}} X\\
tl_3(P\backslash_T B) &\mathrel{\widehat{=}} tl_3(P)\backslash_{csp} B
\end{aligned}
$$

Note that $tl_3(P\backslash_T\{\tau[a|\bar{a}]\}) = tl_3(P)\backslash_{csp}\{\tau[a|\bar{a}]\} = tl_3(P)$, since $\tau[a|\bar{a}]$ actions do not occur in the translated term, $tl_3(P)$. This is necessary, as illustrated subsequently.

Example 3. *CCS process $a\,|\,\bar{a}\,|\,a$, by c2ccsτ, corresponds to CCSTau process*

$$\big((a|_T\bar{a})\backslash_T\{\tau[a|\bar{a}]\}|_T a\big)\backslash_T\{\tau[a,\bar{a}]\}$$

By g^2, this becomes process

$$\big(((a+a_S)|_T(\bar{a}+\bar{a}_S))\backslash_T\{\tau[a|\bar{a}]\}|_T(a+a_S)\big)\backslash_T\{\tau[a|\bar{a}]\}$$

Then, by tl_3, it becomes

$$\begin{aligned}&\big(((a\,\Box\,a_S)\underset{a_S\#2}{\|}(\bar{a}\,\Box\,\bar{a}_S))\backslash_{csp}\{\tau[a|\bar{a}]\}\underset{a_S\#2}{\|}(a\,\Box\,a_S)\big)\backslash_{csp}\{\tau[a|\bar{a}]\}\\ =\;&(a\,\Box\,a_S)\underset{a_S\#2}{\|}(\bar{a}\,\Box\,\bar{a}_S)\underset{a_S\#2}{\|}(a\,\Box\,a_S)\end{aligned}$$

Thanks to $\backslash_{csp}\{\tau[a,\bar{a}]\}$ being vacuous, there will be two possible synchronisations on a_S, corresponding to the original CCS behaviour.

The following abbreviation translates CCSTau into CSPmn.

Definition 10 (CCSTau to CSPmn)**.** *Let P be a CCSTau process. Then:*

$$t2csp_3(P)\;\widehat{=}\;(tl_3\circ conm\circ g^2(P))\backslash_{csp}\{tau\}$$

Link CCS-to-CSPmn. We obtain the translation from CCS to CSP by translating CCS into CCSTau first, using $c2ccs\tau$ (Def.c2ccsτ-par-def), then translating CCSTau into CSPmn, using $t2csp_3$ (Def.10), and finally hiding every a_S synchronisation event.

Definition 11 (CCS to CSPmn)**.** *Let P denote a CCS process. Then:*

$$ccs2csp_3(P)\;\widehat{=}\;(t2csp_3\circ c2ccs\tau(P))\backslash_{csp}\{a_S|a_S\in\mathscr{A}(t2csp_3\circ c2ccs\tau(P))\}$$

Example 4. *The translation of CCS binary synchronisation into CSPmn can be illustrated succinctly as follows:*

$$\begin{aligned}&ccs2csp_3(a.0|\bar{a}.0) && \text{(ccs2csp3-def.11)}\\ =\;&\big(t2csp_3\circ c2ccs\tau(a.0|\bar{a}.0)\big)\backslash_{csp}\{a_S|..\} && \text{(c2ccs}\tau\text{-par-def)}\\ =\;&t2csp_3\big((a.0|_T\bar{a}.0)\backslash_T\{\tau[a|\bar{a}]\}\big)\backslash_{csp}\{a_S\} && \text{(t2csp3-def.10)}\\ =\;&tl_3\circ conm\circ g^2(\{\},(a.0|_T\bar{a}.0)\backslash_T\{\tau[a|\bar{a}]\})\backslash_{csp}\{tau\}\backslash_{csp}\{a_S\} && \text{(g2-def.6)}\\ =\;&tl_3\circ conm\Big(\big((a.0+a_S.0)|_T(\bar{a}.0+\bar{a}_S.0)\big)\backslash_T\{\tau[a|\bar{a}]\}\Big)\backslash_{csp}\{tau\}\backslash_{csp}\{a_S\} && \text{(conm-def.8)}\\ =\;&tl_3\Big(\big((a.0+a_S.0)|_T(\bar{a}.0+a_S.0)\big)\backslash_T\{\tau[a|\bar{a}]\}\Big)\backslash_{csp}\{tau\}\backslash_{csp}\{a_S\} && \text{(tl3-def.9, CSP)}\\ =\;&\big((a\,\Box\,a_S\rightsquigarrow STOP)\underset{\{a_S\#2\}}{\|}(\bar{a}\,\Box\,a_S\rightsquigarrow STOP)\big)\backslash_{csp}\{tau,a_S\}\end{aligned}$$

Example 5. *The translation of recursion with nested parallel can be illustrated as follows. Let $P\,\widehat{=}\,\mu X.(a|\bar{a}.X)$ (or equiv. $P\,\widehat{=}\,a.0\,|\,\bar{a}.P$) be a CCS process. Then, $ix(P)=a_1\,|\,a_2.ix_{\{3..\}}(P)$, where $ix_{\{3..\}}$ denotes that indexing excludes indices 1 and 2. Let us unfold P one step, then:*

$$\begin{aligned}P&=a\,|\,\bar{a}.(a\,|\,\bar{a}.P)\\ ix(P)&=a_1\,|\,\bar{a}_2.(a_3\,|\,\bar{a}_4.ix_{\{5..\}}(P))\end{aligned}$$

The synchronisation pairs are thus $(a_1,\bar{a}_2),(a_1,\bar{a}_4),..,$ *that is, the set* $\{(a_1,\bar{a}_{2k})|k\geq 1\}$. *Then:*

$$g^*(P) = (a_1 + \sum_{k\geq 1} a_{1*2k})\,|\,(\bar{a}_2 + \bar{a}_{12}).g^*(P)$$

We will not be able to generate all the a_{1*2k} *indices since recursion is unbounded. For closure, we give the* temptative *translation of P with ccs2csp:* [3]

$$ccs2csp(P) = \big((a \,\Box\, \Box_{k\geq 1} a_{1*2k}) \mathop{\|}_{\{a_{1*2k}|k\geq 1\}} (\bar{a}_2 \,\Box\, a_{12}) \rightsquigarrow ai2a \circ t2csp \circ c2ccs\tau(P)\big)\backslash_{csp}\{a_{ij}|..\}$$

In contrast, let us define $ccs2csp_3(P)$. *Then:*

$$\begin{aligned} g^2(P) &= (a + a_S)\,|\,(\bar{a} + \bar{a}_S).g^2(P) \\ &= (a + a_S)\,|\,(\bar{a} + \bar{a}_S).\big((a + a_S)\,|\,(\bar{a} + \bar{a}_S).g^2(P)\big) \end{aligned}$$

We can unfold P multiple times, we only ever generate a single name for synchronisation. Then:

$$ccs2csp_3(P) = \big((a \,\Box\, a_S) \mathop{\|}_{a_S\#2} (\bar{a} \,\Box\, a_S) \rightsquigarrow t2csp_3 \circ c2ccs\tau(P)\big)\backslash_{csp}\{a_S\}$$

5 Gstar Implements 2/n-Synchronisation

We discuss here the relation between g^*-renaming ([4]) and m-among-n synchronisation (§3) approaches.

Recall, function g^* (Def.4, [4]) computes for a CCSTau process P all the substitute names corresponding to distinct synchronisation possibilities of P with its environment, plus interleaving. We have proposed an alternative solution based on extending CSP with 2-among-n synchronisation, derived from first extending CSP with m-among-n synchronisation. Whilst this second solution is more elegant than the gstar-renaming one, the problem of its *immediate* implementability in a tool like FDR has been raised.

Given the current version of FDR, m-among-n synchronisation cannot be implemented directly. We remark, however, that one effect of m-among-n synchronisation is to select, using non-deterministic choice, the m processes that are allowed to synchronise; effect which is precisely what function g^* achieves through renaming. We discuss how to relate both results.

Let us refer by CSPgstar the CSP process expressions resulting from translation $ccs2csp$. We can translate CSPgstar expressions into CSPmn expressions as follows.

Definition 12 (gstar2m/n)**.** *Let* a_{ij} *be an* g^* *name,* a_S *an* g^2 *name. Then:* $g^*2g^2 \mathrel{\widehat{=}} \{\tau \mapsto \tau, a_{ij} \mapsto a_S\}$

While g^*2g^2 is a simple renaming function, its application to CSP processes is modified specifically for the parallel operator such as to map $\mathop{\|}_{\{a_{ij}\}}$ unto $\mathop{\|}_{\{a_S\#2\}}$ (instead of $\mathop{\|}_{\{a_S\}}$).

Definition 13. *Let P be a CSP process.*

$$g^*2g^2(STOP) \mathrel{\widehat{=}} STOP \qquad g^*2g^2(P \mathop{\|}_{\{a_{ij}\}} Q) \mathrel{\widehat{=}} g^*2g^2(P) \mathop{\|}_{\{a_S\#2\}} g^*2g^2(Q)$$

$$g^*2g^2(\alpha \rightsquigarrow P) \mathrel{\widehat{=}} g^*2g^2(\alpha) \rightsquigarrow g^*2g^2(P) \qquad g^*2g^2(P\backslash_{csp}B) \mathrel{\widehat{=}} g^*2g^2(P)\backslash_{csp}g^*2g^2(B)$$

$$g^*2g^2(P \sqcap Q) \mathrel{\widehat{=}} g^*2g^2(P) \sqcap g^*2g^2(Q) \qquad g^*2g^2(P \,\Box\, Q) \mathrel{\widehat{=}} g^*2g^2(P) \,\Box\, g^*2g^2(Q)$$

[3] We are lucky that we can tell in advance what the synchronisation indices are, because process P is a simple case.

Theorem 1. *Let P be a CCS processes. Then:* $g^{*}2g^{2}\circ ccs2csp(P)=ccs2csp_{3}(P)$.

Proof. By induction on the structure of CCS processes. When P does not mention CCS parallel, the proof is straightforward. We develop the proof for the parallel case only. We have:

$$\begin{aligned}
& g^{*}2g^{2}\circ ccs2csp(P\,|\,Q) && \text{(ccs2csp-def.4)}\\
=\; & g^{*}2g^{2}\circ ai2a\circ(t2csp(P)\underset{\{a_{ij}\}}{\parallel} t2csp(Q))\backslash_{csp}\{tau,a_{ij}\} && \text{(CSP hide law)}\\
=\; & g^{*}2g^{2}\circ(ai2a\circ t2csp(P)\backslash_{csp}\{tau\}\underset{\{a_{ij}\}}{\parallel} ai2a\circ t2csp(Q)\backslash_{csp}\{tau\})\backslash_{csp}\{a_{ij}\} && \text{(g*2g2-def.12)}\\
=\; & \big(g^{*}2g^{2}\circ ai2a\circ t2csp(P)\backslash_{csp}\{tau\}\underset{\{a_{S}\#2\}}{\parallel} g^{*}2g^{2}\circ ai2a\circ t2csp(P)\backslash_{csp}\{tau\}\big)\backslash_{csp}\{a_{S}\}\\
& && \text{(Induction Hyp., ccs2csp3-def.11)}\\
=\; & ccs2csp_{3}(P\,|\,Q)
\end{aligned}$$

□

We say that g^{*} implements 2-among-n synchronisation.

6 Conclusion and Future Work

[4] proposes a translation of CCS into CSP based on the g^{*}-renaming approach whereby if two processes can synchronise on an action b, then a name unique to these two processes, say b_{ij}, is generated to substitute b. Thus, if more than two processes could initially synchronise on b, only two processes will ever be able to synchronise on b_{ij} after application of g^{*}.

In this paper, we propose an alternative, the m-among-n synchronisation approach, whereby we first extend CSP multiway synchronisation (or n-among-n) to m-among-n synchronisation (extension called CSPmn), from which we derive 2-among-n or binary synchronisation for CSP processes. We then translate CCS binary synchronisation into CSPmn binary synchronisation. Unlike the g^{*}-renaming approach, the m/n-approach is not limited by parallel under recursion since we can generate a single synchronisation name, say a_{S}, independently of the number of processes meant to synchronise on a_{S}.

We have also shown that CSPmn is a conservative extension of CSP (viz., preserves CSP axioms) by defining (CSPmn) m-among-n synchronisation in terms of both (CSP) multiway (or n-among-n) synchronisation and relational renaming.

We are tempted to affirm that m-among-n synchronisation is more expressive than both 2-among-n and n-among-n synchronisation. However, Hatzel et al. [11] propose an encoding from CSP into CCS whereby they encode CSP multiway synchronisation based on CCS binary synchronisation. Our work suggests that in trying to translate CSP into CCS, it would be easier to extend CCS with multiway synchronisation, as we have done here for CSP. Other works on the translation from CSP into CCS include [2], [3], [12], and [10].

We have proposed here the translation from CCS to CSP only. The main reason for this is our interest in using CSP tools such as FDR for reasoning about CCS processes. With regard to this concern, the g^{*}-renaming approach is more readily implementable than the m/n-approach. The latter would require extending FDR with semantics (viz. rules) for m-among-n synchronisation. Alternatively, m-among-n synchronisation can be implemented using function $g^{*}_{\#}$ (Def.15), however, with the limitation on parallel under recursion similar to g^{*} (cf. [4]). Mechanising our results in Isabelle theorem prover is also to be explored in the future.

Acknowledgments. This work was conducted with the financial support of the Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern and Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (`www.lero.ie`). For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

[1] L. Aceto, K.A. Larsen, A. Ingolfsdottir, *An Introduction to Milner's CCS*, 2005. [Last Accessed 19.Aug.2022: http://twiki.di.uniroma1.it/pub/MFS/WebHome/intro2ccs.pdf]

[2] E. Astesiano, E. Zucca, *Semantics of CSP via Translation into CCS*, Mathematical Foundations of Computer Science (MFCS), vol. 118, pp.172-182, 1981. doi:10.1007/3-540-10856-4_83

[3] S.D. Brookes, *On the Relationship of CCS and CSP*, Automata, Languages and Programming, LNCS, vol. 154, pp. 83-96, 1983. doi:10.1007/BFb0036899

[4] G. Ekembe Ngondi, V. Koutavas, A. Butterfield, *Translation of CCS into CSP, Correct up to Strong Bisimulation*, SEFM'21, pp. 243-261, LNCS vol. 13085, Dec. 2021. doi:10.1007/978-3-030-92124-8_14

[5] G. Ekembe Ngondi, *Denotational Semantics of Channel Mobility in UTP-CSP*, Formal Aspects of Computing Journal, May 2021. doi:10.1007/s00165-021-00546-3

[6] G. Ekembe Ngondi, *Denotational Semantics of Mobility in UTP (Unifying Theories of Programming)*, PhD Thesis, University of York, 2016.

[7] *FDR Documentation*, [Last Accessed 19.Aug.2022] `https://cocotec.io/fdr/manual/`

[8] Y. Fu, H. Lu, *On the Expressiveness of Interaction*, TCS, vol. 411, pp. 1387-1451, 2010. doi:10.1016/j.tcs.2009.11.011

[9] H. Garavel, M. Sighireanu, *A Graphical Parallel Composition Operator for Process Algebras*, IFIPAICT, vol. 28, pp. 185-202, 1999. doi:10.1007/978-0-387-35578-8_11

[10] R. van Glabbeek, *Musings on Encodings and Expressiveness*, EPTCS vol. 89, pp. 81–98, 2012. doi:10.4204/EPTCS.89.7

[11] M. Hatzel, C. Wagner, K. Peters, Uwe Nestmann, *Encoding CSP into CCS*, EXPRESS/SOS Workshop, EPTCS vol. 190, pp. 61-75, 2015. doi:10.4204/EPTCS.190.5

[12] M. Hennessy, Wei Li, G.D. Plotkin, *A First Attempt at Translating CSP into CCS*, International Conference on Distributed Computing (ICDC), pp105–115, 1981.

[13] J. He, C.A.R. Hoare, *CSP is a retract of CCS*, TCS, vol. 411, pp. 1311-1337, Elsevier, 2010. doi:10.1016/j.tcs.2009.12.012

[14] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall, 1985.

[15] T. Hoare, Jifeng He, *Unifying Theories of Programming*, Prentice-Hall, 1998.

[16] R. Milner, *Communicating and Mobile Systems: the Pi-calculus*, Cambridge University Press, 1999.

[17] R. Milner, *Communication and Concurrency*, Prentice-Hall, 1989.

[18] R. De Nicola, M. Hennessy, *CCS without tau's*, TAPSOFT'87, LNCS, vol. 249, pp. 138-152, 1987. doi:10.1007/3-540-17660-8_53

[19] K. Peters, *Comparing Process Calculi Using Encodings*, EXPRESS/SOS Workshop, EPTCS, vol. 300, pp. 19–38, 2019. doi:10.4204/EPTCS.300.2

[20] A.W. Roscoe, *The Theory and Practice of Concurrency*, Prentice-Hall, 1998.

[21] D. Sangiorgi, *Introduction to Bisimulation and Coinduction*, Cambridge University Press, 2012.

[22] S. Schneider, *Concurrent and Real-Time Systems - The CSP Approach*, John Wiley& Sons, Ltd, 2000.

[23] *Haskell Prototype Automation of CCS-to-CSP translation*, GitHub Repository, [Last Accessed 12.Oct.2020] https://github.com/andrewbutterfield/ccs2csp

A Proof that CSPmn is a Conservative Extension

In order to prove that CSPmn is conservative, we need to define some auxillary functions. First, we uniquely index the prefixes of CSP processes.

Property 1. *Let P be a CSP process.*

$$ix(STOP) = STOP$$
$$ix(a \rightsquigarrow P) = a_i \rightsquigarrow ix_{-i}(P)$$
$$ix(P \sqcap Q) = ix_1(P) \sqcap ix_2(Q)$$
$$ix(P \Box Q) = ix_1(P) \Box ix_2(Q)$$
$$ix(P \underset{a\#m}{\parallel} Q) = ix_1(P) \underset{B}{\parallel} ix_2(Q)$$
$$B \mathrel{\widehat{=}} \{a_i\#m | a_i \in \mathscr{A}(ix_1(P)) \cup \mathscr{A}(ix_2(Q))\}$$
$$ix(P \backslash_{csp} \{a\}) = ix(P) \backslash_{csp} \{a_i | a_i \in \mathscr{A}(ix(P))\}$$
$$ix(\mu X.P) = \mu X.ix(P)$$
$$ix(X) = X$$

where ix_{-i} is some indexing scheme which does not assign the i-index, and ix_1, ix_2 are indexing schemes that assign disjoint indices.

Then, using *ix*-generated indices we generate unique synchronisation indices. Given a set $\{a_i\}$ of parallel prefixes and a number m of processes meant to synchronise together, $g^*_{a_i\#m}$ generates a unique synchronisation index $a_{i_1..i_m}$.

Definition 14. *Let S, B denote sets of indexed events.*

$$g^*_{a_{i_1}\#m}(S, a_{i_1}) \mathrel{\widehat{=}} \{a_{i_1..i_m} \,|\, i_1 < .. < i_m, \{a_{i_k} \,|\, 1 < k \leq m\} \subseteq S\} \cup \{a_{i_m..i_1} \,|\, i_m < .. < i_1, \{a_{i_k} \,|\, 1 < k \leq m\} \subseteq S\}$$

$$g^*_{\{a_k\#m_k | k \in \mathbf{N}\}}(S, a_i) \mathrel{\widehat{=}} \begin{cases} a_i & a_i \notin \{a_k \,|\, k \in \mathbf{N}\} \\ g^*_{a_i\#m_i}(S, a_i) & \end{cases}$$

Although $g^*_{a\#m}$ denotes relational renaming, we overload its application to processes such that it translates $\underset{a\#m}{\parallel}$ into $\underset{a}{\parallel}$. This corresponds to the following.

Definition 15. *Let P be an ix-indexed CSP processes. Let S be a set of ix-indexed events. Let a#m denote the set $\{a_k\#m_k \,|\, k \in \mathbf{N}\}$, b#n the set $\{b_j\#n_j \,|\, j \in \mathbf{N}\}$. Let $g^*_{a\#m}(P) \mathrel{\widehat{=}} g^*_{a\#m}(\{\}, P)$.*

$$g^*_{a\#m}(S, STOP) \mathrel{\widehat{=}} STOP$$
$$g^*_{a\#m}(S, a \rightsquigarrow P) \mathrel{\widehat{=}} \sum_{b \in g^*_{a\#m}(a)} b \rightsquigarrow g^*_{a\#m}(S, P)$$
$$g^*_{a\#m}(S, P \sqcap Q) \mathrel{\widehat{=}} g^*_{a\#m}(S, P) \sqcap g^*_{a\#m}(S, Q)$$
$$g^*_{a\#m}(S, P \Box Q) \mathrel{\widehat{=}} g^*_{a\#m}(S, P) \Box g^*_{a\#m}(S, Q)$$
$$g^*_{a\#m}(S, \mu X.P) \mathrel{\widehat{=}} \mu X.g^*_{a\#m}(S, P)$$
$$g^*_{a\#m}(S, X) \mathrel{\widehat{=}} X$$
$$g^*_{a\#m}(S, P \backslash_{csp} \{a\}) \mathrel{\widehat{=}} g^*_{a\#m}(S, P) \backslash_{csp} g^*_{a\#m}(S, a)$$
$$g^*_{a\#m}(S, P \underset{b\#n}{\parallel} Q) \mathrel{\widehat{=}} g^*_{a\#m \cup b\#n}(S \cup \mathscr{A}(Q), P) \underset{B}{\parallel} g^*_{a\#m \cup b\#n}(S \cup \mathscr{A}(P), Q)$$
$$B \mathrel{\widehat{=}} \bigcup \{g^*_{a\#m \cup b\#n}(S \cup \mathscr{A}(Q), b_j) | b_j \in \mathscr{A}(P)\} \cup \bigcup \{g^*_{a\#m \cup b\#n}(S \cup \mathscr{A}(P), b_j) | b_j \in \mathscr{A}(Q)\}$$

When $a\#m$ denotes the empty set, we write $g^*_{\#}$ for the corresponding function $g^*_{a\#m}$. Then, the translation of CSPmn into CSP is given by the following.

Definition 16. *Let P be a CSPmn process.* $mn2csp(P) \mathrel{\widehat{=}} g^*_\# \circ ix(P)$

The following theorem establishes a labelled operational correspondence (Def. 2), which turns out a strong bisimulation (Def. 3), between CSPmn and CSP.

Theorem 2. *Let P be a CSPmn process. Let I denote a given sequence of natural numbers.*

1. *If* $P \xrightarrow{a} P'$ *then* $\exists I : mn2csp(P) \xrightarrow{a_I} Q$ *and* $Q \equiv mn2csp(P')$
2. *If* $mn2csp(P) \xrightarrow{a_I} Q$ *then* $\exists! P' : P \xrightarrow{a} P'$ *and* $Q \equiv mn2csp(P')$

Proof. When P does not mention $\underset{a\#m}{\parallel}$, $mn2csp$ behaves like the identity function, hence the theorem holds. By induction, we prove the case for parallel.

(Thrm.2.1.) [Induction step:Parallel]. Let $P_1 \xrightarrow{a} P_1'$. Let $P_2, .., P_n$ denote processes such that $m-1$ among them can perform an a-transition. For ease, we select one such combinations, $P_2..P_m$. The following result applies for all possible combinations. —(Hyp-combine)— Then, by M/N-IndxIfacePar rule (§3),

$$P_1 \underset{a\#m}{\parallel} .. \underset{a\#m}{\parallel} P_n \xrightarrow{a} P_1' \underset{a\#m}{\parallel} P_2' .. \underset{a\#m}{\parallel} P_m' \underset{a\#m}{\parallel} P_{m+1} \underset{a\#m}{\parallel} .. \underset{a\#m}{\parallel} P_n$$

Assume for each P_i that every occurrence of a in P_i is indexed into a_i. (The following applies even if we separate i into distinct indices, e.g., $i_1, i_2, ..$, as many as there are of instances of a in P_i.) —(Hyp-indx)— Then, by (Hyp-combine), (Hyp-indx), and Def.15, $g^*_\#(a) = a_{12..m}$ and:

$$mn2csp(P_1 \underset{a\#m}{\parallel} .. \underset{a\#m}{\parallel} P_n) = P_1[a_{12..m}/a] \underset{a_{12..m}}{\parallel} .. \underset{a_{12..m}}{\parallel} P_m[a_{12..m}/a] \underset{\{\}}{\parallel} P_{m+1} \underset{\{\}}{\parallel} .. \underset{\{\}}{\parallel} P_n$$

By IndxIfacePar rule (§3) and definition of renaming (Tab.2):

$$P_1[a_{12..m}/a] \underset{a_{12..m}}{\parallel} .. \underset{a_{12..m}}{\parallel} P_m[a_{12..m}/a] \underset{\{\}}{\parallel} P_{m+1} \underset{\{\}}{\parallel} .. \underset{\{\}}{\parallel} P_n \xrightarrow{a_{12..m}}$$
$$P_1'[a_{12..m}/a] \underset{a_{12..m}}{\parallel} .. \underset{a_{12..m}}{\parallel} P_m'[a_{12..m}/a] \underset{\{\}}{\parallel} P_{m+1} \underset{\{\}}{\parallel} .. \underset{\{\}}{\parallel} P_n$$

Then, by induction hypothesis.

(Thrm.2.2.) [Induction step: Prallel.] Let $mn2csp(P) \xrightarrow{a_I} Q$. By Par rule, $mn2csp(P) \underset{a_I}{\parallel} mn2csp(P_2) \xrightarrow{a_I} Q \underset{a_I}{\parallel} mn2csp(P_2)$, $a_I \notin \mathscr{A}(mn2csp(P_2))$. By induction hypothesis, $\exists! P' : P \xrightarrow{a} P'$ and $Q \equiv mn2csp(P')$. Then, by Par rule, $P \underset{a\#m}{\parallel} P_2 \xrightarrow{a} P' \underset{a\#m}{\parallel} P_2$. Moreover, $Q \underset{a_I}{\parallel} mn2csp(P_2) \equiv mn2csp(P') \underset{a_I}{\parallel} mn2csp(P_2) = mn2csp(P' \underset{a\#m}{\parallel} P_2)$, by Def.16. □

As a consequence, when m-among-n CSPmn processes, $\underset{a\#m,j}{\Big\|} P_j$, will synchronise on a, m-among-n CSP processes, $\underset{a_{12..m},j}{\Big\|} P_j[a_{12..m}/a]$, will synchronise on $a_{12..m}$, where $12..m$ denotes any combination of m potential synchronising processes. We say that $mn2csp$ implements m-among-n synchronisation.