Translation from CCS to CSP: the m-among-n Synchronisation Approach

Gerard Ekembe Ngondi, Vasileios Koutavas, Andrew Butterfield

EXPRESS/SOS Workshop 2022
CONFEST 2022, Warsaw, Poland

Trinity College Dublin

Lero - the SFI Research Centre for Software

12-16 September 2022

Introduction - 1/2

- CCS (Calculus of Communicating Systems) and CSP (Communicating Sequential Processes) are two prominent calculi for reasoning about concurrent programs
  - CCS - late Prof. R. Milner, Turing Award 1991
  - CSP - Sir Prof. T. Hoare, Turing Award 1980
  - Hundreds of papers published based off both calculi
- What is the difference between CCS and CSP?
  - Ekembe et al. (*CCS to CSP Translation*, Dec.2021) answer this question
    - By translating CCS into CSP, excluding parallel under recursion
    - Shows that the translation function, $ccs2csp$, is correct up to strong bisimulation, viz., $ccs2csp(P) \sim P$
- Our main interest: Unify both CCS and CSP worlds (incl. Pi-calculus and CSPmob)

Introduction - 2/2

- **This paper**: Translate CCS into CSP, *including parallel under recursion*
  - Our new translation, $ccs2csp_3$ is correct up to strong bisimulation
  - Compositional
- **How?** Extend CSP with m-among-n synchronisation, called CSPmn
  - CSPmn is a conservative extension of CSP
  - binary synchronisation can be defined through multiway or n-among-n synchronisation (the default CSP synchronisation mechanism) and renaming
  - m-among-n synchronisation can be defined through multiway synchronisation and renaming

Formalisation – Labelled Operational Correspondence between CCS and CSP

	CCS	CSP
Termination	$0 \not\rightarrow$	$STOP \not\rightarrow$
Prefix	$a.P \xrightarrow{a} P$	$(a \leadsto P) \xrightarrow{a} P$
Tau Prefix	$\tau.P \xrightarrow{\tau} P$	$(tau \leadsto P)\setminus_{csp} \{tau\} \xrightarrow{\tau} P$
Ext Choice	$a.P + b.Q \xrightarrow{a} P$ $a.P + b.Q \xrightarrow{b} Q$	$a \leadsto P \,\Box\, b \leadsto Q \xrightarrow{a} P$ $a \leadsto P \,\Box\, b \leadsto Q \xrightarrow{b} Q$
Int Choice	$\tau.P + \tau.Q \xrightarrow{\tau} P$ $\tau.P + \tau.Q \xrightarrow{\tau} Q$	$P \sqcap Q \xrightarrow{\tau} P$ $P \sqcap Q \xrightarrow{\tau} Q$
Mixed Choice	$a.P + \tau.Q \xrightarrow{a} P$ $a.P + \tau.Q \xrightarrow{\tau} Q$	$(a \leadsto P \,\Box\, tau \leadsto Q)\setminus_{csp} \{tau\} \xrightarrow{a} P\setminus_{csp} \{tau\}$ $(a \leadsto P \,\Box\, tau \leadsto Q)\setminus_{csp} \{tau\} \xrightarrow{\tau} Q\setminus_{csp} \{tau\}$
Restriction	$(a.P) \upharpoonright \{a\} \not\rightarrow$ $(a.P) \upharpoonright \{b\} \xrightarrow{a} P \upharpoonright \{b\}$ $(a.P \mid \bar{a}.Q) \upharpoonright \{a\} \xrightarrow{\tau} P \mid Q$	$(a \leadsto P) \underset{\{a\}}{\parallel} STOP \not\rightarrow$ $(a \leadsto P) \underset{\{b\}}{\parallel} STOP \xrightarrow{a} P \underset{\{b\}}{\parallel} STOP$ $(a \leadsto P \underset{\{a\}}{\parallel} a \leadsto Q)\setminus_{csp} \{a\} \underset{\{a\}}{\parallel} STOP \xrightarrow{\tau}$ $(P \underset{\{a\}}{\parallel} Q)\setminus_{csp} \{a\} \underset{\{a\}}{\parallel} STOP$
Recursion	$\mu X.P \xrightarrow{\alpha} P'$	$\mu X.P \xrightarrow{\alpha} P'$

Formalisation: Parallel - 1/2

- CCS Parallel is complex:

$$\left.\begin{array}{r} a.0 \mid \bar{a}.0 \equiv a.\bar{a}.0 + \bar{a}.a.0 + \tau.0 \\ a.\bar{a}.0 + \bar{a}.a.0 \equiv a.0 \parallel\!\mid \bar{a}.0 \\ \tau.0 \equiv (a.0 \mid \bar{a}.0) \restriction \{a\} \end{array}\right\} \Rightarrow a.0 \mid \bar{a}.0 \equiv (a.0 \parallel\!\mid \bar{a}.0) + (a.0 \mid \bar{a}.0) \restriction \{a\}$$

- In CSP:

$$\Big(\underbrace{(a \rightsquigarrow STOP \parallel\!\mid a \rightsquigarrow STOP)}_{\text{Interleaving} \times \text{Visible!}} \,\Box\, \underbrace{(a \rightsquigarrow STOP \underset{\{a\}}{\parallel} a \rightsquigarrow STOP)}_{\text{Synchronisation} \times \text{Invisible!}} \Big) \setminus_{csp} \{a\}$$

(Req-sep) Thus, we need to separate interleaving from synchronisation!

(Req-sync) We need to make synchronisation visible!

- Solution from Ekembe et al. (*CCS to CSP Translation*, Dec.2021):

$$\Big((a \rightsquigarrow STOP \parallel\!\mid a \rightsquigarrow STOP) \,\Box\, (a_{12} \rightsquigarrow STOP \underset{\{a_{12}\}}{\parallel} a_{12} \rightsquigarrow STOP) \Big) \setminus_{csp} \{a_{12}\}$$

- Limitation: the translation needs to generate every synchronisation index, hence cannot terminate for CCS terms with parallel under recursion

Formalisation: Parallel - 2/2

- New solution: define binary synchronisation in CSP, then translate CCS binary sync. into CSP binary sync.
  - Let $\underset{a\,\#\,m}{\|}$ denote m-among-n synchronisation on event a.
  - E.g., $a \underset{a}{\|} a \underset{a}{\|} a \xrightarrow{a} STOP$ coincides with $a \underset{a\,\#\,3}{\|} a \underset{a\,\#\,3}{\|} a \xrightarrow{a} STOP$ while $a \underset{a\,\#\,2}{\|} a \underset{a\,\#\,2}{\|} a \xrightarrow{a} STOP \underset{a\,\#\,2}{\|} a \not\rightarrow$ and $a \underset{a\,\#\,4}{\|} a \underset{a\,\#\,4}{\|} a \not\rightarrow$
- CCS parallel case $a.0 \mid \bar{a}.0 \mid \bar{a}.0$ corresponds with the following CSPmn process:

$$\Big((a \rightsquigarrow STOP \,|||\, a \rightsquigarrow STOP \,|||\, a \rightsquigarrow STOP) \,\Box\, (a_S \rightsquigarrow STOP \underset{\{a_S\,\#\,2\}}{\|} a_S \rightsquigarrow STOP \underset{\{a_S\,\#\,2\}}{\|} a_S \rightsquigarrow STOP)\Big) \setminus_{csp} \{a_S\}$$

Contrast with Ekembe et al. (*CCS to CSP Translation*, Dec.2021):

$$\Big((a \rightsquigarrow STOP \,|||\, a \rightsquigarrow STOP \,|||\, a \rightsquigarrow STOP) \,\Box\, (a_{12} \,\Box\, a_{13} \rightsquigarrow STOP \underset{\{a_{12},a_{13}\}}{\|} a_{12} \rightsquigarrow STOP \underset{\{a_{12},a_{13}\}}{\|} a_{13} \rightsquigarrow STOP)\Big) \setminus_{csp} \{a_{12}, a_{13}\}$$

CSPmn semantics

- The rules for m/n indexed interface paralell composition are given hereafter.

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

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

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

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

Correctness of CSPmn

- Every CSP process with a_{ij} synchronisation is equivalent to a CSPmn process obtained by mapping every a_{ij} to a single a_S (via renaming), and mapping $\underset{\{a_{ij}\}}{\|}$ unto $\underset{a_S \,\#\, 2}{\|}$
  - Conversely, binary synchronisation can be defined from renaming and mutliway synchronisation
- More generally, m-among-n synchronisation can be defined from renaming and multiway synchronisation
  - Consequence: CSPmn is a conservative extension of CSP
- E.g.

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

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

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

Translation Workflow

1. Make the result of synchronisation visible:
   - in CCS: $(a, \bar{a}) \mapsto \tau$
   - in CCSTau: $(a, \bar{a}) \mapsto \tau[a \mid \bar{a}]$
2. Separate interleaving from synchronisation:
   - Generate a unique index per complementary prefix pairs, e.g., $(a, \bar{a}) \mapsto (a_S, \bar{a}_S)$
3. Translate CCS operators into corresponding CSP operators,
   - e.g., τ maps to tau, $+$ maps to $\Box$, $|$ maps to $\|$
4. Hide a_S synchronisation names

CCSTau - Syntax and Semantics

- CCSTau extends CCS with visible synchronisation and hiding
- To make synchronisation observable we use the following Com rule:

$$Com : \frac{P \xrightarrow{\overline{a}} P' \quad Q \xrightarrow{a} Q'}{P \mid_T Q \xrightarrow{\tau[\overline{a}|a]} P' \mid_T Q'}$$

- We introduce the following hiding rules in the LTS which are similar to the CSP rules:

$$Hide1 : \frac{P \xrightarrow{\beta} P' \quad \beta \notin B}{P \setminus_T B \xrightarrow{\beta} P' \setminus_T B} \qquad Hide2 : \frac{P \xrightarrow{\beta} P' \quad \beta \in B}{P \setminus_T B \xrightarrow{\tau} P' \setminus_T B}$$

CCSTau - Link from CCS

- We describe here a translation of CCS processes into CCSTau.
  - This encoding is concerned with hiding the now-observable synchronisation actions.

Definition 1 ($c2ccs\tau$)

Translation function $c2ccs\tau$, when applied to a CCS process, returns a CCSTau process.

$$c2ccs\tau(P \mid Q) \mathrel{\widehat{=}} (c2ccs\tau(P) \mid_T c2ccs\tau(Q)) \setminus_T \{\tau[a \mid \overline{a}] \mid a \in \mathcal{A}(P), \bar{a} \in \mathcal{A}(Q)\}$$

Theorem 1

Let P be a CCS process. Then: $P \sim c2ccs\tau(P)$.

- Every CCS process is a CCSTau process

Running Example - step 1

$$\boxed{CCS} \xrightarrow{c2ccs\tau} \boxed{CCSTau} \xrightarrow{ix} \xrightarrow{g^*} \xrightarrow{conm} \boxed{CCSTau} \xrightarrow{tl} \boxed{CSP} \xrightarrow{\backslash\{tau, a_{ij}\}} \xrightarrow{ai2a} \boxed{CSP}$$

Figure: $ccs2csp$ Translation workflow

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

Figure: $ccs2csp_3$ Translation workflow

- Running Example: $(a.P \mid \overline{a}.Q) \mid \overline{a}.R$
- We initially translate this process into CCSTau through the $c2ccs\tau$ function (Def.1), which gives us:

$$((a.P' \mid_T \overline{a}.Q') \backslash_T \{\tau[a \mid \overline{a}]\} \mid_T \overline{a}.R') \backslash_T \{\tau[a \mid \overline{a}]\}$$

Running Example - step 2

$$\boxed{CCS} \xrightarrow{c2ccs\tau} \boxed{CCSTau} \xrightarrow{ix} \xrightarrow{g^*} \xrightarrow{conm} \boxed{CCSTau} \xrightarrow{tl} \boxed{CSP} \xrightarrow{\backslash\{tau, a_{ij}\}} \xrightarrow{ai2a} \boxed{CSP}$$

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

- E.g.: $(a.P \mid \overline{a}.Q) \mid \overline{a}.R$
- $c2ccs\tau$ (Def.1): $((a.P' \mid_T \overline{a}.Q')\backslash_T\{\tau[a \mid \overline{a}]\} \mid_T \overline{a}.R')\backslash_T\{\tau[a \mid \overline{a}]\}$
- $(ccs2csp)$ Then ix: $a_1.P'' \mid_T \overline{a}_2.Q'' \mid_T \overline{a}_3.R''$
- $(ccs2csp)$ Then g^*:

$$(a_1 + a_{12} + a_{13}).P''' \mid_T ((\overline{a}_2 + \overline{a}_{12}).Q''') \mid_T (\overline{a}_3 + \overline{a}_{13}).R'''$$

  where $(a + b).S$ is syntactic sugar for $a.S + b.S$.
- In contrast, for $ccs2csp_3$, apply g^2:

$$(a + a_S).P''' \mid_T ((\overline{a} + \overline{a}_S).Q''') \mid_T (\overline{a} + \overline{a}_S).R'''$$

Running Example - step 3

$$\boxed{CCS} \xrightarrow{c2ccs\tau} \boxed{CCSTau} \xrightarrow{ix} \xrightarrow{g^*} \xrightarrow{conm} \boxed{CCSTau} \xrightarrow{tl} \boxed{CSP} \xrightarrow{\backslash\{tau, a_{ij}\}} \xrightarrow{ai2a} \boxed{CSP}$$

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

- E.g.: $(a.P \mid \overline{a}.Q) \mid \overline{a}.R$
- $c2ccs\tau$ (Def.1): $((a.P' \mid_T \overline{a}.Q')\backslash_T\{\tau[a \mid \overline{a}]\} \mid_T \overline{a}.R')\backslash_T\{\tau[a \mid \overline{a}]\}$
- $ix \circ g^*$: $(a_1 + a_{12} + a_{13}).P''' \mid_T ((\overline{a}_2 + \overline{a}_{12}).Q''') \mid_T (\overline{a}_3 + \overline{a}_{13}).R'''$
  - g^2: $(a + a_S).P''' \mid_T ((\overline{a} + \overline{a}_S).Q''') \mid_T (\overline{a} + \overline{a}_S).R'''$
- $conm$ identifies co-names synchronisation events. For $ccs2csp$:

$$((a_1 + a_{12} + a_{13}).P''' \mid_T (\bar{a}_2 + a_{12}).Q''') \mid_T (\bar{a}_3 + a_{13}).R'''$$

- And for $ccs2csp_3$:

$$((a + a_S).P''' \mid_T (\bar{a} + a_S).Q''') \mid_T (\bar{a} + a_S).R'''$$

Operator Translation

- We can now present the translation tl from CCSTau to CSP.

Definition 2

Let P, Q be CCSTau processes and tau be a fresh CSP event.

$$tl_3(0) \mathrel{\widehat{=}} STOP$$

$$tl_3(\tau.P) \mathrel{\widehat{=}} tau \rightsquigarrow tl_3(P)$$

$$tl_3(a.P) \mathrel{\widehat{=}} a \rightsquigarrow tl_3(P)$$

$$tl_3(P + Q) \mathrel{\widehat{=}} tl_3(P) \,\Box\, tl_3(Q)$$

$$tl_3(P \mid_T Q) \mathrel{\widehat{=}} tl_3(P) \underset{\{a \# 2 \mid a \in \mathcal{A}(P) \cap \mathcal{A}(Q)\}}{\parallel} tl_3(Q)$$

$$tl_3(P \restriction B) \mathrel{\widehat{=}} tl_3(P) \restriction_{csp} B$$

$$tl_3(P \setminus_T B) \mathrel{\widehat{=}} tl_3(P) \setminus_{csp} B$$

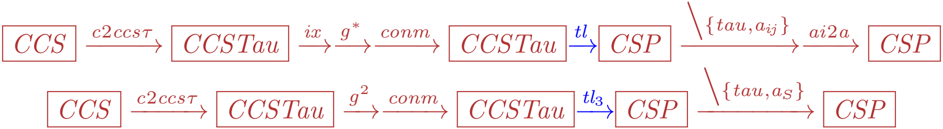
$$tl_3(\mu X.P) \mathrel{\widehat{=}} \mu X.tl_3(P)$$

Definition 3

$$tl(P \mid_T Q) \mathrel{\widehat{=}} tl(P) \underset{\{a \mid a \in \mathcal{A}(P) \cap \mathcal{A}(Q)\}}{\parallel} tl(Q)$$

$$tl(P) = tl_3(P) \quad \text{If } P \text{ is not parallel composition}$$

Running Example - step 4



- E.g.: $(a.P \mid \overline{a}.Q) \mid \overline{a}.R$
- $c2ccs\tau$ (Def.1): $((a.P' \mid_T \overline{a}.Q') \setminus_T \{\tau[a \mid \overline{a}]\} \mid_T \overline{a}.R') \setminus_T \{\tau[a \mid \overline{a}]\}$
- $ix \circ g^*$: $(a_1 + a_{12} + a_{13}).P''' \mid_T ((\overline{a}_2 + \overline{a}_{12}).Q''') \mid_T (\overline{a}_3 + \overline{a}_{13}).R'''$
  - g^2: $(a + a_S).P''' \mid_T ((\overline{a} + \overline{a}_S).Q''') \mid_T (\overline{a} + \overline{a}_S).R'''$
- Then $conm$:
  $((a_1 + a_{12} + a_{13}).P''' \mid_T (\bar{a}_2 + a_{12}).Q''') \mid_T (\bar{a}_3 + a_{13}).R'''$
  - And: $((a + a_S).P''' \mid_T (\bar{a} + a_S).Q''') \mid_T (\bar{a} + a_S).R'''$
- tl (Def.3):

$$((a_1 \Box a_{12} \Box a_{13}) \rightsquigarrow P'''' \underset{\{a_{12}\}}{\parallel} (\bar{a}_2 \Box a_{12}) \rightsquigarrow Q'''') \underset{\{a_{13}\}}{\parallel} (\bar{a}_3 \Box a_{13}) \rightsquigarrow R''''$$

- tl_3 (Def.2):

$$((a \Box a_S) \rightsquigarrow P'''' \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow Q'''') \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow R''''$$

Running Example - step 5

$$\boxed{CCS} \xrightarrow{c2ccs\tau} \boxed{CCSTau} \xrightarrow{ix} \xrightarrow{g^*} \xrightarrow{conm} \boxed{CCSTau} \xrightarrow{tl} \boxed{CSP} \xrightarrow{\backslash\{tau, a_{ij}\}} \xrightarrow{ai2a} \boxed{CSP}$$

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

- E.g.: $(a.P \mid \overline{a}.Q) \mid \overline{a}.R$
- *tl*: $((a_1 \Box a_{12} \Box a_{13}) \rightsquigarrow P'''' \underset{\{a_{12}\}}{\parallel} (\bar{a}_2 \Box a_{12}) \rightsquigarrow Q'''') \underset{\{a_{13}\}}{\parallel} (\bar{a}_3 \Box a_{13}) \rightsquigarrow R''''$
  - tl_3: $((a \Box a_S) \rightsquigarrow P'''' \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow Q'''') \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow R''''$
- The final CSP term is thus:

$$(((a \Box a_{12} \Box a_{13}) \rightsquigarrow P'''' \underset{\{a_{12}\}}{\parallel} (\bar{a} \Box a_{12}) \rightsquigarrow Q'''') \underset{\{a_{13}\}}{\parallel} (\bar{a} \Box a_{13}) \rightsquigarrow R'''')\backslash_{csp}\{tau\}\backslash_{csp}\{a_{12}, a_{13}\}$$

- The final CSPmn term is thus:

$$(((a \Box a_S) \rightsquigarrow P'''' \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow Q'''') \underset{\{a_S \,\#\, 2\}}{\parallel} (\bar{a} \Box a_S) \rightsquigarrow R'''')\backslash_{csp}\{tau\}\backslash_{csp}\{a_S\}$$

Example 2 - Recursion

Let $P \mathrel{\widehat{=}} \mu\, X(a \mid \bar{a}.X)$ (or equiv. $P \mathrel{\widehat{=}} a.0 \mid \bar{a}.P$) be a CCS process.
Then, $ix(P) = a_1 \mid \bar{a}_2.ix_{\{3..\}}(P)$,

$$P = a \mid \bar{a}.(a \mid \bar{a}.P)$$
$$ix(P) = a_1 \mid \bar{a}_2.(a_3 \mid \bar{a}_4.ix_{\{5..\}}(P))$$

The synchronisation pairs are thus $(a_1, \bar{a}_2), (a_1, \bar{a}_4), .., (a_3, \bar{a}_4), ...$ We will not be able to generate all the a_{1*2k} ($k \geq 1$) indices since recursion is unbounded. In contrast, let us define $ccs2csp_3(P)$. Then:

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

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

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

Correctness of the Translation - 1/2

Theorem 2 (Correctness of $ccs2csp$)

Let P be a CCS process. Then:

1. $P \xrightarrow{\tau} P'$ *imply that*
   $\forall\, S \mid S \cap \mathcal{A}(ix(P)) = \{\} : ccs2csp(S, P) \xrightarrow{\tau} ccs2csp(S, P')$
2. $\forall\, S \mid S \cap \mathcal{A}(ix(P)) = \{\} : ccs2csp(S, P) \xrightarrow{\tau} Q$ *imply that*
   $\exists! P' : P \xrightarrow{\tau} P'$ *and* $Q = ccs2csp(S, P')$
3. $P \xrightarrow{a} P'$ *imply that*
   $\forall\, S \mid S \cap \mathcal{A}(ix(P)) = \{\} : ccs2csp(S, P) \xrightarrow{a} ccs2csp(S, P')$
4. $\forall\, S \mid S \cap \mathcal{A}(ix(P)) = \{\} : ccs2csp(S, P) \xrightarrow{a} Q$ *imply that*
   $\exists! P' : P \xrightarrow{a} P'$ *and* $Q = ccs2csp(S, P')$

We say that $ccs2csp$ is correct up to strong bisimulation.

Corollary 4

Let P be a CCS process. Then: $P \sim ccs2csp(P)$.

Correctness of the Translation - 2/2

Definition 5 (gstar2m/n)

Let a_{ij} be an g^* name, a_S an g^2 name. Then: $g^*2g^2 \mathrel{\widehat{=}} \{\tau \mapsto \tau, a_{ij} \mapsto a_S\}$

Definition 6

Let P be a CSP process.

$$g^*2g^2(\alpha \rightsquigarrow P) \mathrel{\widehat{=}} g^*2g^2(\alpha) \rightsquigarrow g^*2g^2(P)$$
$$g^*2g^2(P \underset{\{a_{ij}\}}{\parallel} Q) \mathrel{\widehat{=}} g^*2g^2(P) \underset{\{a_S \,\#\, 2\}}{\parallel} g^*2g^2(Q)$$
$$\ldots$$

Theorem 3

Let P be a CCS process. Then: $g^*2g^2 \circ ccs2csp(P) = ccs2csp_3(P)$.

Corollary 7

Let P be a CCS process. Then: $P \sim ccs2csp_3(P)$.

Conclusion, Future Work

- $ccs2csp$ translation has been implemented in Haskell
  - GitHub Repo: https://github.com/andrewbutterfield/ccs2csp
  - Next: implement $ccs2csp_3$
- Next: extend FDR with m-among-n synchronisation
- Ongoing: Translate Pi-calculus into CSPmob
- Questions?

Structural Properties

- Gorla (*Towards a Unified Approach to Encodability and Separation Results for Process Calculi*, 2010) proposes five requirements for a translation to be *valid*:
  - operational correspondence
    - ✓ a CCS term is strong bisimilar to its translation
  - divergence reflection
    - ✓ if a CSPmn translation diverges then its source CCS term does;
  - success sensitiveness
    - ✓ a CCS term converges if, and only if, its CSPmn translation converges, and both converge to the same success final term
  - name invariance
    - ✓ typically, $ccs2csp_3(f(P)) \sim f(ccs2csp_3(P))$
  - compositionality
    - ✓ Our translation is compositional