An Encoding of the CSP External Choice operator to ACP_τ

Leon Lee

December 9, 2024

1.1 Prerequisites

We represent a language $\mathscr{L}$ as a triple $(\mathbb{P}, [\![\ ]\!])$, where $\mathbb{P}$ is a set of processes composed of actions, and $[\![\ ]\!]$ is a set of operators acting on $\mathbb{P}$. We also define $A \subseteq \mathbb{P}$, where A is the set of actions

Definition 1.1.1: Subsets of A

We define some subsets of A which we will use in our encodings

- $A_0 \subseteq A$ is the set of actions that actually get used in processes
- $H_0 = A - A_0$ is the set of working space operators, or any other action that doesn't get used
- $H_1 = A_0 \uplus \{\texttt{first}, \texttt{next}, \texttt{choose}\}$ is the set of actions, plus some working operators

In general, $A_0 \subseteq H_1 \subseteq A$.
Note that the silent step is not defined in A, and we will define A_τ to be $A \cup \{\tau\}$

Working similarly to [3], we will work in the language ACP_τ, with an additional operator called "Functional Renaming". I will refer to this language as ACP_τ^F. To define translations from CSP to ACP_τ^F we utilise two operators - $\rho_F(p)$ and ∂_H.

- **Functional Renaming**: $\rho_F(p)$ takes a function $F : A \to A$, and where applicable, applies F to each action in p
- **Encapsulation**: ∂_H takes a subset $H \subseteq A$, and restricts any actions that are in H

1.2 Triggering in ACP

We define an operator $\Gamma(p)$ which acts like the MEIJE triggering operator on a process $p \in \mathbb{P}$. For this, we will use the notation of a^∞ to mean $a.a.a\ldots$. First, we define a function f_1 and communications for the operations `first` and `next`.

Definition 1.2.1: F1

Define a function $f_1 : A \to A$ where

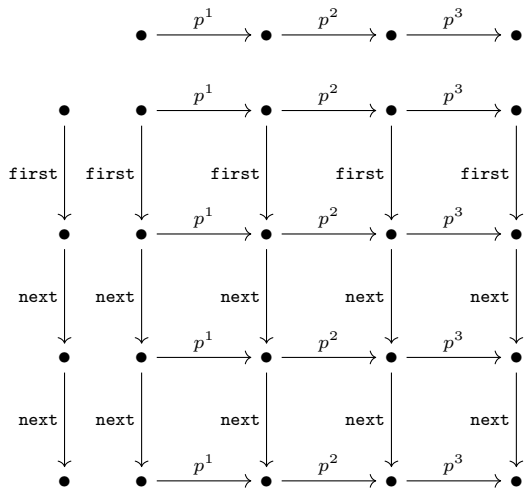
$$f_1(a_{\texttt{first}}) = a_{\texttt{first}}$$
$$f_1(a_{\texttt{next}}) = a$$

Definition 1.2.2: Communication

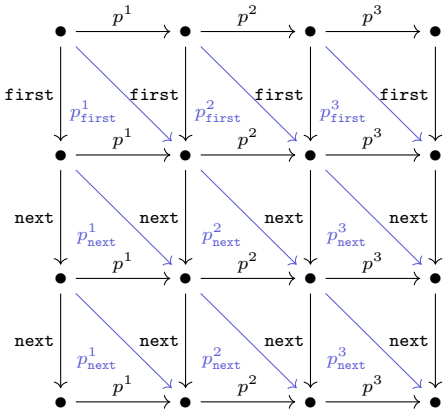
Define communications where

$$a|\texttt{first} = a_{\texttt{first}}$$
$$a|\texttt{next} = a_{\texttt{next}}$$

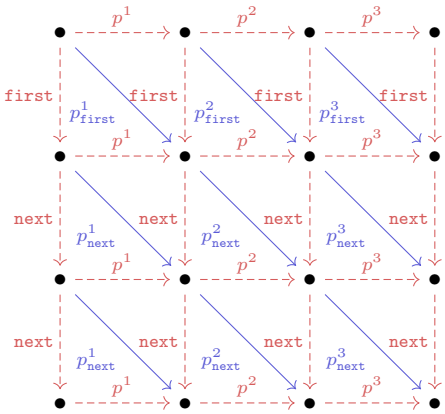
We can now define $\Gamma(p)$ as such:



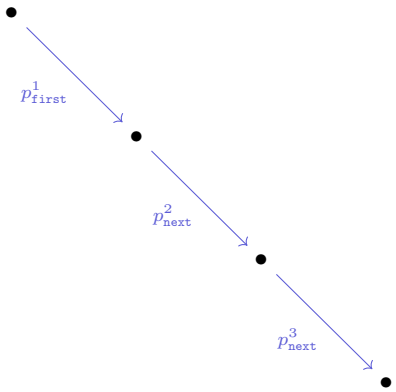
(a) Step 1 - Left Merge on both sides



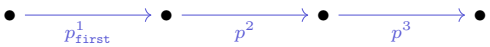
(b) Step 1 - Communications



(c) Step 2 - Restriction of H_1



(d) Step 2 - Result of Restriction



(e) Result of Functional Renaming

Figure 1: Example of $\Gamma(p)$ applied to $p = p^1.p^2.p^3$

Definition 1.2.3: Triggering in ACP

$$\Gamma(p) := \rho_{f_1}[\partial_{H_1}(p||\texttt{first}(\texttt{next}^\infty))]$$

is an operator that turns a trace of a process p, $p_1.p_2.p_3.\ldots$ into the trace

$$p^1_{\texttt{first}}.p^2.p^3.p^4.\ldots$$

This works in the following method:

1. Merge the process p with the process $\texttt{first.next.next}\ldots$. Via 1.2.2, this will produce a lattice of p and $\texttt{first}.(\texttt{next}^\infty)$, with communications on every square, but most importantly, a chain of communications going down the centre of the form.

$$p^1_{\texttt{first}}.p^2_{\texttt{next}}.p^3_{\texttt{next}}\cdots \tag{1}$$

2. Restrict the actions in H_1. Since both $p, \texttt{first}.(\texttt{next}^\infty) \in H_1$ this effectively restricts both sides of the left merge, leaving only communications from the initial state. This leaves equation 1 as the only remaining trace.

3. Apply ρ_{f_1} to equation 1. Via 1.2.1, the final result is

$$p^1_{\texttt{first}}.p^2.p^3\ldots \tag{2}$$

   Which is exactly as stated in Definition 1.2.3.

Note that since $\tau \notin A$, ∂_{H_1} will not restrict τ, and additionally since τ does not communicate, Step 2 effectively becomes any amount of τ steps followed by the diagonal trace immediately following that. This results in cases $\Gamma(p)$ where $p = \tau.p^1.p^2\ldots$ becoming the trace

$$\tau.p^1_{\texttt{first}}.p^2.p^3\ldots$$

effectively skipping τ's, then acting the same as processes that don't start with a τ.

1.3 A Translation of CSP external choice

The external choice operator $\Box$ is defined with the following rules:

$$\frac{P \xrightarrow{a} P'}{P\Box Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P\Box Q \xrightarrow{a} Q'} \qquad \frac{P \xrightarrow{\tau} P'}{P\Box Q \xrightarrow{\tau} P'\Box Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P\Box Q \xrightarrow{a} P\Box Q'} \tag{3}$$

In other words, we can take an external choice by the user, and additionally an internal action will still let an external choice be made after the internal move has been made. This differs from the ACP Alternative Choice operator ($+$), as $+$ will not let you select externally if an internal action is made.

For our encoding, we take $H_1 = A_0 \uplus \texttt{first}, \texttt{next}, \texttt{choose}$ as defined in Definition 1.1.1. We then modify Definition 1.2.2 to include an additional communication for `choose` to make

Definition 1.3.1: Communication for CSP External Choice

$$a|\texttt{first} = a_{\texttt{first}} \qquad a|\texttt{next} = a_{\texttt{next}} \qquad a_{\texttt{first}}|\texttt{choose} = a$$

We can then define an encoding of the CSP external choice operator $\Box$ in ACP in the following equation

$$\mathscr{T}(P\Box Q) = \partial_{H_0}\Big(\Gamma[\mathscr{T}(P)] \,||\, \texttt{choose} \,||\, \Gamma[\mathscr{T}(Q)]\Big)$$

On a process without τ, the function

$$\partial_{H_0}\Big(\Gamma(P) \,||\, \texttt{choose} \,||\, \Gamma(Q)\Big)$$

has identical behaviour to $P + Q$

On processes with internal actions, which we will call $\mathcal{P} = \tau_P.a.P$ and $\mathcal{Q} = \tau_Q.b.Q$, where τ_P indicates the number of starting τ actions in the process, possibly 0, the τ cannot communicate

with `choose` so `choose` will only communicate with a (Since only a will be labelled with `first`). However, the first action of Q will still have the name $b_{\texttt{first}}$ for $b \in A$. This lets the function effectively skip the τ, then perform `choose` on $a.P \,||\, b.Q$ which matches the behaviour of CSP $\Box$.

However, this translation is not strongly bisimilar. This is due to the final restriction ∂_{H_0}, which will usually restrict any stray $a_{\texttt{first}}$ actions (and subsequent actions/processes) that try to communicate. However, since the silent step τ is not in A, we will get left with stray τ actions. This doesn't hold for two processes $a.P$ and $\tau.Q$, since the translation would yield the process

$$a.(\tau||P) + \tau.(P+Q) \neq a.P + \tau.(P+Q)$$

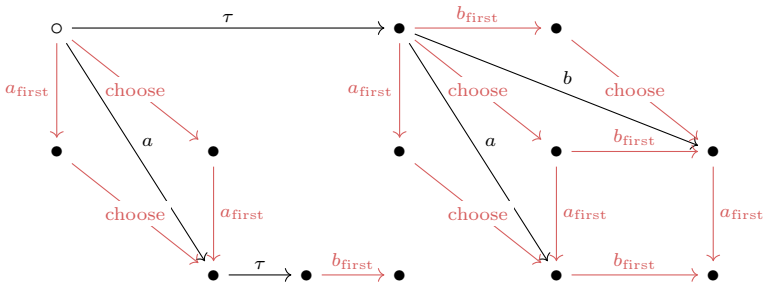


Figure 2: Counterexample for Strong Bisimilarity with the processes $P = a$ and $Q = \tau.b$. The result of the translation is $a.\tau + \tau.(a+b) \not\leftrightarrows a + \tau.(a+b)$. Restricted actions are marked in red.

1.4 Proving our translation of External Choice is valid up to Rooted Branching Bisimulation

We define formally the definition of Rooted Branching Bisimulation:

Definition 1.4.1: Rooted Branching Bisimilarity

Let P and Q be two processes, and R be a relation between nodes of P and nodes of Q. R is a **Branching Bisimulation** between P and Q if:

1. The roots of P and Q are related by R
2. If $s \xrightarrow{a} s'$ for $a \in A \cup \{\tau\}$ is an edge in P, and sRt, then either
   a) $a = \tau$ and $s'Rt$
   b) $\exists t \Rightarrow t_1 \xrightarrow{a} t'$ such that sRt_1 and sRt'
3. If $s \downarrow$ and sRt then there exists a path $t \Rightarrow t'$ in Q to a node t' with $t' \downarrow$ and sRt'

4, 5 : As in 2, 3, with the roles of P and Q interchanged

R is called a **Rooted Branching Bisimulation** if the following root condition is satisfied:

- If $\text{root}(P) \xrightarrow{a} s'$ for $a \in A \cup \{\tau\}$, then there is a t' with $\text{root}(Q) \xrightarrow{a} t'$ and $s'Rt'$
- If $\text{root}(Q) \xrightarrow{a} t'$ for $a \in A \cup \{\tau\}$, then there is a s' with $\text{root}(P) \xrightarrow{a} s'$ and $s'Rt'$
- $\text{root}(g) \downarrow$ iff $\text{root}(h) \downarrow$

In other words, two processes are Rooted Branching Bisimilar if it is strongly bisimilar for the first step, and branching bisimilar for the remaining ones. This is a more desirable outcome because RBB is a congruence [2].

We can now work towards a proof that the external choice is Rooted Branching Bisimilar. In the case that P and Q don't involve an internal action τ, the translation is strongly bisimilar, and therefore RBB since Strong bisimilarity is a finer congruence than Rooted Branching Bisimilarity.

In the case that P or Q involves a τ, the processes are no longer strongly bisimilar, see Figure 2. We will now show that processes with internal actions are also valid up to Rooted Branching Bisimilarity.

The root condition is trivially satisfied. For two processes $a.P$ and $\tau.Q$, the proposed translation $\mathscr{T}(a.P \Box \tau.Q)$ will either yield an a, or a τ, which is intended behaviour.

We will now state some theorems which will simplify the process graphs and make it easier to show the Rooted Branching Bisimulation. A working example is shown which is the process

$$\tau.a \,\Box\, \tau.\tau.b$$

Theorem 1.4.2: Lattice form of CSP External Choice

For two processes P and Q, the resulting process graph of $P \Box Q$ can be rewritten as a lattice with the rows and columns corresponding to the number of τ transitions on either process

An example is shown below with the working example

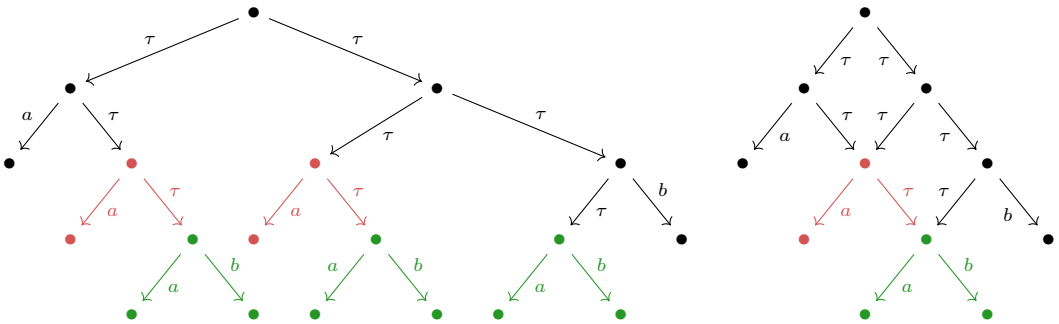


Figure 3: Lattice form of the process $\tau.a \,\Box\, \tau.\tau.b$. A sketch of the lattice form as shown by Partition Refinement is highlighted by the red and green states and transitions.

Proof. It can be inferred from the rules mentioned in Equation 3 that if there is a choice between a τ on both P and Q, then the resulting process graph is the equivalent to a lattice of communication between τ actions. The other actions result in a choice and so the same rule doesn't apply. This result can also be gained from Partition Refinement. □

Writing the process graph in lattice form results in the process graph of the External Choice being very similar to the process graph of the proposed translation. This is because the translation is based on a communication, so there also ends up being a similar τ lattice of the same form, and the only difference is τ transitions on the rest of the process going past the τ-lattice.

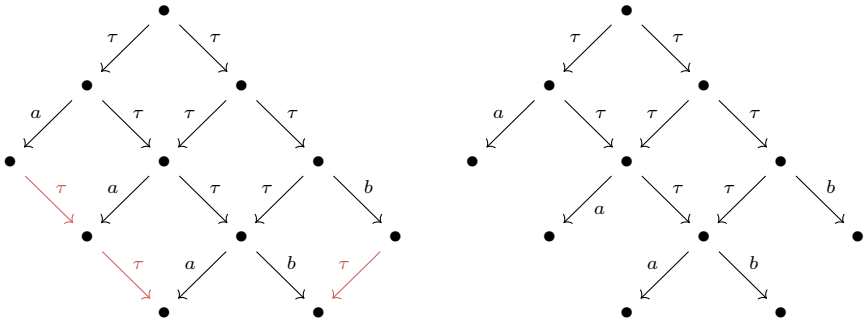


Figure 4: Comparison of proposed translation of $\tau.a \,\square\, \tau.\tau.b$ (Left) and actual representation (Right)

Theorem 1.4.3: Minimal Process of CSP External Choice

For two processes, $\mathcal{P} = \tau_P.a.P$, $\mathcal{Q} = \tau_Q.b.Q$, where τ_P and τ_Q are the number of starting τ transitions, possibly 0, the least number of states of $\mathcal{P} \,\square\, \mathcal{Q}$ is

$$n(\tau_P) \cdot n(\tau_Q) + n(P) + n(Q)$$

states, where $n(P)$ is the number of states in a process P.

Proof. Applying Theorem 1.4.2, We can then undergo the partition refinement algorithm for Strong Bisimilarity as detailed in [1]. Via the rules of the external choice in Equation 3, under an internal choice $P \xrightarrow{\tau} P'$, the process Q remains the exact same, and the same can be said for an internal choice in Q. Therefore, any choice branches from a τ action are equivalent, and are therefore part of the same equivalent class. We can then group the minimal states as follows

$$\text{Number of states in lattice} + \text{Number of states in } P + \text{Number of states in } Q$$

□

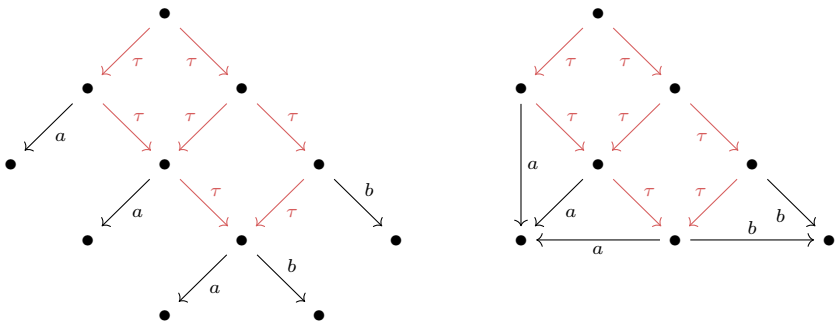


Figure 5: Minimal process of $\tau.a \,\square\, \tau.\tau.b$. Lattice is highlighted in red

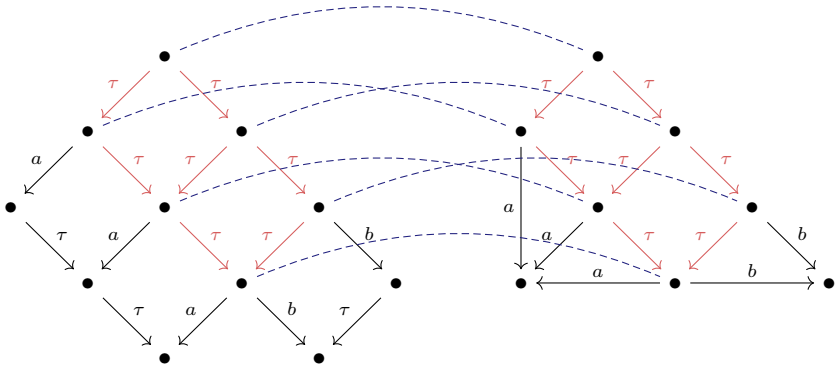
We can now work on the Rooted Branching Bisimulation.

Lemma 1.4.4

There exists an isomorphism from the τ-lattice on the proposed translation to the actual representation of external choice

Proof. We perform action refinement of Strong Bisimilarity. The rest follows trivially. □

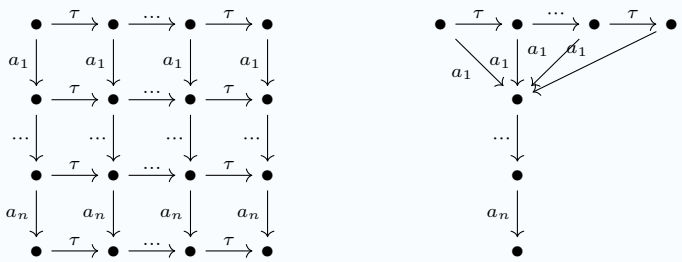
An example is shown below:



Therefore, the only remaining parts of the process graphs we need to check are if the rest of both (disjoint) processes are also Branching Bisimilar. This means that for both processes $\mathcal{P} = \tau_P.a.P$ and $\mathcal{Q} = \tau_Q.b.Q$, the parts we need to find a bisimulation for are of the form $a.P$ put in parallel with a chain of τ actions compared to a process of the form $a.P$ with a leading chain of τ actions, and the same for $b.Q$.

Lemma 1.4.5

Comparisons of $a.P$ and $a.Q$ in the proposed translation, and the actual representation will be of the form



(4)

For any number of τ, possibly 0, and some number n of actions a, where the actual representation will be of minimal form as shown in Theorem 1.4.3. The two process graphs have the same n, and therefore same number of rows.

Theorem 1.4.6

The two processes listed in Equation 4 are Branching Bisimilar

Proof. The two processes are Branching Bisimilar if we can construct a relation both ways.

($\Longrightarrow$): Show there is a branching bisimulation from the proposed translation to the actual representation. The τ transitions along the first row can be linked 1 to 1. For every action a_n to a state s, the only options from s are either a τ move, or a move to a_{n+1}.

- If s moves to a_{n+1}, then via Definition 1.4.1 this is valid via part $2b$ since there exists a $t = t_1 \xrightarrow{a_{n+1}} t'$, namely the corresponding state on the same row.
- If s moves to a τ, then via Definition 1.4.1 this is valid via part $2a$, since $a = \tau$, and all states on the same row are related to each other.

Therefore, a valid relation can be constructed from the proposed translation to the actual representation.

($\Longleftarrow$): Show there is a branching bisimulation from the actual representation to the proposed translation. Similarly to the converse, the transitions along the first row can be linked 1 to 1. For any action a_n to a state s, the only option from s is a move to a_{n+1}. This is clearly valid via the same logic as the first point in the converse argument, where any state corresponding to the same row can be chosen. Therefore, a valid relation can be constructed from the actual representation to the proposed translation.

Since both directions hold a valid translation, this must mean that the two processes are Branching Bisimilar. □

Therefore, since the resulting process graph satisfies the root condition, and via Lemma 1.4.4 and Theorem 1.4.6, we can conclude that the translation is valid up to Rooted Branching Bisimilarity.

References

[1] Rance Cleaveland and Oleg Sokolsky. CHAPTER 6 - Equivalence and Preorder Checking for Finite-State Systems. In J. A. Bergstra, A. Ponse, and S. A. Smolka, editors, *Handbook of Process Algebra*, pages 391–424. Elsevier Science, Amsterdam, January 2001.

[2] Wan Fokkink. Rooted Branching Bisimulation as a Congruence. *Journal of Computer and System Sciences*, 60(1):13–37, February 2000.

[3] Rob van Glabbeek. On the Expressiveness of ACP. In A. Ponse, C. Verhoef, and S. F. M. van Vlijmen, editors, *Algebra of Communicating Processes*, pages 188–217, London, 1995. Springer London.