Honours Project

Leon Lee

4th Year Project Report
Computer Science and Mathematics
School of Informatics
University of Edinburgh

2025

Abstract

This skeleton demonstrates how to use the `infthesis` style for undergraduate dissertations in the School of Informatics. It also emphasises the page limit, and that you must not deviate from the required style. The file `skeleton.tex` generates this document and should be used as a starting point for your thesis. Replace this abstract text with a concise summary of your report.

Research Ethics Approval

This project was planned in accordance with the Informatics Research Ethics policy. It did not involve any aspects that required approval from the Informatics Research Ethics committee.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Leon Lee*)

Acknowledgements

Any acknowledgements go here.

Table of Contents

Chapter 1

Introduction

Chapter 2

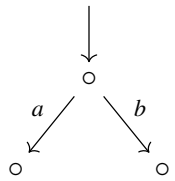
Background

2.1 Process Algebra

With the growing complexities of software and systems of the world, it is key to have methods of modelling more complex systems to get a better understanding of the underlying behaviour behind processes. Efforts have been made in sequential programming as early as the 1930s with Turing Machines, and the λ-calculus. Systems in real life are rarely sequential however, and usually involve multiple processes acting simultaneously, sometimes even synchronising to interact with each other to perform tasks. These tasks that involve modelling multiple processes at once are referred to as a *Concurrent System*. It is clear to see that brute forcing solutions to these problems are significantly harder than a sequential system - the processing time will grow exponentially as the number of processes increase, and modelling a system like a colony of ants is near impossible. Therefore, we will need some way to formalise these Concurrent Systems.

Concurrency has been studied in many different ways, though with the earliest the 1960s with some notable models being Petri nets, or the Actor Model. Process Algebras are one such method of modelling a Concurrent System, where the process is modelled in such a way that it is akin to the Universal Algebras of mathematics - in which operations are defined in an axiomatic approach to create a structurally sound way of defining concurrent systems. (Baeten, 2005) It is easily possible to model simple systems as a flow chart or diagram as you will be able to see throughout this paper, but a formal approach like process algebras will make way for modelling more complex systems, and lays the groundwork to provide a solid foundation to prove and base claims for such systems.

A simple example in action is a process algebra where we only consider the alternative composition operator $+$, where applied to a process $a+b$ means "Choose a, or choose b". Process algebras can typically be modelled in a *Process Graph*, which are diagrams that employ "states", and "actions" to show the traces, or paths, that a process can take. In this case, the process $a+b$ can be modelled in the following way:



Where the graph begins at the top into the first node, and then can either progress to the left node via the action a, or the right node via the action b. a and b are the actions, e.g. "eat" and "drink", while the nodes are the states, e.g. "apple" and "water"

The axioms of the $+$ operator of BPA are as follows:

- **Commutativity**: $a+b$
- **Associativity**: $(a+b)+c = a+(b+c)$
- **Idempotency**: $a+a = a$

Comparable to the operation axioms of a Group or Ring in Mathematics, every other operation in a process algebra is constructed similarly. In practice, most process algebras will have some form of alternative composition, but this is a very simplified example and the developed process algebras that exist are designed to handle a lot more complex situations such as unobservable actions, commonly referred to as τ-actions, recursion, which lets a process repeat itself or other processes, and deadlock, which is a state where no desirable outcomes can be reached.

There are many process algebras that exist, the most famous and seminal being CSP (Brookes et al., 1984), CCS (Milner, 1980), and ACP (Bergstra and Klop, 1984), (Bergstra and Klop, 1989), with some other popular calculi being the π-calculus and its various extensions (Milner et al., 1992), (Parrow and Victor, 1998), (Abadi and Gordon, 1999) which have been used to varying degrees in fields like Biology, Business, and Cryptography, or the Ambient Calculus (Cardelli and Gordon, 1998) which has been used to model mobile devices.

2.2 Encodings of Process Algebra

With the growing number of process algebras, one might begin to ask if there is a way of comparing different process algebra to each other to find the single best one, as a parallel to Turing Machines and the Church-Turing thesis. However, the wide range of applications that different process algebra are used for makes that rather impractical, and the goal of unifying all process algebra into a single theory seems further and further away as more process algebras for even more specified tasks get created.

A more reasonable approach is to compare different process algebras and their expressiveness, two main relevant methods being *absolute* and *relative* expressiveness.(Parrow, 2008) Absolute expressiveness is the idea of comparing a specific process algebra to a question and seeing if it can solve the problem - e.g. if a process algebra is Turing

Complete. However, this merely biparts different algebra - the process algebra that are able to solve a specified problem, and the ones who aren't (Gorla, 2010). Therefore, the question of relative expressiveness - i.e. how one language compares to another is a lot more useful in terms of categorising different process algebras by expressiveness.

A well studied way of comparing expressiveness is through an "encoding", and whether an algebra can be translated from one to another, but not vice versa (Peters, 2019). The general notion of an encoding is not defined by clear boundaries, and the criterion for a valid encoding may vary from language to language, but work has been made to try and generalise the notion of a "valid" encoding (Gorla, 2010), (van Glabbeek, 2018).

2.3 CSP

CSP (Communicating Sequential Processes) (Brookes et al., 1984) is a Process Algebra developed by Tony Hoare based on the idea of message passing via communications. It was developed in the 1980s and was one of the first of its kind, alongside CCS by Milner. CSP uses the idea of action prefixing which is where operators are of the syntax $a \rightarrow P$, where a is an event and P is a process.

As taken from van Glabbeek (2017), the syntax of CSP can be expressed as follows

$$\begin{aligned} P,Q ::= & \text{STOP} \mid \text{div} \mid \text{a} \rightarrow \text{P} \mid \text{P} \sqcap \text{Q} \mid \text{P} \Box \text{Q} \mid \text{P} \triangleright \text{Q} \mid \\ & P||Q \mid P\backslash A \mid f(P) \mid P \,\Theta_A\, Q \mid P \,\Theta_A\, Q \mid p \mid \mu p.P \end{aligned}$$

where the operators are: *inaction*, *divergence*, *action prefixing*, *internal choice*, *external choice*, *sliding choice*, *parallel composition*, *concealment*, *renaming*, *interrupt*, and *throw*.

2.4 ACP

ACP (Algebra of Communicating Processes) is a Process Algebra developed by Jan Bergstra and Jan Willem Klop (Bergstra and Klop, 1984). Compared to CSP, ACP is built up with an axiomatic approach in mind which does away with the idea of action prefixing and instead can allow for unguarded operations. ACP_τ (Bergstra and Klop, 1989) is an extension of ACP that includes an extra action τ which is used to represent actions that are unobservable, or changeable, from a human perspective.

The grammar of ACP_τ as taken from (Bergstra and Klop, 1989) is defined as such:

$$P,Q ::= a \mid \delta \mid E + F \mid E.F \mid E||F \mid E \underline{||} F \mid E|F \mid \partial_H(E) \mid \tau_I$$

where the operators are: *action*, *deadlock*, *alternative composition*, *sequential composition*, *merge*, *left merge*, *communication merge*, *encapsulation*, *abstraction*

[WIP]

Chapter 3

A formal definition of CSP and ACP_F^τ

3.1 Languages

From [EXPRESSIVENESS], we represent a language $\mathscr{L}$ as a pair $(\mathbb{T}, [\![\ ]\!])$, where $\mathbb{T}$ is a set of valid expressions in $\mathscr{L}$, and $[\![\ ]\!]$ is a mapping $[\![\ ]\!]c : \mathbb{T} \to \mathscr{D}$ from $\mathbb{T}$ to a set of meanings $\mathscr{D}$. We also define $A \subseteq \mathbb{T}$, where A is the set of actions

3.2 CSP

As stated above, our proposed grammar of CSP consists of the operations:

$$P,Q ::= \mathrm{STOP} \mid \mathrm{div} \mid a \to P \mid P \sqcap Q \mid P \Box Q \mid P \triangleleft Q \mid$$
$$P||_A Q \mid P\backslash A \mid f(P) \mid P \triangle Q \mid P \theta_A Q \mid$$

where the operators are: *inaction*, *divergence*, *action prefixing*, *internal choice*, *external choice*, *sliding choice*, *parallel composition*, *concealment*, *renaming*, *interrupt*, and *throw*.

These can also be represented in the following GSOS table

3.3 ACP_F^τ

$$P,Q ::= a \mid \delta \mid E + F \mid E.F \mid E \,||\, F \mid E \,\lfloor\!\lfloor\, F \mid E|F \mid \partial_H(E) \mid \tau_I$$

where the operators are: *action*, *deadlock*, *alternative composition*, *sequential composition*, *merge*, *left merge*, *communication merge*, *encapsulation*, *abstraction*

A proposed extension of ACP adds a Functional Renaming operator, as shown in [ON THE EXPRESSIVENESS OF ACP]. From this point forth, we will be using this extension, written as ACP_F^τ.

$\mathrm{div} \xrightarrow{\tau} \mathrm{div}$	$(a \to P) \xrightarrow{a} P$	$P \sqcap Q \xrightarrow{\tau} P$	$P \sqcap Q \xrightarrow{\tau} Q$
$\dfrac{P \xrightarrow{a} P'}{P \Box Q \xrightarrow{a} P'}$	$\dfrac{P \xrightarrow{\tau} P'}{P \Box Q \xrightarrow{\tau} P' \Box Q}$	$\dfrac{Q \xrightarrow{a} Q'}{P \Box Q \xrightarrow{a} Q'}$	$\dfrac{Q \xrightarrow{\tau} Q'}{P \Box Q \xrightarrow{\tau} P \Box Q'}$
$\dfrac{P \xrightarrow{a} P'}{P \triangleright Q \xrightarrow{a} P'}$	$\dfrac{P \xrightarrow{\tau} P'}{P \triangleright Q \xrightarrow{\tau} P' \triangleright Q}$	$P \triangleright Q \xrightarrow{\tau} Q$	$\dfrac{P \xrightarrow{\alpha} P'}{f(P) \xrightarrow{f(\alpha)} f(P')}$
$\dfrac{P \xrightarrow{\alpha} P' \ (\alpha \notin A)}{P \Vert_A Q \xrightarrow{\alpha} P' \Vert_A Q}$	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{a} Q' \ (a \in A)}{P \Vert_A Q \xrightarrow{a} P' \Vert_A Q'}$		$\dfrac{Q \xrightarrow{\alpha} Q' \ (\alpha \notin A)}{P \Vert_A Q \xrightarrow{\alpha} P \Vert_A Q'}$
$\dfrac{P \xrightarrow{\alpha} P' \ (\alpha \notin A)}{P \backslash A \xrightarrow{\alpha} P' \backslash A}$	$\dfrac{P \xrightarrow{a} P' \ (a \in A)}{P \backslash A \xrightarrow{\tau} P' \backslash A}$	$\dfrac{P \xrightarrow{\alpha} P' \ (\alpha \notin A)}{P \,\Theta_A\, Q \xrightarrow{a} P' \,\Theta_A\, Q}$	$\dfrac{P \xrightarrow{a} P' \ (a \in A)}{P \,\Theta_A\, Q \xrightarrow{a} Q}$
$\dfrac{P \xrightarrow{\alpha} P'}{P \triangle Q \xrightarrow{\alpha} P' \triangle Q}$	$\dfrac{Q \xrightarrow{\tau} Q'}{P \triangle Q \xrightarrow{\tau} P' \triangle Q'}$	$\dfrac{Q \xrightarrow{a} Q'}{P \triangle Q \xrightarrow{a} Q'}$	$\mu p.P \xrightarrow{\tau} P[\mu p.P/p]$

Table 3.1: Structural operational semantics of CSP

$(a.P) \xrightarrow{\alpha} P$	$P + Q \xrightarrow{\alpha} P$	$P + Q \xrightarrow{\alpha} Q$
$\dfrac{P \xrightarrow{\alpha} P'}{P \,\Vert\, Q \xrightarrow{\alpha} P' \,\Vert\, Q}$	$\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad a \mid b = c}{P \,\Vert\, Q \xrightarrow{a} P' \,\Vert\, Q'}$	$\dfrac{Q \xrightarrow{\alpha} Q'}{P \,\Vert\, Q \xrightarrow{\alpha} P \,\Vert\, Q'}$
$\dfrac{P \xrightarrow{\alpha} P' \ (\alpha \notin A)}{\partial_H(P) \xrightarrow{\alpha} \partial_H(P')}$	$\dfrac{\langle S_X \mid \mathcal{S} \rangle \xrightarrow{a} P'}{\langle X \mid \mathcal{S} \rangle \xrightarrow{a} P'}$	

Table 3.2: Structural operational semantics of ACP

3.4 Expressiveness

Somethin something we are trying to gain an expressiveness result by translating CSP to ACP_τ. A result of a valid translation would therefore show that CSP is *at least as expressive* as ACP_τ.

Chapter 4

A Translation of CSP to ACP_F^τ

4.1 Direct Translations

Some of the basic operations of CSP have an identical equivalence in ACP, with the only difference being the syntax. These can be easily translated in the following table.

$$\begin{aligned}\mathscr{T}(STOP) &= \delta \\ \mathscr{T}(a \to P) &= a.\mathscr{T}(P) \\ \mathscr{T}(P\backslash A) &= \partial_A \mathscr{T}(P)\end{aligned}$$

4.2 Trivial Translations

- **Divergence** is the process that diverges via infinite internal actions. It is defined by the following rule:

  $$\mathrm{div} \xrightarrow{\tau} \mathrm{div}$$

  and then can be directly translated via recursion in ACP in the following rule:

  $$\mathscr{T}(\mathrm{div}) = \langle X \mid X = \tau.X \rangle$$

- **Renaming** is an operation that renames actions in processes according to a function. There is no equivalent function in plain ACP_τ, with the closest operation being $\tau_I(P)$ which abstracts actions in I to internal actions. This is possible in ACP_F^τ, and in fact our translation is trivially

  $$\mathscr{T}(f(P)) = f(\mathscr{T}(P))$$

- **Internal Choice** is an operation that emulates a choice of actions that cannot be decided by the user. CSP in particular differs from ACP in that external choice and internal choice are separate operations, while in ACP, the alternative choice operator $+$ handles choice, albeit slightly differently. With the internal choice operator τ, a translation for CSP Internal choice into ACP is easily written as

  $$\mathscr{T}(P \sqcap Q) = \tau.\mathscr{T}(P) + \tau.\mathscr{T}(Q)$$

The above translations are all valid up to Strong Bisimilarity. The other operators are slightly harder to translate.

4.3 Helper Operators for the language ACP_F^τ

4.3.1 Subsets of A

Working in the language ACP_τ with the extension of Functional Renaming (written ACP_F^τ), we start by defining some subsets of A which we will use in our encodings.

Definition 4.3.1.1: Subsets of A

The set $A \in \mathbb{T}$ is the set of all actions.

- $A_0 \subseteq A$ is the set of actions that actually get used in processes
- $A_T \subseteq A$ is a set of target actions. This is used in operators such as CSP Parallel Composition, which only communicates over a set.
- $H_0 = A - A_0$ is the set of working space operators, or any other action that doesn't get used
- $H_1 = A_0 \uplus \mathscr{H}$ is the set of actions, plus a set of working operators $\mathscr{H}$

In general, $A_T \subseteq A_0 \subseteq H_1 \subseteq A$.

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

4.3.2 Triggering

We define an operator $\Gamma(P)$ that emulates the Triggering operator of MEIJE [REFER]. For a trace $a.b.\cdots$ on a process P, the triggering operator can be represented as an operator that tags the first action of a process.

First, we define a function $f_{\texttt{trig}}$ and communications for the operations `first` and `next`.

Def 4.3.2.1: Communications

Define communications where

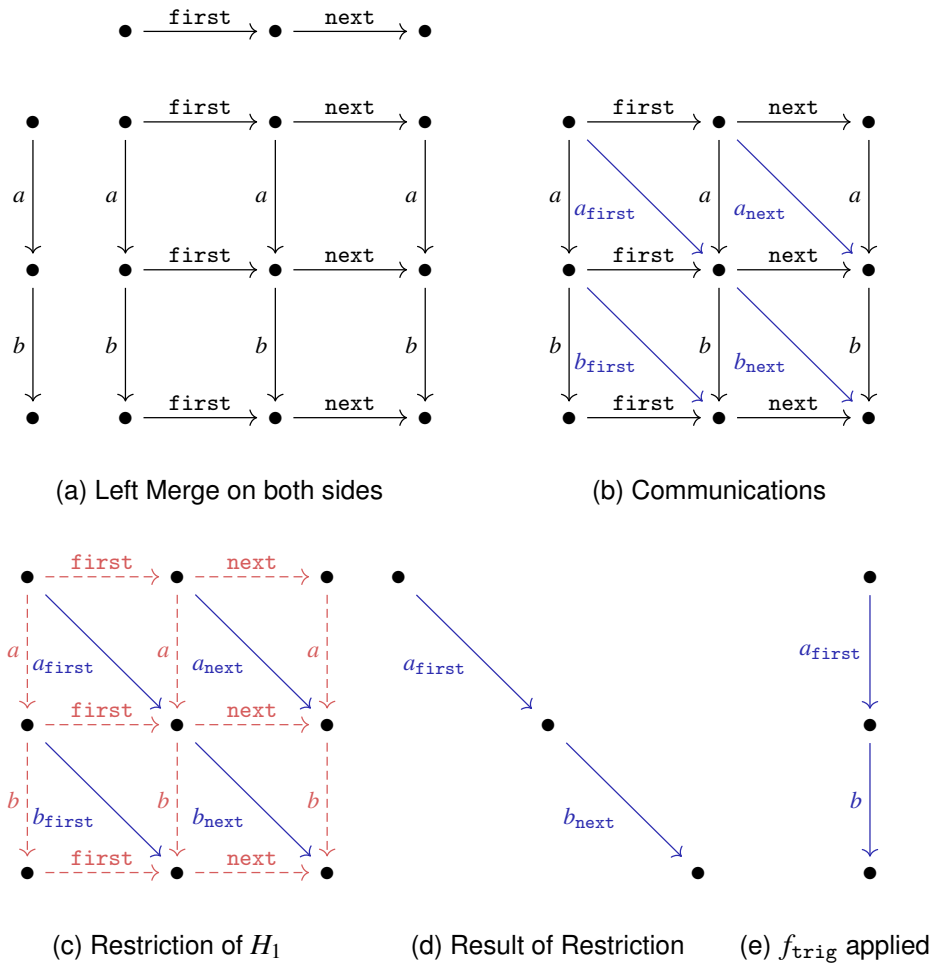
$$a|\texttt{first} = a_{\texttt{first}}$$
$$a|\texttt{next} = a_{\texttt{next}}$$

Definition 4.3.2.2: F1

Define functions $f_{\texttt{trig}} : A \to A$ where

$$f_{\texttt{trig}}(a_{\texttt{first}}) = a_{\texttt{ini}}$$
$$f_{\texttt{trig}}(a_{\texttt{next}}) = a$$

We use the notation of a^∞ as syntactic sugar to mean $\langle X \mid X = a.X \rangle$. Using the sets defined in 4.3.1.1, we can now define $\Gamma(P)$ as such:



(a) Left Merge on both sides (b) Communications

(c) Restriction of H_1 (d) Result of Restriction (e) $f_{\mathtt{trig}}$ applied

Figure 4.1: Example of $\Gamma(P)$ applied to $P = a.b.c$

Definition 4.3.2.3: Triggering in ACP

$$\Gamma(P) := f_{\mathtt{trig}}\left[\partial_{H_1}(P||\mathtt{first}.(\mathtt{next}^\infty))\right]$$

is an operator that turns a trace of a process P, $a.b.c\ldots$ into the trace

$$a_{\mathtt{ini}}.b.c\ldots.$$

This works in the following method:

a) Merge the process P with the process $\mathtt{first}.\mathtt{next}.\mathtt{next}\ldots$. Via Def 4.3.2.1, this will produce a lattice of P and $\mathtt{first}.(\mathtt{next}^\infty)$, with communications on every square, but most importantly, a chain of communications going down the centre of the form.

$$a_{\mathtt{first}}.b_{\mathtt{next}}.c_{\mathtt{next}}\ldots \tag{4.1}$$

b) Restrict the actions in H_1. Since all the actions in $P, \mathtt{first}.(\mathtt{next}^\infty) \in H_1$ this effectively restricts both sides of the left merge, leaving only communications

from the initial state. This leaves equation 4.1 as the only remaining trace.

c) Apply $f_{\mathtt{trig}}$ to equation 4.1. Via 4.3.2.2, the final result is

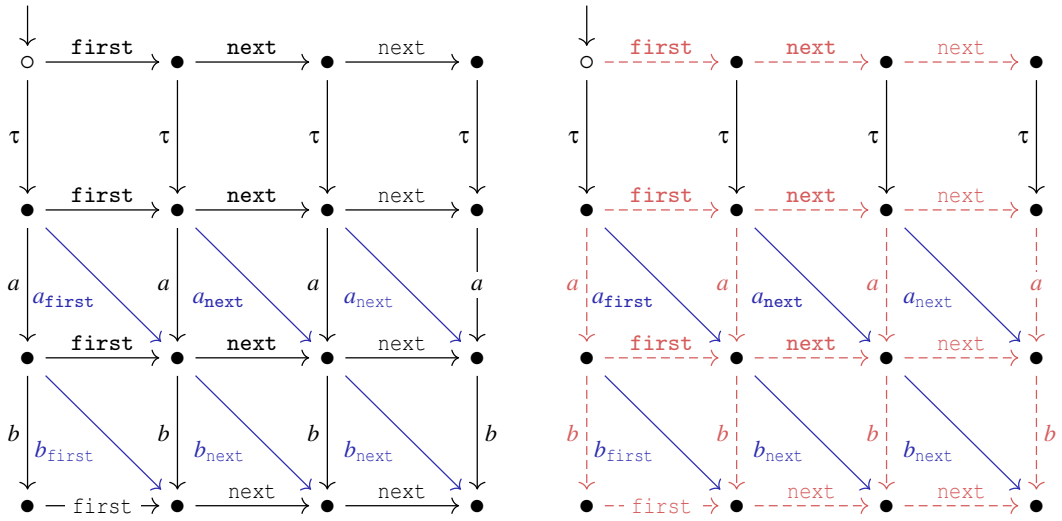
$$a_{\mathtt{ini}}.b.c\ldots \tag{4.2}$$

The process is now exactly as stated in Definition 4.3.2.3.

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

$$\tau.b_{\mathtt{ini}}.c\ldots.$$

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



(a) Left merge with τ communications

(b) Restriction of H_1 with τ actions

Figure 4.2: Example of Triggering operator with internal actions

4.3.3 Associativity and Postfix Function

From the axioms of ACP_τ (Bergstra and Klop, 1989), we have the following axioms of associativity with the communication operator $|$ shown in the table below:

Communication function in ACP_τ
$a\|b = b\|a$
$(a\|b)\|c = a\|(b\|c)$
$\delta\|a = \delta$

Table 4.1: Axioms of ACP_τ Communication

Bearing the above axioms in mind, we have the potential to run into problems with this with our communications. For example, in the proposed translation for the External Choice operator $\Box$ (4.4.3), a simplified version of the translation would have the following communications:

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

This might work at first glance, but the Associativity axiom does not hold true in this case, such as in the following counterexample:

$$\begin{aligned} a|\texttt{first}|\texttt{choose} &= (a|\texttt{first})|\texttt{choose} = a_{\texttt{first}}|\texttt{choose} = a \\ &= a|(\texttt{first}|\texttt{choose}) = a|\delta = \delta \end{aligned}$$

From this example, it is clear that the proposed communications would not satisfy the axioms of ACP_τ. It is for this reason that in 4.3.2.2, we have the rule

$$f_{\texttt{trig}}(a_{\texttt{first}}) = a_{\texttt{ini}} \qquad \text{instead of} \qquad f_{\texttt{trig}}(a_{\texttt{first}}) = a_{\texttt{first}}$$

A preferred communications function is as follows:

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

$$\begin{aligned} a|\texttt{first}|\texttt{choose} &= (a|\texttt{first})|\texttt{choose} = a_{\texttt{first}}|\texttt{choose} = \delta \\ &= a|(\texttt{first}|\texttt{choose}) = a|\delta = \delta \end{aligned}$$

It is important to note that this scenario would never actually occur in practice, since $\Gamma(P)$ takes precedence and hence `first|choose` would never happen, but the communication function must work over every action regardless of whether it will get used in practice.

Lemma 4.3.3.1: Keeping Associativity in Communications

For three actions $a, b, c \in A$, if $a|b = c$, then if c does not appear on the left-hand side of any other communication, Associativity will be satisfied.

Proof. WIP. QED □

We define a compatibility function to prevent issues with associativity. Via 4.3.3.1, define a tag $a_{\mathtt{post}}$ which will act as c for all our communications, which satisfies the Lemma. Then, our final step will be to rename a_{post} back to a for any affected actions. This works in the following way:

$$a \xrightarrow{\text{Rename for Communication}} a_{\mathtt{tag}} \xrightarrow{\text{Communicate with an action}} a_{\mathtt{post}} \xrightarrow{\text{Rename for final result}} a$$

Definition 4.3.3.2: Postfix function

Let $f_{\text{post}} : A \to A$ where

$$f_{\text{post}}(a_{\mathtt{post}}) = a \quad f_{\text{post}}(\alpha) = \alpha$$

4.4 Translations for the remaining CSP Operators

4.4.1 Communications and Functional Renaming

We define communications for our translation in addition to the ones previously defined for our helper functions. These are defined as follows:

Definition 4.4.1.1: Helper Functions

In addition to the function $f_{\mathtt{trig}}$ defined in 4.3.2.2, and the postfix function defined in 4.3.3.2:

$$f_{\mathtt{trig}}(a_{\mathtt{first}}) = a_{\mathtt{ini}} \qquad f_{\mathtt{trig}}(a_{\mathtt{next}}) = a \qquad f_{\mathrm{post}}(a_{\mathtt{post}}) = a \qquad f_{\mathrm{post}}(\alpha) = \alpha$$

We define functions for the remaining operators below. We use the notation A_T to signify a target set, as used in 4.4.2, 4.4.5, and λ to signify actions in A_T

1. $f_{\mathtt{syn}} : A \to A$ is a function that renames any actions in the target set A_T. This is used in the translation of Parallel Composition (4.4.2)
$$f_{\mathtt{syn}}(\lambda) = \lambda_{\mathtt{syn}} \quad f_{\mathtt{syn}}(\alpha) = \alpha$$
2. $f_{\mathtt{origin}} : A \to A$ is a function that renames actions in a process for use in operators. This is used in the translation of the Stopping operator (4.4.4)
$$f_{\mathtt{origin}}(a) = a_{\mathtt{origin}}$$
3. $f_{\mathtt{split}} : A \to A$ is a function that renames actions in a process for use in operators, and also renames actions in the target set A_T. This is used in the translation of the Interrupt operator (4.4.5)
$$f_{\mathtt{split}}(\lambda) = \lambda_{\mathtt{split}} \qquad f_{\mathtt{split}}(a) = a_{\mathtt{origin}}$$

Definition 4.4.1.2: Communications

In addition to the communications for the Triggering operator defined in 4.3.2.1

$$a|\mathtt{first} = a_{\mathtt{first}} \qquad a|\mathtt{next} = a_{\mathtt{next}}$$

we define additional communications for the functions defined above:

1. $a_{\mathtt{syn}}|a_{\mathtt{syn}} = a_{\mathtt{post}}$. This is used in the translation of Parallel Composition (4.4.2)
2. $a_{\mathtt{ini}}|\mathtt{choose} = a_{\mathtt{post}}$. This is used in the translation of External choice (4.4.3)
3. $a_{\mathtt{origin}}|\mathtt{origin} = a_{\mathtt{post}}$. This is used in the translation of Stopping and Interrupt operator (4.4.4, 4.4.5)
4. $a_{\mathtt{ini}}|\mathtt{split} = a_{\mathtt{post}}$. This is used in the translation of the Stopping operator (4.4.5)
5. $a_{\mathtt{split}}|\mathtt{split} = a_{\mathtt{post}}$. This is used in the translation of the Interrupt operator (4.4.5)

4.4.2 Parallel Composition

The parallel composition $||_A$ is defined with the following rules:

$$\frac{P \xrightarrow{\alpha} P' \quad (\alpha \notin A)}{P||_A Q \xrightarrow{\alpha} P'||_A Q} \qquad \frac{Q \xrightarrow{\alpha} Q' \quad (\alpha \notin A)}{P||_A Q \xrightarrow{a} P||_A Q'} \qquad \frac{P \xrightarrow{a} P' \; Q \xrightarrow{a} Q' (\alpha \in A)}{P||_A Q \xrightarrow{a} P'||_A Q'} \tag{4.3}$$

In other words, a left merge action can be taken by all actions of P and Q, which is the same as the ACP_F^τ equivalent of parallel composition with the one difference being that in CSP, the action must be the same in P and Q, whereas in ACP_F^τ the action is defined with a communication function. For our encoding, we take $\mathscr{H} = \{\}$, and therefore $H_1 = A_0$. The goal is to tag actions in the target set A_T, and then define a communication function between identical marked actions. We can do this via the following functions and communications:

Definition 4.4.2.1: Functions and Communications - Parallel Composition

As defined in 4.4.1.1 and 4.4.1.2, the following communications and functions are defined over Parallel Composition:

$$f_{\texttt{syn}}(\lambda) = \lambda_{\texttt{syn}} \quad f_{\texttt{syn}}(\alpha) = \alpha \qquad f_{\text{post}}(a_{\texttt{post}}) = a \qquad f_{\text{post}}(\alpha) = \alpha$$

$$a_{\texttt{syn}} | a_{\texttt{syn}} = a_{\texttt{post}}$$

A translation for Parallel Composition can then be written as the following:

$$\mathscr{T}(P||_A Q) = \partial_{H_0}(f_{\text{post}}(f_{\texttt{syn}}(\mathscr{T}(P))||f_{\texttt{syn}}(\mathscr{T}(Q))))$$

4.4.3 External choice

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

$$\frac{P \xrightarrow{a} P'}{P \Box Q \xrightarrow{a} P'} \qquad \frac{Q \xrightarrow{a} Q'}{P \Box Q \xrightarrow{a} Q'} \qquad \frac{P \xrightarrow{\tau} P'}{P \Box Q \xrightarrow{\tau} P' \Box Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P \Box Q \xrightarrow{a} P \Box Q'} \tag{4.4}$$

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_F^τ Alternative Choice operator $(+)$, as $+$ will not let you select externally if an internal action is made. For our encoding, we take $\mathscr{H} = \{\texttt{first}, \texttt{next}, \texttt{choose}\}$, and therefore $H_1 = A_0 \uplus \{\texttt{first}, \texttt{next}, \texttt{choose}\}$ as defined in Definition 4.3.1.1.

Definition 4.4.3.1: Functions and Communications - External Choice

As defined in 4.4.1.1 and 4.4.1.2, the following communications and functions are defined over External Choice:

$$f_{\texttt{trig}}(a_{\texttt{first}}) = a_{\texttt{ini}} \qquad f_{\texttt{trig}}(a_{\texttt{next}}) = a \qquad f_{\text{post}}(a_{\texttt{post}}) = a \qquad f_{\text{post}}(\alpha) = \alpha$$

$$a_{\texttt{ini}}|\texttt{choose} = a_{\texttt{post}}$$

Additionally, recall that the Triggering operator (4.3.2.3) is defined as:

$$\Gamma(P) := f_{\texttt{trig}}[\partial_{H_1}(P||\texttt{first}(\texttt{next}^\infty))])$$

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

$$\mathscr{T}(P\Box Q) = \partial_{H_0}\Big(f_2\Big[\Gamma[\mathscr{T}(P)]\,||\,\texttt{choose}\,||\,\Gamma[\mathscr{T}(Q)]\Big]\Big)$$

4.4.4 Stopping

The stopping operator $\triangle$ is defined with the following rules:

$$\frac{P \xrightarrow{\alpha} P'}{P\triangle Q \xrightarrow{\alpha} P'\triangle Q} \qquad \frac{Q \xrightarrow{\tau} Q'}{P\triangle Q \xrightarrow{\tau} P\triangle Q'} \qquad \frac{Q \xrightarrow{a} Q'}{P\triangle Q \xrightarrow{a} Q'}$$

In other words, we can take an external choice from P without interrupting the operator, in addition to internal choices from Q. However, the moment an external choice is made from Q, the process can then never return to P.

For our encoding, we take $\mathscr{H} = \{\texttt{first}, \texttt{next}, \texttt{origin}, \texttt{split}\}$, and therefore $H_1 = A_0 \uplus \{\texttt{first}, \texttt{next}, \texttt{origin}, \texttt{split}\}$ as defined in 4.3.1.1.

Definition 4.4.4.1: Functions and Communications - Stopping

As defined in 4.4.1.1 and 4.4.1.2, the following communications and functions are defined over External Choice:

$$f_{\texttt{trig}}(a_{\texttt{first}}) = a_{\texttt{ini}} \qquad f_{\texttt{trig}}(a_{\texttt{next}}) = a \qquad f_{\text{post}}(a_{\texttt{post}}) = a \qquad f_{\text{post}}(\alpha) = \alpha$$

$$f_{\texttt{origin}}(a) = a_{\texttt{origin}}$$

$$a_{\texttt{origin}}|\texttt{origin} = a_{\texttt{post}} \qquad a_{\texttt{ini}}|\texttt{split} = a_{\texttt{post}}$$

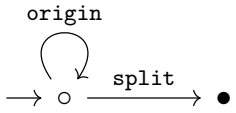
Additionally, recall that the Triggering operator (4.3.2.3) is defined as:

$$\Gamma(P) := f_{\texttt{trig}}[\partial_{H_1}(P||\texttt{first}(\texttt{next}^\infty))])$$

We can now define an encoding of the CSP Stopping operator $\triangle$ in ACP_F^τ. We start off with a new process, which we will call Π. This is defined as the process

$$\Pi = \langle X \mid X = \texttt{origin}.X + \texttt{split}\rangle$$

Or, visualised as a process graph:



From this, an encoding can be written in the following way:

$$\mathscr{T}(P \triangle Q) = \partial_{H_0}\Big(f_{\texttt{post}}\Big[\big(f_{\texttt{origin}}(\mathscr{T}(P))||\Pi\big)||\Gamma(\mathscr{T}(Q))\Big]\Big)$$

4.4.5 Interrupt

The interrupt operator Θ_A is defined with the following rules:

$$\frac{P \xrightarrow{\alpha} P' \quad (a \notin A)}{P\,\Theta_A\,Q \xrightarrow{\alpha} P'\,\Theta_A\,Q} \qquad \frac{P \xrightarrow{a} P' \quad (a \in A)}{P\,\Theta_A\,Q \xrightarrow{a} Q}$$

In other words, we can take as many actions in P as we want, as long as they aren't contained in a set of actions, which we will call A_T. However, the moment an action in A_T is made, The process then diverts to Q. This can be also throught as an error checking operator. Similarly to the stopping operator, for our encoding, we take

$$\mathscr{H} = \{\texttt{first}, \texttt{next}, \texttt{origin}, \texttt{split}\}$$

, and therefore $H_1 = A_0 \uplus \{\texttt{first}, \texttt{next}, \texttt{origin}, \texttt{split}\}$ as defined in 4.3.1.1.

Definition 4.4.5.1: Functions and Communications - Interrupt

As defined in 4.4.1.1 and 4.4.1.2, the following communications and functions are defined over External Choice:

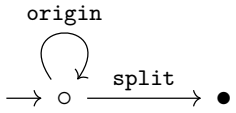
$$f_{\text{post}}(a_{\texttt{post}}) = a \qquad f_{\text{post}}(\alpha) = \alpha \qquad f_{\texttt{split}}(\lambda) = \lambda_{\texttt{split}} \qquad f_{\texttt{split}}(a) = a_{\texttt{origin}}$$

$$a_{\texttt{origin}}|\texttt{origin} = a_{\texttt{post}} \qquad a_{\texttt{split}}|\texttt{split} = a_{\texttt{post}}$$

We can now define an encoding of the CSP Interrupt operator Θ_A in ACP_F^τ. We employ the use of the same operator as in the Stopping operator, Π which is defined as

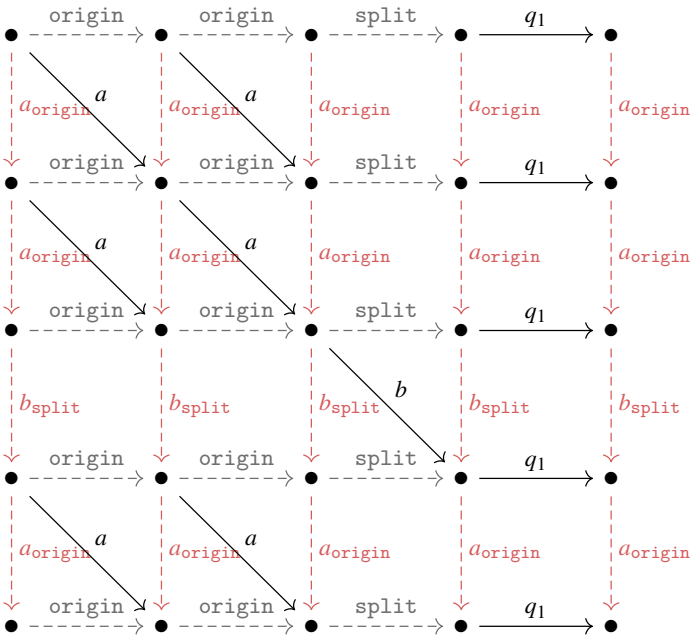
$$\Pi = \langle X \mid X = \texttt{origin}.X + \texttt{split} \rangle$$

Or, visualised as a process graph:



From this, an encoding can be written in the following way:

$$\mathscr{T}(P\,\Theta_A\,Q) = \partial_{H_0}\Big(f_{\texttt{post}}\Big[f_{\texttt{split}}(\mathscr{T}(P))||\Pi.\mathscr{T}(Q)\Big]\Big)$$



4.4.6 Sliding Choice

The sliding choice, or sliding operator $\triangleright$ is defined with the following rules:

$$\frac{P \xrightarrow{a} P'}{P \triangleright Q \xrightarrow{a} P'} \qquad \frac{P \xrightarrow{\tau} P'}{P \triangleright Q \xrightarrow{\tau} P' \triangleright Q} \qquad P \triangleright Q \xrightarrow{\tau} Q$$

In other words, this operator lets you take an external action on P, however there is a second process that may at any point before the external action taken in P "time out" and move to Q instead.

WIP

4.4.7 Final Translation

$$P,Q ::= \text{STOP} \mid \text{div} \mid a \to P \mid P \sqcap Q \mid P \Box Q \mid P \triangleleft Q \mid$$
$$P||_A Q \mid P\backslash A \mid f(P) \mid P \triangle Q \mid P \theta_A Q \mid$$

We define our translation

Definition 4.4.7.1: Translation of CSP to ACP

Let $\mathbb{T}_{\text{CSP}}$ be the expressions in the language CSP, and $\mathbb{T}_{\text{ACP}_F^\tau}$ be expressions in the language ACP_F^τ. We define a translation $\mathscr{T} : \mathbb{T}_{\text{CSP}} \to \mathbb{T}_{\text{ACP}_F^\tau}$ defined as such:

$$\begin{aligned}
\mathscr{T}(STOP) &= \delta \\
\mathscr{T}(\text{div}) &= \langle X \mid X = \tau.X \rangle \\
\mathscr{T}(a \to P) &= a.\mathscr{T}(P) \\
\mathscr{T}(P \sqcap Q) &= \tau.\mathscr{T}(P) + \tau.\mathscr{T}(Q) \\
\mathscr{T}(P \Box Q) &= \partial_{H_0}\Big(f_{\texttt{post}}\Big[\Gamma[\mathscr{T}(P)] \,||\, \texttt{choose} \,||\, \Gamma[\mathscr{T}(Q)]\Big]\Big) \\
\mathscr{T}(P||_A Q) &= \partial_{H_0}\Big(f_{\texttt{post}}\Big[f_{\texttt{syn}}(\mathscr{T}(P))||f_{\texttt{syn}}(\mathscr{T}(Q))\Big]\Big) \\
\mathscr{T}(P\backslash A) &= \partial_A \mathscr{T}(P) \\
\mathscr{T}(f(P)) &= f(\mathscr{T}(P)) \\
\mathscr{T}(P \triangle Q) &= \partial_{H_0}\Big(f_{\texttt{post}}\Big[\big((f_{\texttt{origin}}\mathscr{T}(P))||\Pi\big)||\Gamma(\mathscr{T}(Q))\Big]\Big) \\
\mathscr{T}(P \Theta_A Q) &= \partial_{H_0}\Big(f_{\texttt{post}}\Big[(f_{\texttt{split}}\mathscr{T}(P))||\Pi.\mathscr{T}(Q)\Big]\Big)
\end{aligned}$$

Chapter 5

Validity of the Encoding

Referring back to our translation 4.4.7.1, the translation for External Choice

$$\partial_{H_0}(f_{\texttt{post}}[\Gamma(P)\,||\,\texttt{choose}\,||\,\Gamma(Q)])$$

has identical behaviour to $P+Q$ when dealing with processes with only external actions. However, on processes with internal actions, the translation is not so trivial. The addition of an internal action works in the translation's favour for actions such as deferring a `start` tag to the first visible action (see 4.2), but it can also backfire, as the translation largely relies on removing unwanted left-merges and communications through restriction operators. As internal actions are non-interactable it means restrictions that *should* remove all remaining actions will end up with unwanted left-over τ moves.
For example, if we take the process $a\Box\tau.b \in \text{CSP}$, the resulting translation in ACP_F^τ is

$$\partial_{H_0}\Big(f_{\texttt{post}}\Big[\Gamma(a)\,||\,\texttt{choose}\,||\,\Gamma(\tau.b)\Big]\Big) = \partial_{H_0}\Big(f_{\texttt{post}}\Big[a_{\texttt{ini}}\,||\,\texttt{choose}\,||\,\tau.b_{\texttt{ini}}\Big]\Big)$$

which has the following process graph:

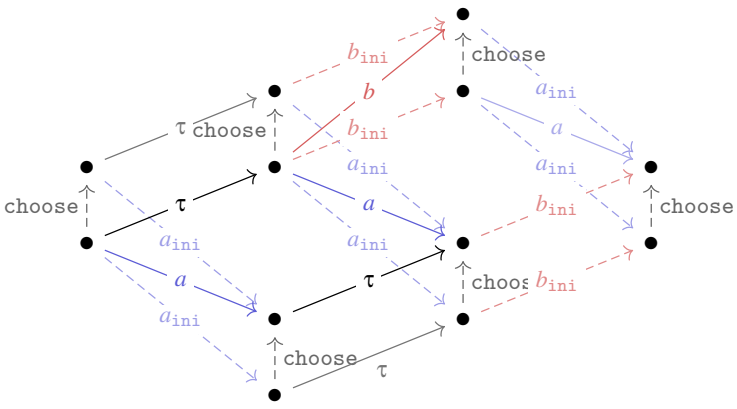


Figure 5.1: Counterexample for Strong Bisimilarity with the processes $P = a$ and $Q = \tau.b$. The result of the translation is $a.\tau + \tau.(a+b) \not\leftrightarrows a + \tau.(a+b)$

5.1 Rooted Branching Bisimilarity

We define formally the definition of Rooted Branching Bisimulation via (Baeten and Weijland, 1990):

Definition 5.1.0.1: 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 (Fokkink, 2000).

We can now work towards a proof that the external choice is Rooted Branching Bisimilar.

From 4.4.7.1, the translation of external choice is:

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

Using the following communications from 4.4.3.1:

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

And the postfix function defined in 4.3.3.2,

Lemma 5.1.0.2

For a process $P \in \text{ACP}_F^\tau$, if there exists P' such that $P \xrightarrow{\tau} P'$ then application of the Triggering operator shown in 4.3.2.3 can be applied as $\Gamma[P] \xrightarrow{\tau} \Gamma[P']$. Alternatively,

$$\Gamma[P] = \tau.\Gamma[P']$$

Proof. We have that $P = \tau.P'$ and therefore $\Gamma[P] = \Gamma[\tau.P']$. However, since internal actions cannot interact with anything inside the triggering operator, the resulting function is the same as if the τ was instead outside the function. A diagram is provided in 4.2. □

Lemma 5.1.0.3

For a process $P \in \text{ACP}_F^\tau$, if there exists P' such that $P \xrightarrow{a} P'$ then for any action $a \in A_0$ we have

$$(\Gamma[P] \,||\, \texttt{choose}) \xrightarrow{a} P' \quad \text{and} \quad (\texttt{choose} \,||\, (\Gamma[P]) \xrightarrow{a} P'$$

Proof. Directly follows from communications □

Lemma 5.1.0.4

For processes $a.P \in \text{ACP}_F^\tau$ where $a \in A_0$, if $\exists b.Q$ where $b \in A_0$ we have

$$\partial_{H_0}(b.Q \,||\, \Gamma[a.P]) = b.Q$$

Alternatively, for processes $\tau^*.P \in \text{ACP}_F^\tau$, where τ^* indicates a chain of consecutive τ, possibly 0, if there exists a process $b.Q$ where $b \in A_0$ we have

$$\partial_{H_0}(b.Q \,||\, \tau^*.\Gamma[P]) = b.Q \,||\, \tau^*$$

Proof. The Triggering operator Γ turns a trace into a renamed trace of the form

$$a_{\texttt{ini}}.b.c \cdots$$

The process $a_{\texttt{ini}}$ does not communicate with anything other than `choose` which is not in A_0. Therefore, the restriction operator will remove all the Γ left merges, leaving $b.Q$. However, if there are internal actions that cannot interact with restrictions and communications, then the final process will be a communication with any remaining internal actions before the first cut off external action. □

Lemma 5.1.0.5

For a process $P \in \text{CSP}$ where $\mathscr{T}(P) \in \text{ACP}_F^\tau$ is strongly bisimilar to P, a process $\tau^* \,||\, \mathscr{T}(P)$ is Branching Bisimilar to the process P. Here, I use the notation of τ^* indicating a chain of τ, possibly 0.

Proof. We work inductively for each number of τ. using the definition of Branching Bisimilarity (5.1.0.1).

Base Cases:

- $P =_{BB} \mathscr{T}(P)$. By our initial assumption, $P \leftrightarrows \mathscr{T}(P)$ which is a strictly finer equivalence, therefore $P =_{BB} \mathscr{T}(P)$ trivially
- $P =_{BB} \tau.\mathscr{T}(P)$. If $P \xrightarrow{a} P'$, for $a \in A_0$ (CSP has no action τ), then indeed there exists $\tau.\mathscr{T}(P) \xrightarrow{\tau} \mathscr{T}(P) \xrightarrow{a} \mathscr{T}(P')$, and the relations hold due to the bisimilarity of P. The converse is also true, and if $\tau.\mathscr{T}(P) \xrightarrow{\tau} \mathscr{T}(P)$ then we indeed have that $a = \tau$ and $\mathscr{T}(P) =_{BB} P$

Inductive Step: For a process $\mathcal{P} = \tau^n.\mathscr{T}(P)$, where n indicates a certain number of consecutive τ-moves, we assume that $P =_{BB} \mathcal{P}$ holds. Now given $\tau.\mathcal{P}$, we want to show this is also Branching Bisimilar to P. If $P \xrightarrow{a} P'$ for $a \in A_0$ then there exists

$$\tau.\mathcal{P} \xrightarrow{\tau} \mathcal{P} \xRightarrow{n} \mathscr{T}(P) \xrightarrow{a} \mathscr{T}(P')$$

Where both the proceeding trace, and relations hold from the inductive hypothesis. This can also be more succinctly written as

$$\tau.\mathcal{P} \xRightarrow{n+1} \mathscr{T}(P) \xrightarrow{a} \mathscr{T}(P')$$

Similarly to the $n = 1$ base case, the converse is also true, and if $\tau.\mathscr{T}(P) \xrightarrow{\tau} \mathscr{T}(P)$ then we indeed have that $a = \tau$ and $\mathscr{T}(P) =_{BB} \mathcal{P}$ which is true from the inductive hypothesis. □

5.1.1 Proof of Rooted Branching Bisimilarity [WIP]

We define a bisimulation relation.

Definition 5.1.1.1: Rooted Branching Bisimulation Relation

Let $\mathbb{T}_{\text{CSP}}$ be the expressions in the language CSP, and $\mathbb{T}_{\text{ACP}_F^\tau}$ be expressions in the language ACP_F^τ. We use the translation $\mathscr{T} : \mathbb{T}_{\text{CSP}} \to \mathbb{T}_{\text{ACP}_F^\tau}$ as defined in 4.4.7.1.

We now define a Rooted Branching Bisimulation between $\mathbb{T}_{\text{CSP}}$ and $\mathbb{T}_{\text{ACP}_F^\tau}$:

$$\mathcal{B} := \{P, \mathscr{T}(P) \mid P \in \text{CSP}\}$$

Proof. Let $P, Q \in \text{CSP}$ be two processes. We want to show that $P \square Q =_{RBB} \mathscr{T}(P \square Q)$. i.e.: we want to show that any move will result in a process that satisfies RBB. We prove this inductively by showing that any smaller process of P is also Rooted Branching Bisimilar.

We exhaust all possible moves that $P \square Q$ can take, and confirm that $\mathscr{T}(P \square Q)$ can also take them, up to Rooted Branching Bisimilarity.

Case 1: a action on P. Let $a \in A_0$ and P' such that $P \xrightarrow{a} P'$. In the domain of CSP, this results in the process

$$P \square Q \xrightarrow{a} P'$$

Now working in ACP_F^τ, we want to show that the translation is valid up to RBB, i.e. $\mathscr{T}(P') =_{RBB} P'$. Via Lemma 5.1.0.3, we can derive the following equation

$$\mathscr{T}(P \square Q) = \partial_{H_0}\Big(f_{\texttt{post}}\Big[\Gamma[\mathscr{T}(P)] \,||\, \texttt{choose} \,||\, \Gamma[\mathscr{T}(Q)]\Big]\Big) \xrightarrow{a} \partial_{H_0}\Big(f_{\texttt{post}}\Big[\mathscr{T}(P') \,||\, \Gamma[\mathscr{T}(Q)]\Big]\Big)$$

This is not yet a process that is comparable to P', so we look at the next step.

1. **Case 1A**: $\exists b \in A_0$ such that $Q \xrightarrow{b} Q'$. Then via Lemma 5.1.0.4, this results in

   $$\partial_{H_0}\Big(f_{\texttt{post}}\Big[\mathscr{T}(P') \,||\, \Gamma[\mathscr{T}(Q)]\Big]\Big) = \partial_{H_0}(f_{\texttt{post}}[\mathscr{T}(P')]) = \mathscr{T}(P')$$

   Which is strongly bisimilar.

2. **Case 1B**: $Q \xrightarrow{\tau} Q'$. Then via Lemma 5.1.0.2, we can derive the following equation:

   $$\partial_{H_0}\Big(f_{\texttt{post}}\Big[\mathscr{T}(P') \,||\, \Gamma[\mathscr{T}(Q)]\Big]\Big) = \partial_{H_0}\Big(f_{\texttt{post}}\Big[\mathscr{T}(P') \,||\, \tau.\Gamma[\mathscr{T}(Q')]\Big]\Big)$$

   Case 1B can be repeated as many times as needed until the first action of Q is an external one. Via Lemma 5.1.0.4, this results in

   $$\partial_{H_0}\Big(f_{\texttt{post}}\Big[\mathscr{T}(P') \,||\, \tau^*.\Gamma[\mathscr{T}(Q''')]\Big]\Big) = \partial_{H_0}(f_{\texttt{post}}[\mathscr{T}(P') \,||\, \tau^*]) = \mathscr{T}(P') \,||\, \tau^*$$

   Which is Branching Bisimilar to P' via Lemma 5.1.0.5

The first action (a) can happen Strongly on the root, and any further actions are either also Strongly Bisimilar, or Branching Bisimilar. Therefore, an a action on P is Rooted Branching Bisimilar as required.

Case 2: τ action on P. Let P' such that $P \xrightarrow{\tau} P'$. In the domain of CSP, this results in the process

$$P \Box Q \xrightarrow{\tau} P' \Box Q$$

Now working in ACP_F^τ, we want to show that the translation is valid up to RBB, i.e. $\mathscr{T}(P' \Box Q) =_{RBB} P' \Box Q$. Via Lemma 5.1.0.2, we can now derive the following equation

$$\partial_{H_0}\left(f_1\left[\Gamma[\mathscr{T}(P)] \,||\, \texttt{choose} \,||\, \Gamma[\mathscr{T}(Q)]\right]\right) \xrightarrow{\tau}$$
$$\partial_{H_0}\left(f_1\left[\Gamma[\mathscr{T}(P')] \,||\, \texttt{choose} \,||\, \Gamma[\mathscr{T}(Q)]\right]\right)$$

Which is strongly bisimilar to $P' \Box Q$, therefore a τ action on P is also valid up to Rooted Branching Bisimilarity.

Case 3, Case 4: The same logic from option 1 and option 2 can be applied to Q and P to obtain processes that satisfy Rooted Branching Bisimilarity.

We have now exhausted all cases, and therefore can conclude that our translation of CSP External Choice is Rooted Branching Bisimilar, and therefore a Congruence. □

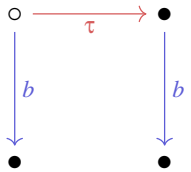
5.1.2 Stopping

Similarly to external choice, this is also Rooted Branching Bisimilar.

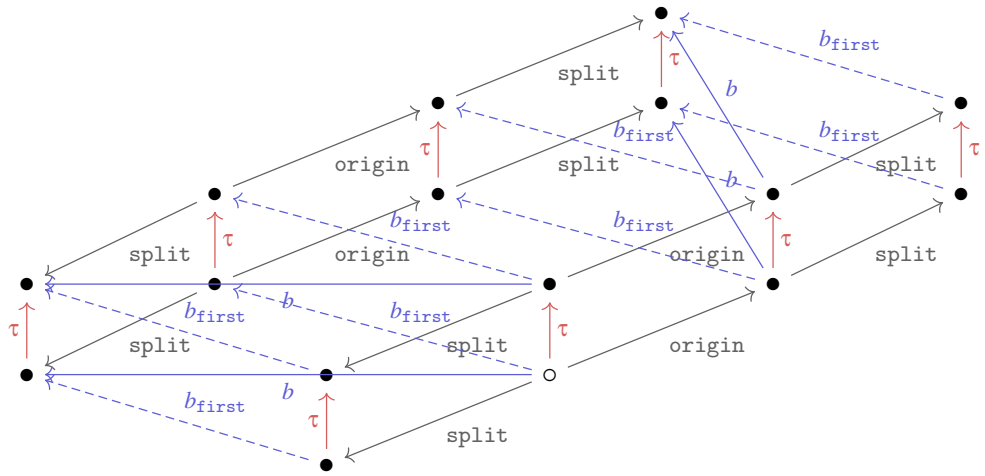
A counterexample to the encoding being Strongly Bisimilar is with the trivial example

$$P = \tau, Q = b$$

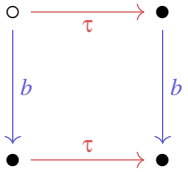
This should yield the following process graph:



However, it yields the following



Which reduces down to:



5.1.3 Generalising

Lemma 5.1.3.1: Maybe?

I claim that a Strong Bisimulation cannot occur and RBB is the finest equivalence able to be translated

Proof. Something about how taus cannot be renamed / communicated. Have not thought of it past there but it feels like it might be right as all the translations have the same issue. □

Chapter 6

Results

Chapter 7

Conclusions

Bibliography

Martín Abadi and Andrew D. Gordon. A Calculus for Cryptographic Protocols: The Spi Calculus. *Information and Computation*, 148(1):1–70, January 1999. ISSN 0890-5401. doi: 10.1006/inco.1998.2740.

J. C. M. Baeten. A brief history of process algebra. *Theoretical Computer Science*, 335 (2):131–146, May 2005. ISSN 0304-3975. doi: 10.1016/j.tcs.2004.07.036.

J. C. M. Baeten and W. P. Weijland. *Process Algebra*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, 1990. doi: 10.1017/CBO9780511624193.

J. A. Bergstra and J. W. Klop. Process algebra for synchronous communication. *Information and Control*, 60(1):109–137, January 1984. ISSN 0019-9958. doi: 10.1016/S0019-9958(84)80025-X.

J. A. Bergstra and J. W. Klop. ACPτ a universal axiom system for process specification. In Martin Wirsing and Jan A. Bergstra, editors, *Algebraic Methods: Theory, Tools and Applications*, pages 445–463, Berlin, Heidelberg, 1989. Springer Berlin Heidelberg. ISBN 978-3-540-46758-8.

S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A Theory of Communicating Sequential Processes. *J. ACM*, 31(3):560–599, June 1984. ISSN 0004-5411. doi: 10.1145/828.833.

Luca Cardelli and Andrew D. Gordon. Mobile ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computation Structures*, pages 140–155, Berlin, Heidelberg, 1998. Springer. ISBN 978-3-540-69720-6. doi: 10.1007/BFb0053547.

Wan Fokkink. Rooted Branching Bisimulation as a Congruence. *Journal of Computer and System Sciences*, 60(1):13–37, February 2000. ISSN 0022-0000. doi: 10.1006/jcss.1999.1663.

Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Information and Computation*, 208(9):1031–1053, September 2010. ISSN 0890-5401. doi: 10.1016/j.ic.2010.05.002.

Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, Berlin, Heidelberg, 1980. ISBN 978-3-540-10235-9 978-3-540-38311-6. doi: 10.1007/3-540-10235-3.

Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Information and Computation*, 100(1):1–40, 1992. ISSN 0890-5401. doi: 10.1016/0890-5401(92)90008-4.

J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings. Thirteenth Annual IEEE Symposium on Logic in Computer Science (Cat. No.98CB36226)*, pages 176–185, June 1998. doi: 10.1109/LICS.1998.705654.

Joachim Parrow. Expressiveness of Process Algebras. *Electronic Notes in Theoretical Computer Science*, 209:173–186, April 2008. ISSN 1571-0661. doi: 10.1016/j.entcs.2008.04.011.

Kirstin Peters. Comparing Process Calculi Using Encodings. *Electron. Proc. Theor. Comput. Sci.*, 300:19–38, August 2019. ISSN 2075-2180. doi: 10.4204/EPTCS.300.2.

Rob van Glabbeek. A Branching Time Model of CSP. In Thomas Gibson-Robinson, Philippa Hopcroft, and Ranko Lazić, editors, *Concurrency, Security, and Puzzles: Essays Dedicated to Andrew William Roscoe on the Occasion of His 60th Birthday*, pages 272–293. Springer International Publishing, Cham, 2017. ISBN 978-3-319-51046-0. doi: 10.1007/978-3-319-51046-0_14.

Rob van Glabbeek. A theory of encodings and expressiveness (extended abstract) - (extended abstract). In Christel Baier and Ugo Dal Lago, editors, *Foundations of Software Science and Computation Structures - 21st International Conference, FOSSACS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings*, volume 10803 of *Lecture Notes in Computer Science*, pages 183–202. Springer, 2018. doi: 10.1007/978-3-319-89366-2_10.

Appendix A

Diagrams