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

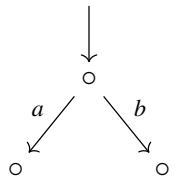
Background

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

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

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*

Chapter 3

A formal definition of CSP and ACP

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 $[\![\ ]\!] : \mathbb{T} \to \mathscr{D}$ from $\mathbb{T}$ to a set of meanings $\mathscr{D}$. We also define $A \subseteq \mathbb{T}$, where A is the set of actions

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

Chapter 4

A Translation of CSP to ACP

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

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

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

[INSERT TABLE HERE]

As they are in GSOS format, these operations are compositional in CSP. For a valid translation into ACP, we will want the resulting translation to be compositional as well.

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.

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

4.2 Trivial Translations

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

$$\text{div} \xrightarrow{\tau} \text{div}$$

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

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

- **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.

  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. A clear translation is then shown to be

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

- **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 ACP_F^τ

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 4.3.0.1: 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
- $H_0 = A - A_0$ 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, $A_0 \subseteq H_1 \subseteq A$.

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

4.3.1 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_1 and communications for the operations `first` and `next`.

Definition 4.3.1.1: F1

Define functions $f_1 : A \to A$ where

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

Def 4.3.1.2: Communications

Define communications where

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

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

Definition 4.3.1.3: 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, $a.b.c.\ldots$ into the trace

$$a_{\texttt{first}}.b.c.\ldots$$

This works in the following method:

1. Merge the process P with the process $\texttt{first}.\texttt{next}.\texttt{next}\ldots$. Via Def 4.3.1.2, 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.
$$a_{\texttt{first}}.b_{\texttt{next}}.c_{\texttt{next}}\cdots \tag{4.1}$$
2. Restrict the actions in H_1. Since all the actions in $P, \texttt{first}.(\texttt{next}^\infty) \in H_1$ this effectively restricts both sides of the left merge, leaving only communications from the initial state. This leaves equation 4.1 as the only remaining trace.
3. Apply ρ_{f_1} to equation 4.1. Via 4.3.1.1, the final result is
$$a_{\texttt{first}}.b.c\ldots \tag{4.2}$$

   The process is now exactly as stated in Definition 4.3.1.3.

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 $P = \tau.b.c\ldots$ becoming the trace

$$\tau.b_{\texttt{first}}.c.\ldots$$

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

Chapter 5

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. 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.

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

First appendix

A.1 First section

Any appendices, including any required ethics information, should be included after the references.

Markers do not have to consider appendices. Make sure that your contributions are made clear in the main body of the dissertation (within the page limit).