university of groningen

faculty of science and engineering

The Leader Election Problem in a Typed Pi-calculus

Bachelor's Project Computing Science

July 2024

Author: Barnabas Tarcali
Student Number: S4360923
First Supervisor: prof. dr. J. A. Pérez Parra.
Second Supervisor: Dr. D. Frumin.

Abstract

There are many versions of the pi-calculus, the calculus of interaction and concurrency. However, only a few variants based in linear logic can express the non-determinism of the full (untyped) pi-calculus while ensuring deadlock freedom. In this paper, we examine the expressive power of $\mathsf{s}\pi^+$, a typed pi-calculus in which well-typed processes are deadlock-free by construction. We aim at establishing that $\mathsf{s}\pi^+$ is as expressive as the full pi-calculus by considering the *leader election problem*, i.e., by exhibiting a well-typed system that describes a symmetric elective network of size five. This document details five attempts at modeling the leader election in $\mathsf{s}\pi^+$: each attempt tries to solve the main challenge of the previous one, while revealing limitations of the typing system. Our results are negative: they provide evidence of the trade-off between expressiveness and strong correctness properties for typed processes. We conclude by identifying three key factors that limit the expressiveness of $\mathsf{s}\pi^+$.

Contents

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 Aim of the Paper . . . 3
- 1.3 Motivation . . . 4
- 1.4 Proposal . . . 4
- 1.5 Paper Structure . . . 4

2 Background **5**
- 2.1 Linear Logic . . . 5
- 2.2 Pi-calculus . . . 6
- 2.3 Propositions-as-sessions . . . 7
- 2.4 $s\pi^+$, an existing pi-calculus . . . 7

3 Expressiveness of Different Calculi **10**
- 3.1 Expressiveness . . . 10
- 3.2 Leader election problem . . . 11

4 Results and Discussion **13**
- 4.1 First Attempt . . . 13
- 4.2 Second Attempt . . . 14
- 4.3 Third Attempt . . . 15
- 4.4 Fourth Attempt . . . 16
- 4.5 Fifth Attempt . . . 19

5 Conclusion **21**

6 Future Work **22**

List of Figures

1 π-calculus: syntax (top) and reduction rules (bottom) . . . 6
2 Types, their interpretations (up), and their duals (down) . . . 7
3 Typing rules for $s\pi^+$ . . . 9
4 Reduction rules for “valued” some and none . . . 14
5 Partial typing . . . 19

1 Introduction

1.1 Background

As the number of critical systems that use software increases, producing correct programs becomes more and more important. One approach is thorough testing in software development, but to quote E. W. Dijkstra: "Program testing can be used to show the presence of bugs, but never to show their absence!" [1]. Another option is to produce programs that are correct by design; when we talk about critical systems, it is important that we can prove that they behave as intended.

In the case of sequential programs, we already have some options for ensuring correctness by design: Hoare logic provides one system that allows us to reason about correctness with the use of Hoare triples. The lambda-calculus, another logical framework, uses functions to express programs, and its rules make it easy to use induction when reasoning about correctness. Strictly sequential programs, however, are quite rare in practice; moreover, verification tools for sequential programs cannot be used on concurrent and parallel programs. When considering the correctness in concurrent systems, a well-established approach is based on *process calculi*, a family of formal languages able to model parallelism. Each calculus in this set provides algebraic rules that enable compositional modeling and analysis; this makes them an attractive tool to effectively reason about program correctness. The pi-calculus is arguably the most well-known and developed member of the process calculi family.

What lambda-calculus is for functional programming is what the pi-calculus aspires to be for parallel programming, namely a precise mathematical description, with a minimal set of rules and operations, that can effectively express all parallel programs. This in itself is an ambitious task, but there are multiple variants of the pi-calculus, each slightly different and tailored to analyze certain concrete problems. They proved to be an effective tool for reasoning about the correctness of parallel programs, especially when coupled with so-called *session types*. The mathematical foundation of these types is now known to originate from Girard's *linear logic*, a resource-aware logic. This allows us to use tools of this logical system to justify important correctness properties for processes, with the emphasis being on *communication safety* (absence of communication errors) and *deadlock-freedom* (absence of "stuck" process configurations).

1.2 Aim of the Paper

The focus of this paper is to examine whether an existing version of typed pi-calculus is expressive enough to describe the *leader election problem* and, more specifically, an elective network of five processes. The project can be fitted in a main area of CS: Theoretical Computer Science, more precisely Programming Language Theory and Formal Methods. The research question that we will answer is: "Can a symmetric leader election problem for five

processes be described as a well-typed process in $\mathsf{s}\pi^+$?". We chose this calculus as our benchmark because it is a typed pi-calculus that guarantees deadlock-freedom by construction, as its type system is rooted in linear logic.

1.3 Motivation

As elective networks are widely used (e.g. in databases), this project could provide solid foundations to already existing or new implementations. The focus of this project is on the understanding of types, the interplay between concepts such as *linearity* and *non-determinism*, and the possible limitations in expressiveness that come from using linear logic as a base for our parallel systems.

1.4 Proposal

There is a number of preexisting research articles that present solutions to the leader election problem, but in an untyped setting. We will use these works as the starting point for our work. The proposed project will make use of the tools of experimental mathematics, in the following way. We take the existing solutions for the problem from a different version of the pi-calculus and rewrite it in a typed system. Once we have done this, we examine it and look for typing or reduction rule conflicts. We are hopefully left with a typeable expression by eliminating these one by one, by rewriting the expressions in different formats. This process will lead to a better understanding of the problem, and to discover the necessary properties to express it. If a language is developed that can describe an elective network, a comparative analysis will be carried out, to assess which features are the ones that enable it to do so.

We expect three possible outcomes: a *positive result* would be finding a well-typed system in $\mathsf{s}\pi^+$ that describes the leader election problem; a *semi-positive result* is if we find a process that is typeable under some minor assumptions; and a *negative result* is if we do not find a well-typed program, and we cannot identify assumptions that make it typeable, but we can explain the restricting factors.

1.5 Paper Structure

Section 2 provides the foundational knowledge that is necessary to understand the related works and the research question. In Section 3 we provide an overview of relevant previous research papers, that separate both typed and not typed pi-calculi by their expressive powers. Section 4 describes attempts and results with a focus on what types actually do, and some possible limiting factors in using them. Section 5 presents and debates the outcome of the research project. Finally, Section 6 presents questions that emerged during our work, with possible directions for future research.

2 Background

To help understanding the context of the project, we provide an overview of relevant fields. We will build heavily upon these foundational pieces.

2.1 Linear Logic

In most logical settings (in the following we will take *intuitionistic logic* as an example) truth is free. In other words, once a judgement is derived it can be freely used in the proof, as many times as one wants. However, not all things are free or unlimited, therefore when we want to reason about resources (e.g. money) we have to default to a different logical system and linear logic (introduced by Girard in [2]) provides a good framework for us.

We chose intuitionistic logic as an example of traditional logical system with a good reason. As Curry and Howard discovered, proofs in this logic correspond to function types in functional programs. It also means that proofs can be run and checked automatically. This landmark theoretical result is named Curry-Howard correspondence (or isomorphism) [3]. The following judgement holds in intuitionistic logic (here $A \to B$ means 'from A it follows that B', and $A \times B$ is 'A and B'):

$$A, A \to B \vdash A \times B \tag{1}$$

Note how A is used twice in this proof and because in this setting truth does not have a cost, it is valid.

Things work differently in linear logic. As mentioned earlier, we have to carefully monitor resource usage. We cannot have our money and spend it too, this is captured by the following:

$$A, A \multimap B \nvdash A \otimes B \tag{2}$$

where $A \multimap B$ reads as 'consuming A yields B' and $A \otimes B$ as 'both A and B'. If we assume that for 10 Euros we can buy a pizza and we know that we have 10 Euros, we can deduce, that we will either have the 10 Euros or a pizza, but not both:

$$A, A \multimap B \vdash A \& B \tag{3}$$

Linear logic has two types of disjunctions and conjunctions: additive and multiplicative. The multiplicative conjunct is the connective $\otimes$, the additive conjunct of A and B is $A \& B$ (this means A or B, but we have a choice over which one). The types of disjunctions work similarly: the multiplicative one ($A \parr B$) describes that exactly one of A or B will happen, but we do not know which one, and we have to be ready for both; whereas the additive ($A \oplus B$) means the alternative occurrence of A and B, a choice which we have no control over.

$P, Q ::= \mathbf{0}$	inaction	$\mid x(y); P$	input
$\mid \overline{x}[y]; P$	output	$\mid \overline{x}.\ell; P$	select
$\mid x.\mathtt{case}\{i : P_i\}_{i \in I}$	branch	$\mid (\boldsymbol{\nu} x)(P \mid Q)$	connect
$\mid P + Q$	non-determinism	$\mid P \mid Q$	parallel

$$(\boldsymbol{\nu} x)(\overline{x}[a]; P + M_1 \mid x(y); Q + M_2) \to (\boldsymbol{\nu} x)(P \mid Q\{a/y\}) \quad \text{communication}$$

$$\overline{x}.\ell; P \mid x.\mathtt{case}\{i : Q_i\}_{i \in I} \to P \mid Q_\ell, \text{ where } \ell \in I \quad \text{selection}$$

Figure 1: π-calculus: syntax (top) and reduction rules (bottom)

2.2 Pi-calculus

When talking about formal description of parallelism, we have to mention the process calculi family. Process calculi are precise mathematical models, that provide a set of algebraic rules with which we can express a wide variety of behaviours.

The pi-calculus is a member of this family. It treats parallelism and the communication between parallel processes as message-passing algorithms. Parallel process communicate through channels (each channel provides two-way communication option for two processes), with the sending of names (these can be values, names of channels, etc.). When it was introduced by Milner, Parrow, and Walker, the motivation was to create the 'lambda-calculus of parallelism' [4]. This meant a minimal set of simple rules that can still describe potentially complex systems.

Usually in the pi-calculus $P, Q, \ldots$ are used to denote processes, and $x, y, \ldots$ are used for channel names, therefore we will use this as well. Figure 1 describes the syntax (top) of a simple pi-calculus. $\mathbf{0}$ means inaction (the process that can take no action anymore), $x(y)$ waits for an input y on channel x, conversely $\overline{x}[y]$ sends a name y along channel x. $\overline{x}.\ell$ chooses a label ℓ, whereas $i : P_i.\mathtt{case}_{i \in I}$ offers branches with labels from I. $(\boldsymbol{\nu} x)(P \mid Q)$ connects the processes P and Q with the channel x, and restricts its usage to those two processes. $P + Q$ describes the process that can behave as P or Q. $P \mid Q$ denotes two parallel processes, that do not interact with each other.

On Figure 1(bottom) we can see the reduction rules (atomic steps in the computation). The rule 'communication' describes how message-passing works (here $Q\{a/y\}$ denotes the substitution of y for a in Q). It also encapsulates the rule for non-determinism. It states that the process can arbitrarily choose to behave as P (and Q), but the other parts (M_1, M_2) are discarded. 'Selection' provides a rule for choice; $\overline{x}.\ell$ selects the label ℓ on x, and in turn $i : Q_i.\mathtt{case}_{i \in I}$ continues with the process Q_ℓ, that had the label ℓ.

A, B ::= X	propositional variable	$X^{\perp}/\overline{X}$	dual of a variable
$A \otimes B$	'tensor', output A then behave as B	$A \parr B$	'par', input A then behave as B
$A \oplus B$	'plus', select from A and B	$A \& B$	'with', offer choice from A and B
1	unit for $\otimes$	$\perp$	unit for $\parr$

$$\overline{\mathbf{1}} = \perp \qquad \overline{A \otimes B} = \overline{A} \parr \overline{B} \qquad \overline{A \oplus B} = \overline{A} \& \overline{B}$$
$$\overline{\perp} = \mathbf{1} \qquad \overline{A \parr B} = \overline{A} \otimes \overline{B} \qquad \overline{A \& B} = \overline{A} \oplus \overline{B}$$

Figure 2: Types, their interpretations (up), and their duals (down)

2.3 Propositions-as-sessions

Session types or propositions-as-sessions connect two seemingly non-related fields. Just as the Curry-Howard isomorphism provides a solid foundation in logic for lambda-calculus, and for functional programming, this correspondence establishes a similar link between linear logic and process calculi. In this case propositions can be viewed as session types, proofs as processes, and communication as cut elimination.

The type of a name or channel describes the communication protocol on said channel. The typing of a process P is the collection of the types of the domain of P, meaning all free names that occur in P. As we want to ensure communication safety, we restrict channels as a two way channel between exactly two processes. With this restriction, we can expect that if we have processes P and Q, that communicate on channel x the type of x in P has to be the dual of the type of x in Q. In other words, if P sends an integer on x and then closes it, our system respects communication safety only if Q waits for an integer on x, then waits for closure.

We provide the interpretation of types in linear logic in terms of communication protocol [5] on Figure 2 (up), together with the formal definition of duality (down). Because this interpretation relates linear logic to a process calculus, the symbols of linear logic get "new" meanings.

2.4 $s\pi^+$, an existing pi-calculus

In this paper we will examine a calculus that provides non-confluent non-determinism, the $s\pi^+$, as described in [6]. To understand the main differences between this calculus and the full pi-calculus we have to explain the following relation: ($\bowtie$). (i) $\overline{x}[y] \bowtie \overline{x}[z]$, (ii) $x(y) \bowtie x(z)$, and (iii) $\alpha \bowtie \alpha$ otherwise. This allows us to match outputs and inputs that happen on the same channel. The (simplified) precongurence $P \succeq_x Q$ (on channel x is defined as the following [6]: $P = \left(\Vdash_{i \in I} \mathtt{C_i}[\alpha_i; P_i]\right) \Vdash \left(\Vdash_{j \in J} \mathtt{C_j}[\beta_j; Q_j]\right)$ and $Q = \Vdash_{i \in I} \mathtt{C_i}[\alpha_i; P_i]$, where
(i) $\forall i, i' \in I.\, \alpha_i \bowtie \alpha_{i'}$ and $subj\{\alpha_i\} = \{x\}$, and
(ii) $\forall i \in I. \forall j \in J.\, \alpha_i \not\bowtie \beta_j \wedge x \in fn(\beta_j; Q_j)$;

This definition talks about how in the case of non-deterministic composition we can discard branches. P and Q contain matching prefixes on x, while P may contain additional branches with different or blocked prefixes on x; x must appear in the hole of the contexts in the additional branches in P (enforced with $x \in \mathit{fn}(\ldots)$), to ensure that no matching prefixes are discarded.

This precongurance is what dictates most of the reduction rules in $\mathsf{s}\pi^+$. For more technical properties, consult the paper from Heuvel, Paulus, Nantes-Sobrinho, and Pérez.

Another important difference is that we can signal/wait for errors and unavailability in $\mathsf{s}\pi^+$. This is done by the messages $\overline{x}.\mathtt{some}, \overline{x}.\mathtt{none}, x.\mathtt{some}_{\tilde{\omega}}$. $\overline{x}.\mathtt{some}$ announces availability on channel x, conversely $\overline{x}.\mathtt{none}$ means an error signal on x. $x.\mathtt{some}_{\tilde{\omega}}; P$ waits for a signal on x; if it is available, it continues as P, however if $\overline{x}.\mathtt{none}$ is recieved, P is discarded, and the error is propagated by $\overline{\tilde{\omega}}.\mathtt{none}$ (where $\tilde{\omega}$ is the set of the names appearing in P). Let us illustrate this with an example:

$$
\begin{aligned}
\mathsf{P}_{\mathsf{Alice}} &:= \overline{x}[\mathit{title}]; x.\mathtt{some}_y; x(\mathit{book}); \overline{x}[]; \overline{y}.\mathtt{some}; \overline{y}[\mathit{book}]; \overline{y}[] \\
\mathsf{P}_{\mathsf{Brian}} &:= y.\mathtt{some}; y(\mathit{book}); y() \\
\mathsf{P}_{\mathsf{Server}} &:= x(\mathit{title}); (\overline{x}.\mathtt{none} \nparallel \overline{x}.\mathtt{some}; \overline{x}[\mathit{book}]; x()) \\
\mathsf{Sys} &:= (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(\mathsf{P}_{\mathsf{Alice}} \mid \mathsf{P}_{\mathsf{Brian}}) \mid \mathsf{P}_{\mathsf{Server}})
\end{aligned}
$$

Here, Alice wants to request a book from the server, then she wants to send it to Brian. However, not all books are available. So Alice sends the title to the server, the server then (non-deterministically) signals the (un)availability of the book back to Alice. First consider when the book is available:

$$
\begin{aligned}
\mathsf{P}_{\mathsf{Alice}} &:= x(\mathit{book}); \overline{x}[]; \overline{y}.\mathtt{some}; \overline{y}[\mathit{book}]; \overline{y}[] \\
\mathsf{P}_{\mathsf{Brian}} &:= y.\mathtt{some}; y(\mathit{book}); y() \\
\mathsf{P}_{\mathsf{Server}} &:= \overline{x}[\mathit{book}]; x() \\
\mathsf{Sys} &:= (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(\mathsf{P}_{\mathsf{Alice}} \mid \mathsf{P}_{\mathsf{Brian}}) \mid \mathsf{P}_{\mathsf{Server}})
\end{aligned}
$$

The server synchronized with its available side, sending a '*some*' signal to Alice. The rest is straight-forward, Alice will receive the book on x and forward it to Brian on y. But what happens when the book is not available?

$$
\begin{aligned}
\mathsf{P}_{\mathsf{Alice}} &:= x.\mathtt{some}_y; x(\mathit{book}); \overline{x}[]; \overline{y}.\mathtt{some}; \overline{y}[\mathit{book}]; \overline{y}[] \\
\mathsf{P}_{\mathsf{Brian}} &:= y.\mathtt{some}; y(\mathit{book}); y() \\
\mathsf{P}_{\mathsf{Server}} &:= \overline{x}.\mathtt{none} \\
\mathsf{Sys} &:= (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(\mathsf{P}_{\mathsf{Alice}} \mid \mathsf{P}_{\mathsf{Brian}}) \mid \mathsf{P}_{\mathsf{Server}})
\end{aligned}
$$

The server signals the error to Alice, who in turn, will discard her "normal" continuation,

and will propagate the error to Brian:

$$\begin{aligned} \mathsf{P}_{\mathsf{Alice}} &:= \overline{y}.\mathtt{none} \\ \mathsf{P}_{\mathsf{Brian}} &:= y.\mathtt{some}; y(book); y() \\ \mathsf{P}_{\mathsf{Server}} &:= \mathbf{0} \\ \mathsf{Sys} &:= (\boldsymbol{\nu} x)((\boldsymbol{\nu} y)(\mathsf{P}_{\mathsf{Alice}} \mid \mathsf{P}_{\mathsf{Brian}}) \mid \mathsf{P}_{\mathsf{Server}}) \end{aligned}$$

This will cancel the continuation for Brian, and all processes will terminate.

The typing rules of $\mathsf{s}\pi^+$ (Figure 3) are based on the previously introduced propositions-as-sessions view. Rule T$\parallel$ is introduced to type non-deterministic compositions (the two sides have to be typeable under the same context, to ensure communication safety and deadlock freedom), and the rules T&SOME and T&NONE type processes that send (un)available signals. To type a process that waits for a signal T$\oplus$SOME is introduced, here the subsequent actions in P have to be typeable under the & monad, meaning the type of every channel in P (except for x) has to be some $\&A$.

$$[\textsc{Tcut}]\ \frac{P \vdash \Gamma, x{:}A \qquad Q \vdash \Delta, x{:}\overline{A}}{(\boldsymbol{\nu} x)(P \mid Q) \vdash \Gamma, \Delta} \qquad [\textsc{Tmix}]\ \frac{P \vdash \Gamma \qquad Q \vdash \Delta}{P \mid Q \vdash \Gamma, \Delta} \qquad [\mathrm{T}\!\parallel]\ \frac{P \vdash \Gamma \qquad Q \vdash \Gamma}{P \parallel Q \vdash \Gamma}$$

$$[\textsc{Tempty}]\ \frac{}{\mathbf{0} \vdash \emptyset} \qquad [\mathrm{T}\mathbf{1}]\ \frac{}{\overline{x}[] \vdash x{:}\mathbf{1}} \qquad [\mathrm{T}\bot]\ \frac{P \vdash \Gamma}{x(); P \vdash \Gamma, x{:}\bot} \qquad [\mathrm{T}\otimes]\ \frac{P \vdash \Gamma, y{:}A \qquad Q \vdash \Delta, x{:}B}{\overline{x}[y]; (P \mid Q) \vdash \Gamma, \Delta, x{:}A \otimes B}$$

$$[\mathrm{T}\parr]\ \frac{P \vdash \Gamma, y{:}A, x{:}B}{x(y); P \vdash \Gamma, x{:}A \parr B} \qquad [\mathrm{T}\oplus]\ \frac{P \vdash \Gamma, x{:}A_j \qquad j \in I}{\overline{x}.j; P \vdash \Gamma, x{:}\oplus\{i : A_i\}_{i \in I}}$$

$$[\mathrm{T}\&]\ \frac{\forall i \in I.\ P_i \vdash \Gamma, x{:}A_i}{x.\mathtt{case}\{i : P_i\}_{i \in I} \vdash \Gamma, x{:}\&\{i : A_i\}_{i \in I}} \qquad [\mathrm{T}\&\mathtt{some}]\ \frac{P \vdash \Gamma, x{:}A}{\overline{x}.\mathtt{some}; P \vdash \Gamma, x{:}\&A}$$

$$[\mathrm{T}\&\mathtt{none}]\ \frac{}{\overline{x}.\mathtt{none} \vdash x{:}\&A} \qquad [\mathrm{T}\oplus\mathtt{some}]\ \frac{P \vdash \&\Gamma, x{:}A}{x.\mathtt{some}_{dom(\Gamma)}; P \vdash \&\Gamma, x{:}\oplus A}$$

Figure 3: Typing rules for $\mathsf{s}\pi^+$ [7].

3 Expressiveness of Different Calculi

3.1 Expressiveness

As previously seen, there are multiple variants of pi-calculus, some are typed, and some are not. These calculi that implement session types can be used to show deadlock freedom and correctness as can be seen from this paper [8], where it was used to design crash-preventing mechanisms. However, the inclusion of session types is also restrictive in terms of expressiveness. One could think about this as a trade-off, where stricter rules mean we can prove more/stronger properties, but we can express fewer problems altogether.

The standard procedure to establish that two calculi have the same expressiveness is to provide an encoding (translation) from one calculus to the other and vice versa. To prove separation result (i.e., to prove that one calculus is strictly more expressive than the other) we have to show that one can describe certain behaviours while the other cannot. Palamidessi showed the differnce in expressiveness between the synchronous full pi-calculus and the asynchronous version using a symmetrical version of the leader election problem [9].

Two other patterns are also established as feasible separators, namely the M-configuration and the star pattern ([10, 11]). As our focus is on the leader election problem, and there have not been successful attempts in typed settings, we cannot use an encoding to a calculus that is proved to be expressive enough, we have to construct the system ourselves.

All of these patterns involve non-determinism in the form of choice. As we have seen, in the full (untyped) pi-calculus non-deterministic choice is exclusive once made: it commits to one alternative and discards the rest. Such a unconstrained/careless discarding of resources does not go well with the resource-conscious view of linear logic and the session types originated in it. Henceforth several typed calculi have confluent non-determinism. The confluence lemma states that a step reducing an output and an alternative step reducing an input cannot be conflict to each other and thus can be executed in any order. In the full pi-calculus this confluence lemma is not valid, because inputs and outputs can be combined within a single choice construct and can thus be in conflict [10].

Traditionally session types include choice, where the server offers options, and each one of the clients chooses one. The concept of mixed sessions was introduced, which further expanded the approachable problems with typed pi-calculus [12]. With mixed choice, the line between server and clients became thinner. The only constraint on these sessions is that the choice determines the active communication channel. As it was showed in recent papers [10], the leader election problem can be expressed in the full pi-calculus, but not in these typed calculi variants. This came as a surprising result, as the intuition was that the new options on choice would be expressive enough.

Another direction to capture the full pi-calculus in a type system is to introduce different

types of messages that can be sent through a channel. By introducing *none* and *some* as messages, researchers showed effective ways to correctly express error in parallel programs [13]. Using the new messages, they provided a method to express the familiar 'try... catch...' structure using the pi-calculus.

Relying on the previous results, efforts in other directions are shown by Van den Heuvel, Paulus, Nantes-Sobrinho, and Pérez [6]. They introduced another operation, with its own reduction and typing rules, which allows for a "lazy" commitment and a choice between communication on different channels. This operation invalidates the confluence lemma in the newly proposed calculus but preserves the capability to prove deadlock freedom and type preservation.

We suspect that the main limiting factor (in terms of expressiveness w.r.t. non-determinism) is the existence of the confluence lemma in a calculus; therefore we will mainly focus and base our research on the work in [6].

3.2 Leader election problem

As our research focuses on the leader election problem, it is worth expanding on. A network $P := (\boldsymbol{\nu}\tilde{x})(P_1 \mid \cdots \mid P_k$ in the full (untyped) pi-calculus is an elective network of size k iff for every maximal execution it unguards some output on n (where n is the id of the leader), and exactly one leader is announced. There are multiple types of electoral systems, but a symmetric version with five processes has been used to differentiate between exressiveness ([9, 10, 11]).

This system looks like this in the full pi-calculus [9]:

$$
\begin{aligned}
\mathsf{S} &:= (\boldsymbol{\nu}x, y, z, w, u, a, b, c, d, e)(\mathsf{P_1} \mid \mathsf{P_2} \mid \mathsf{P_3} \mid \mathsf{P_4} \mid \mathsf{P_5}) \\
\mathsf{P_1} &:= \overline{e}[] + a(); (\overline{x}[] + u(); \overline{1}[]) \\
\mathsf{P_2} &:= \overline{a}[] + b(); (\overline{y}[] + w(); \overline{2}[]) \\
\mathsf{P_3} &:= \overline{b}[] + c(); (\overline{z}[] + x(); \overline{3}[]) \\
\mathsf{P_4} &:= \overline{c}[] + d(); (\overline{u}[] + y(); \overline{4}[]) \\
\mathsf{P_5} &:= \overline{d}[] + e(); (\overline{w}[] + z(); \overline{5}[])
\end{aligned}
$$

The election happens in two steps. In the first round two processes go through to the second round, two processes terminate, and one deadlocks. In the second phase one from the two processes terminates, and the other one announces itself as the leader. Any process that sends a message is out of the race (as it takes no continuation). Let us see a possible reduction that elects P_1 as the leader. First P_1 and P_4 have to go through from the first

stage, so reductions happen on a and d:

$$\begin{aligned}
\mathsf{P}_1 &:= (\overline{x}[] + u(); \overline{1}[]) \\
\mathsf{P}_2 &:= \mathbf{0} \\
\mathsf{P}_3 &:= \overline{b}[] + c(); (\overline{z}[] + x(); \overline{3}[]) \\
\mathsf{P}_4 &:= (\overline{u}[] + y(); \overline{4}[]) \\
\mathsf{P}_5 &:= \mathbf{0}
\end{aligned}$$

Now there is only one possible reduction (on channel u), after which we get the following:

$$\begin{aligned}
\mathsf{P}_1 &:= \overline{1}[] \\
\mathsf{P}_2 &:= \mathbf{0} \\
\mathsf{P}_3 &:= \overline{b}[] + c(); (\overline{z}[] + x(); \overline{3}[]) \\
\mathsf{P}_4 &:= \mathbf{0} \\
\mathsf{P}_5 &:= \mathbf{0}
\end{aligned}$$

As there are no more possible reductions, and only one observable (1) signals, the leader P_1 is elected. As the processes are symmetrical, each process can be elected in a similar fashion [9].

4 Results and Discussion

In this section we describe five of our attempts to achieve a well-typed system in $\mathsf{s}\pi^+$ that describes a symmetric elective network of size five. Each attempt has a different focus, and builds on the main take-aways from the previous attempt.

- The first attempt (section 4.1) describes a naive translation from the full pi-calculus into the $\mathsf{s}\pi^+$. This translation fails, because does not respect the pre-congruence introduced in Section 2.4.
- The second attempt (section 4.2) tries to achieve a system that behaves as an elective network of size five, but falls short as there are difficulties when it comes to signalling errors. We fix these problems by introducing new atomic messages together with their reduction rules.
- The third attempt (section 4.3) focuses on achieving a well-typed system from the previous attempt. Our main limiting factor is during the second step, as the propagation of failure is restrictive in its continuation.
- In the fourth attempt (section 4.4) we take a step back, and try to build up the system starting with the second phase in the leader election. We achieve a set of processes that sometimes can elect a leader; however, the cyclical nature of the system makes their parallel composition not well-typed.
- In the fifth and final attempt (section 4.5) we want to achieve a system that is well-typed (bar the cyclical composition), and always acts an elective network. We discover that the typing rules for propagating error are too restrictive together with the current rule for cut. A new rule for cut is proposed, but it cannot guarantee deadlock-freedom.

4.1 First Attempt

As described in the proposal section of the introduction, we started with a naive implementation (see section 3.2) of the leader election problem into $\mathsf{s}\pi^+$. Of course we need to change the operators to match the ones in $\mathsf{s}\pi^+$. We get these processes:

$$
\begin{aligned}
\mathsf{S} &:= (\boldsymbol{\nu} x, y, z, w, u, a, b, c, d, e)(P_1 \mid P_2 \mid P_3 \mid P_4 \mid P_5) \\
\mathsf{P}_1 &:= \overline{e}[] \mathbin{+\!\!+} a(); (\overline{x}[] \mathbin{+\!\!+} u(); \overline{1}[]) \\
\mathsf{P}_2 &:= \overline{a}[] \mathbin{+\!\!+} b(); (\overline{y}[] \mathbin{+\!\!+} w(); \overline{2}[]) \\
\mathsf{P}_3 &:= \overline{b}[] \mathbin{+\!\!+} c(); (\overline{z}[] \mathbin{+\!\!+} x(); \overline{3}[]) \\
\mathsf{P}_4 &:= \overline{c}[] \mathbin{+\!\!+} d(); (\overline{u}[] \mathbin{+\!\!+} y(); \overline{4}[]) \\
\mathsf{P}_5 &:= \overline{d}[] \mathbin{+\!\!+} e(); (\overline{w}[] \mathbin{+\!\!+} z(); \overline{5}[])
\end{aligned}
$$

For the sake of readability we introduce the function $\varphi(P)$ that takes a process as and renames channels. $\varphi = [a \to b, b \to c, c \to d, d \to e, e \to a, x \to y, y \to z, z \to w, w \to u, u \to$

$$[\textsc{Tval.some}]\ \frac{\alpha = \beta, \alpha, \beta \in I}{\overline{x}.\mathtt{some}(\alpha); P \mid x.\mathtt{some}_{\overline{w}}(\beta); Q \rightsquigarrow P \mid Q}$$

$$[\textsc{Tval.none}]\ \frac{\alpha = \beta, \alpha, \beta \in I}{\overline{x}.\mathtt{none}(\alpha) \mid x.\mathtt{some}_{\overline{w}}(\beta); Q \rightsquigarrow \overline{\overline{w}}.\mathtt{none}(\beta)}$$

Figure 4: Reduction rules for "valued" some and none

$x, 1 \to 2, 2 \to 3, 3 \to 4, 4 \to 5, 5 \to 1]$ Therefore $\varphi(P_1) = P_2, \varphi(P_2) = P_3 \ldots \varphi(P_5) = P_1$.

Our focus at first is that the described system is reducible (i.e. communication can occur) and describes an electoral system. We have to check the rule for pre-congruence. Definition 3, rule $1.ii$ in [6] states that the name on which we want to preform reduction has to be free on both sides of the non-deterministic composition. We have to change our system, as currently this property does not hold. This attempt fails, as $\mathsf{s}\pi^+$ has reduction rules different from the full pi-calculus.

4.2 Second Attempt

Now that we know the first problem we need to fix to achieve an electoral system (the domain of names should be equal on both sides of $\nmid$), we can modify our processes accordingly. We also can notice how we will 'discard continuations' and so it comes naturally to use the tools provided for signalling availability. Our prospect system will look like this:

$\mathsf{S} := (\boldsymbol{\nu} x, y, z, w, u, a, b, c, d, e)(P_1 \mid P_2 \mid P_3 \mid P_4 \mid P_5)$
$\mathsf{P_1} := (\boldsymbol{\nu} i)\big((\overline{e}[]; \overline{i}.\mathrm{left}; \overline{a}.\mathtt{none}) \nmid (a(); \overline{i}.\mathrm{right}; \overline{e}.\mathtt{some})\big) \mid i.\mathtt{case}\{\mathrm{left} : a.\mathtt{some}; \mathsf{Q_1}; \mathrm{right} : e.\mathtt{some}; \mathsf{Q_1}\}$
$\mathsf{P_2} = \varphi(\mathsf{P_1}), \mathsf{P_3} = \varphi(\mathsf{P_2}), \mathsf{P_4} = \varphi(\mathsf{P_3}), \mathsf{P_5} = \varphi(\mathsf{P_4})$
$\mathsf{Q_1} := (\boldsymbol{\nu} i)\big((\overline{x}[]; \overline{i}.\mathrm{left}; \overline{z}.\mathtt{none}) \nmid (z(); \overline{i}.\mathrm{right}; \overline{x}.\mathtt{some})\big) \mid i.\mathtt{case}\{\mathrm{left} : z.\mathtt{some}; \overline{1}[]; \mathrm{right} : x.\mathtt{some}; \overline{1}[]\}$
$\mathsf{Q_2} = \varphi(\mathsf{Q_1}), \mathsf{Q_3} = \varphi(\mathsf{Q_2}), \mathsf{Q_4} = \varphi(\mathsf{Q_3}), \mathsf{Q_5} = \varphi(\mathsf{Q_4})$

Here we wanted to achieve a reducible system. The processes are still symmetric, but we added the second step differently. It would happen in the processes labelled $\mathsf{Q}_{\ldots}$, with the same structure. The idea is that every process that synchronizes with its left side sends an unavailable signal, and therefore discards its corresponding second step in the 'internal' channel named i (note that when parallel composed, we can alpha-rename these internal channels so that there is no naming conflict). But first let us check, does everything work as planned?

Unfortunately not. In a scenario, where the left side of $\mathsf{P_1}$ and the right side of $\mathsf{P_5}$ synchronize on channel e, and the left side of $\mathsf{P_3}$ and the right side of $\mathsf{P_2}$ synchronize on channel b the following could happen. We should keep $\mathsf{Q_5}$ and $\mathsf{Q_2}$, while discarding $\mathsf{Q_1}$ and $\mathsf{Q_3}$. But the $\overline{a}.\mathtt{some}$ (from $\mathsf{P_2}$) could interact with either $a.\mathtt{some}$ - the one from $\mathsf{P_1}$ or the one from $\mathsf{P_2}$. This could cause a deadlock in the second step, as $\mathsf{Q_1}$ and $\mathsf{Q_5}$ do not share any names.

But notice how the only time when we do not send names or values through a channel is when we signal availability. What if we had an atomic interaction, a valued some and none, which could only interact with waitings that have the same value? This would provide us with a finer control over parallelism. Figure 4 describes the newly proposed reduction rules. Now we can modify the processes with this new operator:

$$\begin{aligned}
&\mathsf{S} := (\boldsymbol{\nu} x, y, z, w, u, a, b, c, d, e)(P_1 \mid P_2 \mid P_3 \mid P_4 \mid P5)\\
&\mathsf{P}_1 := (\boldsymbol{\nu} i)\big((\overline{e}[]; \overline{i}.\mathrm{left}; \overline{a}.\mathtt{none}(\alpha) \mathbin{\Vert} (a(); \overline{i}.\mathrm{right}; \overline{e}.\mathtt{some}(\beta))\big)\mid\\
&i.\mathtt{case}\{\mathrm{left} : a.\mathtt{some}(\alpha); \mathsf{Q}_1; \mathrm{right} : e.\mathtt{some}(\beta); \mathsf{Q}_1\}\\
&\mathsf{P}_2 = \varphi(\mathsf{P}_1), \mathsf{P}_3 = \varphi(\mathsf{P}_2), \mathsf{P}_4 = \varphi(\mathsf{P}_3), \mathsf{P}_5 = \varphi(\mathsf{P}_4)
\end{aligned}$$

Our initial second attempt failed to achieve a (not necessarily well-typed) system that describes the leader election problem in $\mathsf{s}\pi^+$. We fixed the minor shortcomings with newly proposed syntax and reduction rules for *some* and *none*.

4.3 Third Attempt

Now that we have a program that describes an elective network of size five we will turn our attention to whether it is well-typed.

The first conflict is with the non-deterministic composition. The typing rule T$\mathbin{\Vert}$ states that the two sides need to be typeable under the same context. In other words they can only differ in choices (and the subsequent actions on that channel), the order of the operations, and the available/unavailable signals on a channel.

Secondly, the parallel composition with the "internal" channels is problematic. TCUT tells us that we are only allowed to parallel compose two process if they share exactly one name, and the type of that name in one process is the dual of the type of the same name in the other. In our previous attempt, there where three names shared, which clearly violates this typing rule.

If we now modify our system to adhere to these rules we will get the following:

$$\begin{aligned}
&\mathsf{S} := (\boldsymbol{\nu} x, y, z, w, u, a, b, c, d, e)(P_1 \mid P_2 \mid P_3 \mid P_4 \mid P_5)\\
&\mathsf{P}_1 := (\boldsymbol{\nu} i)\big((\overline{e}.\mathtt{some}(\alpha); \overline{e}.right; a.\mathtt{some}(\beta); a.\mathtt{case}\begin{Bmatrix}\mathsf{left} : \overline{i}[channel_1]; \overline{i}.left; \overline{\overline{channel_1}.\mathtt{none}(\alpha)}\\ \mathsf{right} : \overline{i}[channel_1]; \overline{i}.right; \overline{\overline{channel_1}.\mathtt{some}(\alpha)}\end{Bmatrix}) \mathbin{\Vert}\\
&(a.\mathtt{some}(\alpha); a.\mathtt{case}\begin{Bmatrix}\mathsf{left} : \overline{i}[channel_1]; \overline{i}.left; \overline{e}.\mathtt{some}(\beta); \overline{e}.left; \overline{\overline{channel_1}.\mathtt{none}(\beta)}\\ \mathsf{right} : \overline{i}[channel_1]; \overline{i}.right; \overline{\overline{channel_1}.\mathtt{some}(\beta)}; \overline{e}.\mathtt{some}(\beta); \overline{e}.left\end{Bmatrix})\big)\mid\\
&(i(channel); i.\mathtt{case}\begin{Bmatrix}\mathsf{left} : channel.\mathtt{some}(\alpha); \mathsf{Q}_1\\ \mathsf{right} : channel.\mathtt{some}(\beta); \mathsf{Q}_1\end{Bmatrix})\\
&\mathsf{P}_2 = \varphi(\mathsf{P}_1), \mathsf{P}_3 = \varphi(\mathsf{P}_2), \mathsf{P}_4 = \varphi(\mathsf{P}_3), \mathsf{P}_5 = \varphi(\mathsf{P}_4)
\end{aligned}$$

However, there is still one problem with the typing of each process, and it has to do with how signalling error happens. The typing rule T$\oplus$some describes that every discardable resource has to start with announcing its (un)availability. So every name in Q_1 has to start with

$\overline{\cdots}$.some or $\overline{\cdots}$.none. This would make it impossible to for the processes in the second step to communicate with each other.

We did not achieve a set of well-typed processes, that when parallel composed describe the leader election problem. The main obstacle was in propagating the error, as typing rules restrict us in what messages can be sent first on a channel in processes $Q_1, \ldots, Q_5$.

4.4 Fourth Attempt

To solve this, we have to take a step back. We decided to discard the previous efforts (temporarily at least) and start afresh. This also means we tried to use the readily available tools, and did not consider the valued *some* and *none* further. Our approach here was to start with the second step and then to try and build the first step around it.

Our candidate processes looked like this:

$$\mathsf{P}_1 := \left(\overline{x}.right; \overline{x}.\mathsf{some}; \overline{x}[]; y.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\\ \mathsf{right} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\end{array}\right\}\right) \nparallel$$
$$\left(y.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : \overline{x}.left; y.\mathsf{some}_{x,1}; \overline{1}.\mathsf{some}; y(); \overline{x}.\mathsf{none}\\ \mathsf{right} : \overline{x}.left; y.\mathsf{some}_{x,1}; \overline{1}.\mathsf{some}; y(); \overline{x}.\mathsf{none}\end{array}\right\}\right)$$
$$\mathsf{P}_2 = \varphi(\mathsf{P}_1), \mathsf{P}_3 = \varphi(\mathsf{P}_2), \mathsf{P}_4 = \varphi(\mathsf{P}_3), \mathsf{P}_5 = \varphi(\mathsf{P}_4)$$

P_1 is typeable with $\mathsf{P}_1 \vdash x : \oplus\{l : \&1; r : \&1\}, y : \&\{l : \oplus\bot, r : \oplus\bot\}$. We do not type the observable 1. Notice how x is the dual of y. Because of the symmetry in our system, now every name appears twice, where one instance is the dual of the other. Another interesting thing is that if we parallel compose these five processes, they can behave as an electoral system.

$$\mathsf{P}_1 := \left(\overline{x}.right; \overline{x}.\mathsf{some}; \overline{x}[]; y.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\\ \mathsf{right} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\end{array}\right\}\right) \nparallel$$
$$\left(y.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : \overline{x}.left; y.\mathsf{some}_{x,1}; \overline{1}.\mathsf{some}; y(); \overline{x}.\mathsf{none}\\ \mathsf{right} : \overline{x}.left; y.\mathsf{some}_{x,1}; \overline{1}.\mathsf{some}; y(); \overline{x}.\mathsf{none}\end{array}\right\}\right)$$
$$\mathsf{P}_2 := \left(\overline{y}.right; \overline{y}.\mathsf{some}; \overline{y}[]; z.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none}\\ \mathsf{right} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none}\end{array}\right\}\right) \nparallel$$
$$\left(z.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none}\\ \mathsf{right} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none}\end{array}\right\}\right)$$
$$\mathsf{P}_3 := \left(\overline{z}.right; \overline{z}.\mathsf{some}; \overline{z}[]; w.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : w.\mathsf{some}_3; w(); \overline{3}.\mathsf{none}\\ \mathsf{right} : w.\mathsf{some}_3; w(); \overline{3}.\mathsf{none}\end{array}\right\}\right) \nparallel$$
$$\left(w.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : \overline{z}.left; w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none}\\ \mathsf{right} : \overline{z}.left; w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none}\end{array}\right\}\right)$$
$$\mathsf{P}_4 := \left(\overline{w}.right; \overline{w}.\mathsf{some}; \overline{w}[]; u.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none}\\ \mathsf{right} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none}\end{array}\right\}\right) \nparallel$$
$$\left(u.\mathsf{case}\left\{\begin{array}{l}\mathsf{left} : \overline{w}.left; u.\mathsf{some}_{w,4}; \overline{4}.\mathsf{some}; u(); \overline{w}.\mathsf{none}\\ \mathsf{right} : \overline{w}.left; u.\mathsf{some}_{w,4}; \overline{4}.\mathsf{some}; u(); \overline{w}.\mathsf{none}\end{array}\right\}\right)$$

$$\mathsf{P}_5 := \big(\overline{u}.right; \overline{u}.\mathsf{some}; \overline{u}[]; x.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : x.\mathsf{some}_5; x(); \overline{5}.\mathsf{none} \\ \mathsf{right} : x.\mathsf{some}_5; x(); \overline{5}.\mathsf{none}\end{matrix}\right\}\big) \nparallel$$
$$\big(x.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : \overline{u}.left; x.\mathsf{some}_{u,5}; \overline{5}.\mathsf{some}; x(); \overline{u}.\mathsf{none} \\ \mathsf{right} : \overline{u}.left; x.\mathsf{some}_{u,5}; \overline{5}.\mathsf{some}; x(); \overline{u}.\mathsf{none}\end{matrix}\right\}\big)$$

If we want to "elect" P_5 as the leader, we first have to choose to make a reduction on x.

$$\mathsf{P}_1 := \overline{x}.\mathsf{some}; \overline{x}[]; y.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none} \\ \mathsf{right} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\end{matrix}\right\}$$
$$\mathsf{P}_2 := \big(\overline{y}.right; \overline{y}.\mathsf{some}; \overline{y}[]; z.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none} \\ \mathsf{right} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none}\end{matrix}\right\}\big) \nparallel$$
$$\big(z.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none} \\ \mathsf{right} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none}\end{matrix}\right\}\big)$$
$$\mathsf{P}_3 := \big(\overline{z}.right; \overline{z}.\mathsf{some}; \overline{z}[]; w.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : w.\mathsf{some}_3; w(); \overline{3}.\mathsf{none} \\ \mathsf{right} : w.\mathsf{some}_3; w(); \overline{3}.\mathsf{none}\end{matrix}\right\}\big) \nparallel$$
$$\big(w.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : \overline{z}.left; w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none} \\ \mathsf{right} : \overline{z}.left; w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none}\end{matrix}\right\}\big)$$
$$\mathsf{P}_4 := \big(\overline{w}.right; \overline{w}.\mathsf{some}; \overline{w}[]; u.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none} \\ \mathsf{right} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none}\end{matrix}\right\}\big) \nparallel$$
$$\big(u.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : \overline{w}.left; u.\mathsf{some}_{w,4}; \overline{4}.\mathsf{some}; u(); \overline{w}.\mathsf{none} \\ \mathsf{right} : \overline{w}.left; u.\mathsf{some}_{w,4}; \overline{4}.\mathsf{some}; u(); \overline{w}.\mathsf{none}\end{matrix}\right\}\big)$$
$$\mathsf{P}_5 := \overline{u}.left; x.\mathsf{some}_{u,5}; \overline{5}.\mathsf{some}; x(); \overline{u}.\mathsf{none}$$

Now there are three possible reductions: the processes can communicate on $u, w,$ or z. If the reduction happens on w, the processes will signal availability on 5 and 3, if it happens on z, 5 and 2 will be available, if the reduction happens on u and then w only 5 will be available. So let us see the different options.

$$\mathsf{P}_1 := \overline{x}.\mathsf{some}; \overline{x}[]; y.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none} \\ \mathsf{right} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\end{matrix}\right\}$$
$$\mathsf{P}_2 := \big(\overline{y}.right; \overline{y}.\mathsf{some}; \overline{y}[]; z.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none} \\ \mathsf{right} : z.\mathsf{some}_2; z(); \overline{2}.\mathsf{none}\end{matrix}\right\}\big) \nparallel$$
$$\big(z.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none} \\ \mathsf{right} : \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none}\end{matrix}\right\}\big)$$
$$\mathsf{P}_3 := \overline{z}.left; w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none}$$
$$\mathsf{P}_4 := \overline{w}.\mathsf{some}; \overline{w}[]; u.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none} \\ \mathsf{right} : u.\mathsf{some}_4; u(); \overline{4}.\mathsf{none}\end{matrix}\right\}$$
$$\mathsf{P}_5 := \overline{u}.left; x.\mathsf{some}_{u,5}; \overline{5}.\mathsf{some}; x(); \overline{u}.\mathsf{none}$$

And then the only possible reduction:

$$\mathsf{P}_1 := \overline{x}.\mathsf{some}; \overline{x}[]; y.\mathsf{case}\left\{\begin{matrix}\mathsf{left} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none} \\ \mathsf{right} : y.\mathsf{some}_1; y(); \overline{1}.\mathsf{none}\end{matrix}\right\}$$
$$\mathsf{P}_2 := \overline{y}.left; z.\mathsf{some}_{y,2}; \overline{2}.\mathsf{some}; z(); \overline{y}.\mathsf{none}$$
$$\mathsf{P}_3 := w.\mathsf{some}_{z,3}; \overline{3}.\mathsf{some}; w(); \overline{z}.\mathsf{none}$$

$$\mathsf{P}_4 := \overline{w}.\mathtt{some}; \overline{w}[]; u.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : u.\mathtt{some}_4; u(); \overline{4}.\mathtt{none}\\ \mathsf{right} : u.\mathtt{some}_4; u(); \overline{4}.\mathtt{none}\end{array}\right\}$$
$$\mathsf{P}_5 := \overline{u}.left; x.\mathtt{some}_{u,5}; \overline{5}.\mathtt{some}; x(); \overline{u}.\mathtt{none}$$

This will ultimately result in sending none on channels 1, 2, and 4; and some on channels 5, and 3. The reductions to make channels 5, and 2 available are similar. But if we first reduce on u these are the processes:

$$\mathsf{P}_1 := \overline{x}.\mathtt{some}; \overline{x}[]; y.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : y.\mathtt{some}_1; y(); \overline{1}.\mathtt{none}\\ \mathsf{right} : y.\mathtt{some}_1; y(); \overline{1}.\mathtt{none}\end{array}\right\}$$
$$\mathsf{P}_2 := \left(\overline{y}.right; \overline{y}.\mathtt{some}; \overline{y}[]; z.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : z.\mathtt{some}_2; z(); \overline{2}.\mathtt{none}\\ \mathsf{right} : z.\mathtt{some}_2; z(); \overline{2}.\mathtt{none}\end{array}\right\}\right) \parallel$$
$$\left(z.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : \overline{y}.left; z.\mathtt{some}_{y,2}; \overline{2}.\mathtt{some}; z(); \overline{y}.\mathtt{none}\\ \mathsf{right} : \overline{y}.left; z.\mathtt{some}_{y,2}; \overline{2}.\mathtt{some}; z(); \overline{y}.\mathtt{none}\end{array}\right\}\right)$$
$$\mathsf{P}_3 := \left(\overline{z}.right; \overline{z}.\mathtt{some}; \overline{z}[]; w.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : w.\mathtt{some}_3; w(); \overline{3}.\mathtt{none}\\ \mathsf{right} : w.\mathtt{some}_3; w(); \overline{3}.\mathtt{none}\end{array}\right\}\right) \parallel$$
$$\left(w.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : \overline{z}.left; w.\mathtt{some}_{z,3}; \overline{3}.\mathtt{some}; w(); \overline{z}.\mathtt{none}\\ \mathsf{right} : \overline{z}.left; w.\mathtt{some}_{z,3}; \overline{3}.\mathtt{some}; w(); \overline{z}.\mathtt{none}\end{array}\right\}\right)$$
$$\mathsf{P}_4 := \overline{w}.left; u.\mathtt{some}_{w,4}; \overline{4}.\mathtt{some}; u(); \overline{w}.\mathtt{none}$$
$$\mathsf{P}_5 := x.\mathtt{some}_{u,5}; \overline{5}.\mathtt{some}; x(); \overline{u}.\mathtt{none}$$

Following this with a reduction on w:

$$\mathsf{P}_1 := \overline{x}.\mathtt{some}; \overline{x}[]; y.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : y.\mathtt{some}_1; y(); \overline{1}.\mathtt{none}\\ \mathsf{right} : y.\mathtt{some}_1; y(); \overline{1}.\mathtt{none}\end{array}\right\}$$
$$\mathsf{P}_2 := \left(\overline{y}.right; \overline{y}.\mathtt{some}; \overline{y}[]; z.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : z.\mathtt{some}_2; z(); \overline{2}.\mathtt{none}\\ \mathsf{right} : z.\mathtt{some}_2; z(); \overline{2}.\mathtt{none}\end{array}\right\}\right) \parallel$$
$$\left(z.\mathtt{case}\left\{\begin{array}{l}\mathsf{left} : \overline{y}.left; z.\mathtt{some}_{y,2}; \overline{2}.\mathtt{some}; z(); \overline{y}.\mathtt{none}\\ \mathsf{right} : \overline{y}.left; z.\mathtt{some}_{y,2}; \overline{2}.\mathtt{some}; z(); \overline{y}.\mathtt{none}\end{array}\right\}\right)$$
$$\mathsf{P}_3 := \overline{z}.left; w.\mathtt{some}_{z,3}; \overline{3}.\mathtt{some}; w(); \overline{z}.\mathtt{none}$$
$$\mathsf{P}_4 := u.\mathtt{some}_{w,4}; \overline{4}.\mathtt{some}; u(); \overline{w}.\mathtt{none}$$
$$\mathsf{P}_5 := x.\mathtt{some}_{u,5}; \overline{5}.\mathtt{some}; x(); \overline{u}.\mathtt{none}$$

These processes will ultimately collapse into sending *some* on 5, and *none* on channels 1-4.

As we have seen, the processes are typable when considered in isolation. Now let us examine whether the parallel composition of these processes is well-typed. Cutting on y, z, w, and composing the first four processes is fine, as the already composed processes and the newly added one share only one name, i.e., $((\boldsymbol{\nu} w)((\boldsymbol{\nu} z)((\boldsymbol{\nu} y)(P_1 \mid P_2)) \mid P_3) \mid P_4)$. Sadly, when we want to add P_5 we discover that we would need to cut on two names (x and u), which is not allowed by typing. This does not come as a big surprise: the leader election problem is inherently circular, and $\mathsf{s}\pi^+$ disallows circular dependencies to avoid deadlocks.

In this attempt we discovered two big obstacles: the not well-typed nature of circular networks in $\mathsf{s}\pi^+$, and the inherent non-determinism of the order of interactions in parallel sys-

tems. The first problem is not unsolvable. As shown in other papers [14], the introduction of priorities in the typing system can facilitate the typing of cyclic networks. The second problem originates in the operator $\parallel$, as the two sides essentially have to describe the same session, just with different implementations. A system is an electoral system if it always behaves as one, therefore we still need to add the second step (as seen in the solution in the full pi-calculus).

4.5 Fifth Attempt

Our latest attempt took us close to a typeable system, which always elects a leader in one round. Because of the previously mentioned inherent non-determinism in parallel processes (we do not know when a communication will occur) we have to include the second step. We will try to use the same form, that we have in our existing system, but we are bounded by the typing rule for $x.\mathtt{some}_{\overline{\omega}}$. If we change $\mathsf{P}_1 \dots \mathsf{P}_5$ into

$$\mathsf{P}_1 := \big(\overline{x}.right; \overline{x}.\mathtt{some}; \overline{x}[]; y.\mathtt{case}\begin{Bmatrix}\mathsf{left} : y.\mathtt{some}_a; y(); \overline{a}.\mathtt{none} \\ \mathsf{right} : y.\mathtt{some}_a; y(); \overline{a}.\mathtt{none}\end{Bmatrix}\big)$$
$$\parallel \big(y.\mathtt{case}\begin{Bmatrix}\mathsf{left} : \overline{x}.left; y.\mathtt{some}_{x,a}; \overline{a}.\mathtt{some}; y(); \overline{x}.\mathtt{none} \\ \mathsf{right} : \overline{x}.left; y.\mathtt{some}_{x,a}; \overline{a}.\mathtt{some}; y(); \overline{x}.\mathtt{none}\end{Bmatrix}\big)$$

and parallel compose it with the following $\mathsf{Q}_1 \dots \mathsf{Q}_5$, where:

$$\mathsf{Q}_1 := a.\mathtt{some}_{c,1}\big((\overline{c}.\mathtt{some}; \overline{c}[]; a.\mathtt{some}_1; a(); \overline{1}.\mathtt{none}) \parallel (a.\mathtt{some}_{c,1}; a(); \overline{1}.\mathtt{some}; \overline{c}.\mathtt{none})\big)$$
$$\mathsf{Q}_2 = \varphi(\mathsf{Q}_1), \mathsf{Q}_3 = \varphi(\mathsf{Q}_2), \mathsf{Q}_4 = \varphi(\mathsf{Q}_3), \mathsf{Q}_5 = \varphi(\mathsf{Q}_4)$$

then we get a system that elects a leader in two steps.
However because Q_1 starts $a.\mathtt{some}_{\dots}; Q_1'$ every other channel that is in the domain of Q_1' needs to be typeable under the & monad. If both sides start with $\overline{\alpha}.\mathtt{some}$, then there will not be possible reductions in the second step. This means that one side of the non-deterministic composition needs to start with $\beta.\mathtt{some}_{\dots}$, in that case it must be true that: $\beta = a$, as a is the only channel that is free to continue with anything else than an (un)available signal in the domain of Q_1'

Notice how this leads to problems with cut, as now Q_1 does not have the dual of a, so we cannot parallel compose and cut P_1 and Q_1 with respect to a. One possible solution would be the introduction of a new rule, that allows for partial cut, with some additional constraints.

$$[\textsc{Tcut.par}]\ \frac{P \vdash \Gamma, x{:}A_1.A_2 \qquad Q \vdash \Delta, x{:}\overline{A_1} \qquad R \vdash \Theta, x{:}\overline{A_2}}{(\boldsymbol{\nu} x)P \mid Q \mid R \vdash \Gamma, \Delta, \Theta}$$

Figure 5: Partial typing

This rule allows us to parallel compose and cut w.r.t. a the following: $(\boldsymbol{\nu} a)(P_1 \mid Q_1 \mid Q_4)$, as P_1 has the dual of the first $a.\mathtt{some}_{\dots}$ in Q_1 and Q_4 carries the rest.

We would still have a problem with circular compositions, although not unsolvable. But this newly introduced rule would be a rather drastic change, and would violate the by-construction guaranteed communication safety and deadlock-freedom: $(x.\texttt{some}; x.\texttt{some}; x()) \mid (\overline{x}.\texttt{some}; \overline{x}[]) \mid (\overline{x}.\texttt{none})$ would be well typed, but could result in a deadlock (as it can "consume" some two times, one some and a none, or one none, and the order is not specified).

5 Conclusion

As seen in the previous section, we obtained negative results. We were quite surprised by this, as our initial assumption (that confluent non-determinism is the limiting factor) looks wrong. In this section we will summarize our understanding of the limiting factors, as well as our perception about how great of an obstacle they are.

Circular composition

As we have seen in section 4.4 $\mathsf{s}\pi^+$ does not allow circular compositions in a well typed system. Because the leader election problem is inherently symmetric (with respect to the participating processes), it will introduce circular composition. However, this challenge is solvable by introducing so-called priorities. As we can see in [14] this is an effective tool to provide the possibility for typing circular connections. This is extendable to $\mathsf{s}\pi^+$ but in itself would not be enough and could not guarantee deadlock-freedom.

Propagation of errors

Another great challenge is the propagation of errors. As $\mathsf{s}\pi^+$ has non-confluent non-determinism, the tension between linearity and discarding resources is strong. Signalling that we have thrown off some resources provides a good solution. It also introduces a new form of non-determinism (as we could see in section 4.3), where we would want to make sure that certain parts interact whereas others do not, but we are bounded by the reduction rules of the non-deterministic operator. We believe this is a problem that would require a different framework.

Non-determinism in the implementation of sessions

Finally, the last limiting factor can be found in the non-deterministic composition as seen in section 4.5. The two composed processes must implement the same session, the only difference allowed is in the order of communication, the choice in labels (and therefore subsequent actions on that channel), and the available/unavailable signals. We do not have full mixed choice available: we have to make the same kind of choices, but in different order.

To summarize our findings and answer our research question: we were unable to derive a typeable system in $\mathsf{s}\pi^+$ that describes the leader election problem for five processes. The results are inconclusive (in this case counterexamples are not good enough) as we lack rigid proof. Our research, however, provided us with a better understanding of what types really are, how they can be a limiting factor, and where may the boundaries of expressiveness lay when we use them.

6 Future Work

During this project we found many possible avenues that we find worth exploring. One possible direction is the exploration of valued some and none. Do they add expressiveness? Is our proposed reduction rule in line with the system of linear logic? If so, what would be the corresponding typing rules? These are all questions that were not closely related to the aim of this thesis.

Another immediate direction for continuation is the proof that $\mathsf{s}\pi^+$ cannot express an elective network of size 5. This project provides a solid foundation for this, and hopefully inspires in the right direction. A related question is under what assumptions do we have a well-typed system for the leader election problem?

Finally, the introduction of circularity in $\mathsf{s}\pi^+$, together with priorities is a lucrative option for research as well. We think that this might be the least challenging to add, and it would improve the expressive power of $\mathsf{s}\pi^+$.

References

[1] Edsger Wybe Dijkstra. Notes on Structured Programming. EWD249 – circulated privately – http://www.cs.utexas.edu/users/EWD/ewd02xx/EWD249.PDF – geprüft: 12. August 2002, April 1970.

[2] Jean-Yves Girard. Linear logic. 50(1):1–101.

[3] William Alvin Howard. The formulae-as-types notion of construction. In Haskell Curry, Hindley B., Seldin J. Roger, and P. Jonathan, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Academic Press, 1980.

[4] Robin Milner. Elements of interaction - turing award lecture. *Commun. ACM*, 36(1):78–89, 1993.

[5] Philip Wadler. Propositions as sessions. In Peter Thiemann and Robby Bruce Findler, editors, *ACM SIGPLAN International Conference on Functional Programming, ICFP'12, Copenhagen, Denmark, September 9-15, 2012*, pages 273–286. ACM, 2012.

[6] Bas van den Heuvel, Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez. Typed non-determinism in functional and concurrent calculi. In Chung-Kil Hur, editor, *Programming Languages and Systems - 21st Asian Symposium, APLAS 2023, Taipei, Taiwan, November 26-29, 2023, Proceedings*, volume 14405 of *Lecture Notes in Computer Science*, pages 112–132. Springer, 2023.

[7] Bas van den Heuvel, Joseph W. N. Paulus, Daniele Nantes-Sobrinho, and Jorge A. Pérez. Typed non-determinism in functional and concurrent calculi, 2023.

[8] Adam D. Barwell, Ping Hou, Nobuko Yoshida, and Fangyi Zhou. Designing asynchronous multiparty protocols with crash-stop failures. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 263, 2023. Type: Conference paper.

[9] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Math. Struct. Comput. Sci.*, 13(5):685–719, 2003.

[10] Kirstin Peters and Nobuko Yoshida. On the expressiveness of mixed choice sessions. In Valentina Castiglioni and Claudio Antares Mezzina, editors, *Proceedings Combined 29th International Workshop on Expressiveness in Concurrency and 19th Workshop on Structural Operational Semantics, EXPRESS/SOS 2022, and 19th Workshop on Structural Operational Semantics Warsaw, Poland, 12th September 2022*, volume 368 of *EPTCS*, pages 113–130, 2022.

[11] Kirstin Peters and Nobuko Yoshida. Separation and encodability in mixed choice multiparty sessions (technical report). *CoRR*, abs/2405.08104, 2024.

[12] Filipe Casal, Andreia Mordido, and Vasco T. Vasconcelos. Mixed sessions. *Theor. Comput. Sci.*, 897:23–48, 2022.

[13] Luís Caires and Jorge A. Pérez. Linearity, control effects, and behavioral types. In Hongseok Yang, editor, *Programming Languages and Systems*, volume 10201, pages 229–259. Springer Berlin Heidelberg, 2017. Series Title: Lecture Notes in Computer Science.

[14] Bas van den Heuvel and Jorge A. Pérez. Asynchronous session-based concurrency: Deadlock-freedom in cyclic process networks, 2024.