

Races in Classical Linear Logic

Wen Kokke



Master of Science by Research
Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2017

Abstract

Process calculi based in logic, such as π DILL and CP, provide a foundation for deadlock-free concurrent programming, but at the cost of excluding non-determinism and races. We introduce CP_{ND} (nodcap), which extends CP with a novel account of non-determinism. Our approach draws on bounded linear logic to provide a strongly-typed account of standard process calculus expressions of non-determinism. We show that our extension is expressive enough to capture many uses of non-determinism in untyped calculi, such as non-deterministic choice, while preserving CP's meta-theoretic properties, including deadlock freedom.

Acknowledgements

I would like to thank Phil Wadler, who taught me that there is no such thing as too many examples, and who encouraged my tendency to fill all of them with emoji.

I would like to thank Garrett Morris, who saved my half-baked theories so often that I've lost count, and for the bakeware—thanks!

I would like to thank Simon Fowler, who showed me around the Forum on the day that I first got to Edinburgh, and who has been wonderfully kind ever since.

And finally, I would like to thank Rudi Horn, Thomas Wright, and all the others who helped me during the writing of this dissertation by giving me invaluable feedback or simply clarifying little things...

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.

—*Wen Kokke*

Contents

1	Introduction	1
2	Background	3
2.1	Classical Processes	3
2.1.1	Terms and types	3
2.1.2	Multiplicatives and in- and interdependence	7
2.1.3	Additives and choice	9
2.1.4	Structural rules and duality	11
2.1.5	Commuting conversions	11
2.1.6	Example	12
2.1.7	Properties of RCP	12
2.2	Non-determinism, logic, and session types	16
3	CP as a type system for the π-calculus	21
3.1	Canonical forms	22
3.2	Evaluation contexts	24
3.3	Progress	25
3.4	Rewriting versus commuting	28
4	Non-deterministic Classical Processes	31
4.1	Terms and types	32
4.2	Typing clients and servers	35
4.2.1	Clients and pooling	36
4.2.2	Servers and contraction	36
4.3	Running clients and servers	37
4.4	Properties of CP_{ND}	39
4.4.1	Preservation	39

4.4.2	Canonical forms and progress	40
4.4.3	Evaluation contexts	41
4.4.4	Progress	44
4.4.5	Termination	47
4.5	CP_{ND} and non-deterministic local choice	47
5	Conclusions and future work	53
5.1	Mechanisation of CP_{ND}	53
5.2	Relation to bounded linear logic	53
5.3	Name restriction and parallel composition	54
5.4	Recursion and resource variables	54
5.5	Cuts with leftovers	55
5.6	Relation to exponentials in CP	55

Chapter 1

Introduction

Consider the following scenario:

John and Mary are working from home one morning when they get a craving for a slice of cake. Being denizens of the web, they quickly find the nearest store which does home deliveries. Unfortunately for them, they both order their cake at the *same* store, which has only one slice left. After that, all it can deliver is disappointment.

This is an example of a *race condition*. We can model this scenario in the π -calculus, assuming 🧑, 🧑 and 🛒 are three processes modeling John, Mary and the store, and 🍰 and 🙌 are two channels giving access to a slice of cake and disappointment, respectively. As expected, this process has two possible outcomes: either John gets the cake, and Mary gets disappointment, or vice versa.

$$\begin{aligned} & (\bar{x}(\text{🍰}).\bar{x}(\text{🙌}).\text{🛒} \mid x(y).\text{🧑} \mid x(z).\text{🧑}) \\ & \quad \Downarrow_{\star} \\ & (\text{🛒} \mid \text{🧑}\{\text{🍰}/y\} \mid \text{🧑}\{\text{🙌}/z\}) \quad \text{or} \quad (\text{🛒} \mid \text{🧑}\{\text{🙌}/y\} \mid \text{🧑}\{\text{🍰}/z\}) \end{aligned}$$

While John or Mary may not like all of the outcomes, it is the store which is responsible for implementing the online delivery service, and the store is happy with either outcome. Thus, the above is an interaction we would like to be able to model.

Now consider another scenario, which takes place *after* John has already bought the cake:

Mary is *really* disappointed when she finds out the cake has sold out. John, always looking to make some money, offers to sell the slice to

her for a profit. Mary agrees to engage in a little bit of back-alley cake resale, but sadly there is no trust between the two. John demands payment first. Mary would rather get her slice of cake before she gives John the money.

This is an example of a *deadlock*. We can also model this scenario in the π -calculus, assuming that £ is a channel giving access to some adequate amount of money.

$$(x\langle z \rangle.\bar{y}\langle \text{🍰} \rangle.\text{👤} \mid y\langle w \rangle.\bar{x}\langle \text{£} \rangle.\text{👩}) \not\Rightarrow^* \star$$

The above process does not reduce. As both John and Mary would prefer the exchange to be made, this interaction is desired by *neither*. Thus, the above is an interaction we would *somehow* like to exclude.

Session types [8] statically guarantee that concurrent programs, such as those above, respect communication protocols. Session-typed calculi with logical foundations, such as π DILL [5] and CP [14], obtain deadlock freedom as a result of a close correspondence with logic. The same correspondence, however, also rules out non-determinism and race conditions.

In this dissertation, we present CP_{ND} (nodcap), an extension of CP [14] with a novel account of non-determinism and races. Inspired by bounded linear logic [7], we introduce a form of shared channels, in which the type of a shared channel tracks how many times it is reused. As in the untyped π -calculus, sharing introduces the potential for non-determinism. We show that our approach is sufficient to capture practical examples of races, such as the web store, as well as other formal characterizations of non-determinism, such as non-deterministic choice. However, CP_{ND} does not lose the metatheoretical benefits of CP: we show that it enjoys termination and deadlock-freedom.

This dissertation proceeds as follows. In chapter 2, we introduce CP and reproduce the proofs of some of its meta-theoretical results, and discuss related work which also addresses the addition of non-determinism to logic-inspired session typed process calculi. In chapter 3, we introduce an alternative reduction strategy for CP, which more closely resembles reduction in the π -calculus. In chapter 4, we introduce CP_{ND} . Finally, in chapter 5, we conclude with a discussion of the work done in this dissertation and potential avenues for future work.

Chapter 2

Background

2.1 Classical Processes

In this section, we will discuss a rudimentary subset of the typed process calculus CP [9, 14], which we will refer to as RCP. We have chosen to discuss only a subset in order to keep the discussion of CP_{ND} in chapter 4 as simple as possible. RCP is the subset of CP which corresponds to rudimentary linear logic [7, RLL], also known as multiplicative-applicative linear logic. We foresee no problems in extending the proofs from chapter 4 to cover the remaining features of CP, polymorphism and the exponentials $!A$ and $?A$.

This chapter proceeds as follows. In section 2.1.1, we introduce the terms, the structural congruence, and the types of RCP. In sections 2.1.2 to 2.1.5, we discuss the terms and their corresponding types, in small groups, together with their typing and reduction rules. In section 2.1.7, we prove preservation, progress and termination for RCP.

2.1.1 Terms and types

The term language for RCP is a variant of the π -calculus [12]. Its terms are defined by the following grammar:

Definition 2.1 (Terms)

$P, Q, R := x \leftrightarrow y$	link
$ \nu x.(P \mid Q)$	parallel composition, or “cut”
$ x[y].(P \mid Q)$	“output”
$ x(y).P$	“input”
$ x[].0$	halt
$ x().P$	wait
$ x[\text{inl}].P$	select left choice
$ x[\text{inr}].P$	select right choice
$ \text{case } x \{P; Q\}$	offer binary choice
$ \text{case } x \{\}$	offer nullary choice

The variables x, y, z and w range over channel names. The construct $x \leftrightarrow y$ links two channels [2, 13], forwarding messages received on x to y and vice versa. The construct $\nu x.(P \mid Q)$ creates a new channel x , and composes two processes, P and Q , which communicate on x , in parallel. Therefore, in $\nu x.(P \mid Q)$ the name x is bound in both P and Q . In $x(y).P$ and $x[y].(P \mid Q)$, round brackets denote input, square brackets denote output. We use bound output [13], meaning that both input and output bind a new name. In $x(y).P$ the new name y is bound in P . In $x[y].(P \mid Q)$, the new name y is only bound in P , while x is only bound in Q .

Terms in RCP are identified up to structural congruence, which states that parallel compositions $\nu x.(P \mid Q)$ are associative and commutative. It is defined as follows:

Definition 2.2 (Structural congruence)

We define the structural congruence $\equiv$ as a reflexive, transitive congruence over terms which satisfies the following additional axioms:

$$\begin{aligned}
(\leftrightarrow\text{-comm}) \quad x \leftrightarrow y &\equiv y \leftrightarrow x \\
(\nu\text{-comm}) \quad \nu x.(P \mid Q) &\equiv \nu x.(Q \mid P) \\
(\nu\text{-assoc}_1) \quad \nu x.(P \mid \nu y.(Q \mid R)) &\equiv \nu y.(\nu x.(P \mid Q) \mid R) \quad \text{if } x \notin R \text{ and } y \notin P
\end{aligned}$$

We deviate from the original presentation of CP here, as the structural congruence defined by Wadler [14] does not include $(\leftrightarrow\text{-comm})$. We include it here because

it fits well with our notion of links, and it simplifies our reduction system and the proofs of meta-theoretical properties of CP. We do not add an axiom for $(\nu\text{-assoc}_2)$, as it follows from definition 2.2. Throughout this dissertation, we will leave uses of the transitivity and congruence rules implicit.

Lemma 2.3 ($\nu\text{-assoc}_2$)

If $x \notin R$ and $y \notin P$, then $\nu y.(\nu x.(P \mid Q) \mid R) \equiv \nu x.(P \mid \nu y.(Q \mid R))$. □

Proof.

$$\begin{aligned}
\nu y.(\nu x.(P \mid Q) \mid R) &\equiv && \text{by } (\nu\text{-comm}) \\
\nu y.(\nu x.(Q \mid P) \mid R) &\equiv && \text{by } (\nu\text{-comm}) \\
\nu y.(R \mid \nu x.(Q \mid P)) &\equiv && \text{by } (\nu\text{-assoc}_1) \\
\nu x.(\nu y.(R \mid Q) \mid P) &\equiv && \text{by } (\nu\text{-comm}) \\
\nu x.(P \mid \nu y.(R \mid Q)) &\equiv && \text{by } (\nu\text{-comm}) \\
\nu x.(P \mid \nu y.(Q \mid R)) &&&
\end{aligned}$$

The side conditions for $(\nu\text{-assoc}_1)$ are given. ■

Furthermore, structural congruence is a symmetric relation.

Theorem 2.4 (Symmetry)

If $P \equiv Q$, then $Q \equiv P$. □

Proof. By induction on the structure of the equivalence proof. The only interesting case is $(\nu\text{-assoc}_1)$, which follows from lemma 2.3. ■

Channels in RCP are typed using a session type system which corresponds to RLL, the multiplicative, additive fragment of linear logic. The types are defined by the following grammar:

Definition 2.5 (Types)

A, B, C	$:= A \otimes B$	pair of independent processes
	$ A \wp B$	pair of interdependent processes
	$ \mathbf{1}$	unit for $\otimes$
	$ \perp$	unit for $\wp$
	$ A \oplus B$	internal choice
	$ A \& B$	external choice
	$ \mathbf{0}$	unit for $\oplus$
	$ \top$	unit for $\&$

Duality plays a crucial role in both linear logic and session types. In RCP, the two endpoints of a channel are assigned dual types. This ensures that, for instance, whenever a process *sends* across a channel, the process on the other end of that channel is waiting to *receive*. Each type A has a dual, written $A^\perp$.

Definition 2.6 (Duality)

$$\begin{aligned}
(A \otimes B)^\perp &= A^\perp \wp B^\perp & \mathbf{1}^\perp &= \perp \\
(A \wp B)^\perp &= A^\perp \otimes B^\perp & \perp^\perp &= \mathbf{1} \\
(A \oplus B)^\perp &= A^\perp \& B^\perp & \mathbf{0}^\perp &= \top \\
(A \& B)^\perp &= A^\perp \oplus B^\perp & \top^\perp &= \mathbf{0}
\end{aligned}$$

Duality is an involutive function.

Lemma 2.7 (Involutive) We have $A^{\perp\perp} = A$. □

Proof. By induction on the structure of the type A . ■

Environments associate channels with types.

Definition 2.8 (Environments)

We define environments as follows:

$$\Gamma, \Delta, \Theta ::= x_1 : A_1 \dots x_n : A_n$$

Names in environments must be unique, and environments Γ and Δ can only be combined as Γ, Δ if $\text{fv}(\Gamma) \cap \text{fv}(\Delta) = \emptyset$.

Typing judgements associate processes with collections of typed channels.

Definition 2.9 (Typing judgements)

A typing judgement $P \vdash x_1 : A_1 \dots x_n : A_n$ denotes that the process P communicates along channels $x_1 \dots x_n$ following protocols $A_1 \dots A_n$. Typing judgements can be constructed using the inference rules in fig. 2.1.

$$\begin{array}{c}
\frac{}{x \leftrightarrow y \vdash x : A, y : A^\perp} \text{Ax} \quad \frac{P \vdash \Gamma, x : A \quad Q \vdash \Delta, x : A^\perp}{\nu x.(P \mid Q) \vdash \Gamma, \Delta} \text{CUT} \\
\\
\frac{P \vdash \Gamma, y : A \quad Q \vdash \Delta, x : B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x : A \otimes B} (\otimes) \quad \frac{P \vdash \Gamma, y : A, x : B}{x(y).P \vdash \Gamma, x : A \wp B} (\wp) \\
\\
\frac{}{x[] . 0 \vdash x : \mathbf{1}} (\mathbf{1}) \quad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp} (\perp) \\
\\
\frac{P \vdash \Gamma, x : A}{x[\text{inl}].P \vdash \Gamma, x : A \oplus B} (\oplus_1) \quad \frac{P \vdash \Gamma, x : B}{x[\text{inr}].P \vdash \Gamma, x : A \oplus B} (\oplus_2) \\
\\
\frac{P \vdash \Gamma, x : A \quad Q \vdash \Delta, x : B}{\text{case } x \{P; Q\} \vdash \Gamma, \Delta, x : A \& B} (\&) \\
\\
(\text{no rule for } \mathbf{0}) \quad \frac{}{\text{case } x \{\} \vdash \Gamma, x : \top} (\top)
\end{array}$$

Figure 2.1: Typing judgement for the multiplicative applicative subset of RCP.

Reductions relate processes with their reduced forms. They are defined as follows:

Definition 2.10 (Term reduction)

A reduction $P \Rightarrow P'$ denotes that the process P can reduce to the process P' in a single step. Reductions can only be constructed using the rules in figs. 2.2 and 2.3. The relation $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$.

We will discuss the interpretations of each connective, together with their typing and reduction rules, in sections 2.1.2 to 2.1.4.

2.1.2 Multiplicatives and in- and interdependence

The multiplicatives $(\otimes, \wp)$ deal with independence and interdependence:

$$\begin{array}{lll}
(\leftrightarrow_1) & \nu x.(w \leftrightarrow x \mid P) & \Longrightarrow P\{w/x\} \\
(\leftrightarrow_2) & \nu x.(x \leftrightarrow w \mid P) & \Longrightarrow P\{w/x\} \\
(\beta \otimes \mathfrak{A}) & \nu x.(x[y].(P \mid Q) \mid x(z).R) & \Longrightarrow \nu y.(P \mid \nu x.(Q \mid R\{y/z\})) \\
(\beta \mathbf{1} \perp) & \nu x.(x[\perp].0 \mid x().P) & \Longrightarrow P \\
(\beta \oplus \&_1) & \nu x.(x[\text{inl}].P \mid \text{case } x \{Q; R\}) & \Longrightarrow \nu x.(P \mid Q) \\
(\beta \oplus \&_2) & \nu x.(x[\text{inr}].P \mid \text{case } x \{Q; R\}) & \Longrightarrow \nu x.(P \mid R)
\end{array}$$

$$\begin{array}{c}
\frac{P \Longrightarrow P'}{\nu x.(P \mid Q) \Longrightarrow \nu x.(P' \mid Q)} (\gamma \nu) \\
\frac{P \equiv Q \quad Q \Longrightarrow Q' \quad Q' \equiv P'}{P \Longrightarrow P'} (\gamma \equiv)
\end{array}$$

Figure 2.2: Term reduction rules for RCP.

$$\begin{array}{lll}
(\kappa \otimes_1) & \nu x.(y[z].(P \mid Q) \mid R) & \Longrightarrow y[z].(\nu x.(P \mid R) \mid Q) \quad \text{if } x \notin Q \\
(\kappa \otimes_2) & \nu x.(y[z].(P \mid Q) \mid R) & \Longrightarrow y[z].(P \mid \nu x.(Q \mid R)) \quad \text{if } x \notin P \\
(\kappa \mathfrak{A}) & \nu x.(y(z).P \mid R) & \Longrightarrow y(z).\nu x.(P \mid R) \\
(\kappa \perp) & \nu x.(y().P \mid R) & \Longrightarrow y().\nu x.(P \mid R) \\
(\kappa \oplus_1) & \nu x.(y[\text{inl}].P \mid R) & \Longrightarrow y[\text{inl}].\nu x.(P \mid R) \\
(\kappa \oplus_2) & \nu x.(y[\text{inr}].P \mid R) & \Longrightarrow y[\text{inr}].\nu x.(P \mid R) \\
(\kappa \&) & \nu x.(\text{case } y \{P; Q\} \mid R) & \Longrightarrow \text{case } y \{\nu x.(P \mid R); \nu x.(Q \mid R)\} \\
(\kappa \top) & \nu x.(\text{case } y \{\} \mid R) & \Longrightarrow \text{case } y \{\}
\end{array}$$

Figure 2.3: Commutative conversions for CP.

- A channel of type $A \otimes B$ represents a pair of channels, which communicate with two independent processes—that is to say, two processes who share no channels. A process acting on a channel of type $A \otimes B$ will send one endpoint of a fresh channel, and then split into a pair of independent processes. One of these processes will be responsible for an interaction of type A over the fresh channel, while the other process continues to interact as B .
- A channel of type $A \mathfrak{A} B$ represents a pair of interdependent channels, which are used within a single process. A process acting on a channel of type $A \mathfrak{A} B$ will receive a channel to act on, and communicate on its channels in whatever order it pleases. This means that the usage of one channel can

depend on that of another—e.g. the interaction of type B could depend on the result of the interaction of type A , or vice versa, and if A and B are complex types, their interactions could likewise interweave in complex ways.

While the rules for $\otimes$ and $\wp$ introduce input and output operations, these are inessential—the essential distinction lies two in the fact that $(\otimes)$ composes two independent processes, and therefore *must* split the environment between them, whereas $(\wp)$ uses a single process, which then can—and must—use all the channels in the environment.

$$\frac{P \vdash \Gamma, y:A \quad Q \vdash \Delta, x:B}{x[y].(P \mid Q) \vdash \Gamma, \Delta, x:A \otimes B} (\otimes) \quad \frac{P \vdash \Gamma, y:A, x:B}{x(y).P \vdash \Gamma, x:A \wp B} (\wp)$$

The β -reduction rule for terms introduced by $(\otimes)$ and $(\wp)$ implements the behaviour outlined above:

$$\nu x.(x[y].(P \mid Q) \mid x(z).R) \Longrightarrow \nu y.(P \mid \nu x.(Q \mid R\{y/z\}))$$

The function $P\{x/y\}$ denotes the substitution of the name x for the name y in the term P . The rules for the multiplicative units $(\mathbf{1}, \perp)$ follow the same pattern, except for the nullary instead of the binary case:

- A term constructed by $(\mathbf{1})$ must composes *zero* independent processes, and thus must halt. Furthermore, it must be able to split its environment between zero processes, and thus its environment must be empty.
- A term constructed by $(\perp)$, on the other hand, consists of a single process, which is not further restricted.

The rules for $\mathbf{1}$ and $\perp$ introduce a nullary send and receive operation, such as those found in the polyadic π -calculus [11].

$$\frac{}{x[] . 0 \vdash x:\mathbf{1}} (\mathbf{1}) \quad \frac{P \vdash \Gamma}{x().P \vdash \Gamma, x:\perp} (\perp)$$

The β -reduction rule for terms introduced by $(\mathbf{1})$ and $(\perp)$ implements the behaviour outlined above:

$$\nu x.(x[] . 0 \mid x().P) \Longrightarrow P$$

2.1.3 Additives and choice

The additives $(\oplus, \&)$ deal with choice:

- A process acting on a channel of type $A \oplus B$ either sends the value **inl** to select an interaction of type A or the value **inr** to select one of type B .
- A process acting on a channel of type $A \& B$ receives such a value, and then offers an interaction of either type A or B , correspondingly.

In essence, the additive operations implement sending and receiving of a single bit of information (**inl** or **inr**) and branching based on the value of that bit.

$$\frac{P \vdash \Gamma, x:A}{x[\text{inl}].P \vdash \Gamma, x:A \oplus B} (\oplus_1) \quad \frac{P \vdash \Gamma, x:A \quad Q \vdash \Delta, x:B}{\text{case } x \{P; Q\} \vdash \Gamma, \Delta, x:A \& B} (\&)$$

The rule for constructing a process which sends **inr**, $(\oplus_2)$, has been omitted, but can be found in fig. 2.1. The β -reduction rules for terms introduced by $(\oplus_1)$, $(\oplus_2)$ and $(\&)$ implement the behaviour outlined above.

$$\begin{aligned} \nu x.(x[\text{inl}].P \mid \text{case } x \{Q; R\}) &\Longrightarrow \nu x.(P \mid Q) \\ \nu x.(x[\text{inr}].P \mid \text{case } x \{Q; R\}) &\Longrightarrow \nu x.(P \mid R) \end{aligned}$$

The rules for the additive units (**0**, $\top$) follow the same pattern, except for a nullary choice:

- There is *no* rule for **0**, as a process acting on a channel of that type would have to select one of *zero* options, which is clearly impossible.
- A process acting on a channel of type $\top$ will wait to receive a choice of out *zero* options. Since this will clearly never arrive, we have two options: either we block, waiting forever, or we simply crash.

It may seem odd at first to include a type for the process which cannot possibly exist, and for the process which waits forever, but these make sensible units for choice. When offered a choice of type $A \oplus \mathbf{0}$, one can either choose to interact as A , or choose to commit to doing the impossible. Similarly, when offering a choice of type $A \& \top$, one can safely implement the right branch with a process which waits forever, as no sound process will ever be able to select that branch anyway.

$$(\text{no rule for } \mathbf{0}) \quad \frac{}{\text{case } x \{ \} \vdash \Gamma, x:\top} (\top)$$

As there is no way to construct a process of type **0**, there is no reduction rule for the additive units.

2.1.4 Structural rules and duality

Duality plays a crucial role in session type systems. In section 2.1.3, we saw that duality ensures a process offering a choice is always matched with a process making a choice. In section 2.1.2, we saw that it also ensures that, for instance a process which uses communication on x to decide what to send on y is matched with a pair of independent processes on x and y , a property which is crucial to deadlock freedom, as it prevents circular dependencies.

Duality appears in the typing rules for two RCP term constructs. Forwarding, $x \leftrightarrow y$, connects two dual channels with dual endpoints, while composition, $\nu x.(P \mid Q)$, composes two processes P and Q with a shared channel x , requiring that they follow dual protocols on x .

$$\frac{}{x \leftrightarrow y \vdash x:A, y:A^\perp} \text{Ax} \quad \frac{P \vdash \Gamma, x:A \quad Q \vdash \Delta, x:A^\perp}{\nu x.(P \mid Q) \vdash \Gamma, \Delta} \text{Cut}$$

There are two reduction rules which deal with the interactions between forwarding and compositions. These implement the intuition that if a process is meant to communicate on x , x is forwarding to y , and nobody else is listening on x , then the process might as well start communicating on y .

$$\nu x.(w \leftrightarrow x \mid P) \Longrightarrow P\{w/x\}$$

We can do this because RCP implements a binary session type system, meaning that each communication has only two participants, and therefore we know that no other process is communicating on x .

Wadler [14] has two reduction rules which deal with links. The first of these can be seen above, and the second of these deals with the case in which the link is flipped. As we consider links commutative, this rule is derivable.

2.1.5 Commuting conversions

The commuting conversions are not reductions typically found in the π -calculus. Instead, they are based on permutation cuts, which are commonly used in logic to define a procedure for cut elimination. These cut elimination steps push a cut deeper into a term, past unrelated inference rules. When viewed from the perspective of process calculus, they allow us to pull actions upwards, past unrelated cuts. The commuting conversion are defined in fig. 2.3.

2.1.6 Example

The multiplicatives are responsible for structuring communication, and it is this structure which rules out deadlocked interactions. Let us go back to our example of a deadlocked interaction from chapter 1:

$$(x\langle z \rangle.\bar{y}\langle \text{cake} \rangle.\text{John} \mid y\langle w \rangle.\bar{x}\langle \text{money} \rangle.\text{Mary})$$

If we want to translate this interaction to CP, we run into a problem: there is no *plain* sending construct in CP—we only have $x[y].(P \mid Q)$, which requires that the remainder of the interaction is split in two independent processes. This enforces a certain structure on the program. Either John will already have to have the cake in his hands, or Mary will already have to have the money in the bank. We model the second scenario below, assuming ,  and  are processes modeling John, Mary, and Mary's bank, and  and  are the types of two channels which give access to a slice of cake and appropriate payment.

$$\frac{\frac{\text{John} \vdash \Gamma, y : \text{money}^\perp, x : \text{cake}}{x(y).\text{John} \vdash \Gamma, x : \text{money}^\perp \text{ cake}} \wp \quad \frac{\frac{\text{Bank} \vdash \Delta, z : \text{money} \quad \text{Mary} \vdash \Theta, x : \text{cake}^\perp}{x[z].(\text{Bank} \mid \text{Mary}) \vdash \Delta, x : \text{money} \otimes \text{cake}^\perp} \otimes}{\nu x.(x(y).\text{John} \mid x[z].(\text{Bank} \mid \text{Mary})) \vdash \Gamma, \Delta, \Theta} \text{CUT}$$

The resulting process reduces, as expected:

$$\nu x.(x(y).\text{John} \mid x[z].(\text{Bank} \mid \text{Mary})) \Rightarrow \nu y.(\text{Bank} \mid \nu x.(\text{John} \mid \text{Mary}))$$

2.1.7 Properties of RCP

In this section, we will prove three important properties of RCP, namely preservation, progress and termination.

2.1.7.1 Preservation

Preservation is the fact that term reduction preserves typing. In order to prove this, we will first need to prove that structural congruence preserves typing.

Theorem 2.11 (Preservation for $\equiv$)

If $P \vdash \Gamma$ and $P \equiv Q$, then $Q \vdash \Gamma$.

□

Proof. By induction on the structure of the equivalence. The cases for reflexivity, transitivity and congruence are trivial. The two interesting cases, for $(\nu\text{-comm})$ and $(\nu\text{-assoc}_1)$ are given in fig. 2.4 $\blacksquare$

$$\begin{aligned}
(\leftrightarrow\text{-comm}) \quad & \frac{}{x \leftrightarrow y \vdash x:A, y:A^\perp} \text{Ax} \\
\equiv \quad & \frac{\frac{}{y \leftrightarrow x \vdash y:A^\perp, x:A^{\perp\perp}} \text{Ax}}{y \leftrightarrow x \vdash y:A^\perp, x:A} \text{LEMMA 2.7} \\
(\nu\text{-comm}) \quad & \frac{P \vdash \Gamma, x:A \quad Q \vdash \Delta, x:A^\perp}{\nu x.(P \mid Q) \vdash \Gamma, \Delta} \text{CUT} \\
\equiv \quad & \frac{Q \vdash \Delta, x:A^\perp \quad \frac{P \vdash \Gamma, x:A}{P \vdash \Gamma, x:A^{\perp\perp}} \text{LEMMA 2.7}}{\nu x.(Q \mid P) \vdash \Gamma, \Delta} \text{CUT} \\
(\nu\text{-assoc}_1) \quad & \frac{P \vdash \Gamma, x:A \quad \frac{Q \vdash \Delta, x:A^\perp, y:B \quad R \vdash \Theta, y:B^\perp}{\nu y.(Q \mid R) \vdash \Delta, \Theta, x:A^\perp} \text{CUT}}{\nu x.(P \mid \nu y.(Q \mid R)) \vdash \Gamma, \Delta, \Theta} \text{CUT} \\
\equiv \quad & \frac{\frac{P \vdash \Gamma, x:A \quad Q \vdash \Delta, x:A^\perp, y:B}{\nu x.(P \mid Q) \vdash \Gamma, \Delta, y:B} \text{CUT} \quad R \vdash \Theta, y:B^\perp}{\nu y.(\nu x.(P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta} \text{CUT}
\end{aligned}$$

Figure 2.4: Type preservation for the structural congruence of RCP

Then, we can prove preservation.

Theorem 2.12 (Preservation)

If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$. $\square$

Proof. By induction on the structure of the reduction. See fig. 2.5 for $(\leftrightarrow_1)$, $(\leftrightarrow_2)$, and the β -reduction rules, and figs. 2.6 and 2.7 for the commutative conversions. The case for $(\gamma\nu)$ is trivial by the induction hypothesis, and the case for $(\gamma\equiv)$ is trivial by the induction hypothesis and theorem 2.11. $\blacksquare$

2.1.7.2 Progress

Progress is the fact that every term is either in some canonical form, or can be reduced further. In order for a statement of progress to make sense, we need a definition of canonical form. The canonical form used by CP is “any term which is not a cut.” We will refer to this canonical form as *weak head normal form*, for its relation to the eponymous λ -calculus normal form.

Definition 2.13 (Weak head normal form)

A process P is in weak head normal form if it is not a cut.

As CP has a tight correspondence with classical linear logic, so does its proof of progress have a tight correspondence with (part of) the procedure proof normalisation for classical linear logic for classical linear logic [6].

Theorem 2.14 (Progress, WHNF)

If $P \vdash \Gamma$, then either P is in weak head normal form, or there exists a P' such that $P \Rightarrow P'$. □

Proof. By induction on the structure of P . The only interesting case is the case where P is a cut $\nu x.(P' \mid Q')$. In every other case, P is in weak head normal form. There are three possibilities:

- Both P' and Q' act on x .
We apply one of the β -reduction rules.
- Either P' or Q' acts on an external channel.
We apply one of the commuting conversions.
- Otherwise one or both of P' and Q' are themselves cuts.
We apply the induction hypothesis. ■

If we extend the reduction system with all congruence rules—not just $(\gamma\nu)$ for reduction under cuts, but for reduction under any term context—then we can strengthen our canonical form, and extend our proof for progress to the *full* proof normalisation procedure.

Definition 2.15 (Normal form)

A process P is in normal form if it does not contain any cuts.

Theorem 2.16 (Progress, NF)

If $P \vdash \Gamma$, then either P is in normal form, or there exists a P' such that $P \Longrightarrow P'$. □

Proof. Either P is in normal form, or there is a cut *somewhere* in P . If there is a cut, then we obtain a reduction by theorem 2.14 and congruence. ■

Wadler [14] opts to leave these additional congruence rules out, because “such rules do not correspond well to our notion of computation on processes”, and his choice is analogous to a common practice in the λ -calculus to not allow reduction under lambdas.

It might not be so odd to allow these reductions in the context of CP. If one conceives of the two sides of a parallel composition in $y(z).\nu x.(P \mid Q)$ as separate processes, both waiting on an external communication on y , it does not seem odd to allow the communication on x to happen. It is simply eager a form of evaluation.

2.1.7.3 Termination

Termination is the fact that if we iteratively apply progress to obtain a reduction, and apply that reduction, we will eventually end up with a term in canonical form. Its proof is quite simple, owing to the fact that our reduction rules were all inspired by cut reductions from classical linear logic.

Theorem 2.17 (Termination)

If $P \vdash \Gamma$, then there are no infinite $\Longrightarrow$ reduction sequences. □

Proof. Every reduction reduces a single cut to zero, one or two cuts. However, each of these cuts is *smaller*, in the sense that the type of the channel on which the communication takes place is smaller, as each reduction eliminates a connective—see figs. 2.5 to 2.7. Furthermore, each instance of the structural congruence preserves the size of the cut—see fig. 2.4. Therefore, there cannot be an infinite reduction sequence. ■

2.2 Non-determinism, logic, and session types

In recent work, we have seen the extension of π DILL and CP with operators for non-deterministic behaviour [1, 3, 4]. These extensions all implement an operator known as non-deterministic local choice. While this operator is written as $P + Q$, it should not be confused with input-guarded choice [12] from the π -calculus. Essentially, non-deterministic local choice can be summarised by the following typing and reduction rules:

$$\frac{P \vdash \Gamma \quad Q \vdash \Gamma}{P + Q \vdash \Gamma} \qquad \begin{array}{l} P + Q \Longrightarrow P \\ P + Q \Longrightarrow Q \end{array}$$

There are some problems with non-deterministic local choice. First of all, the non-determinism arises from the fact that for any term $P + Q$, two different reduction rules apply simultaneously. These reduction rules are written specifically to introduce non-determinism. This is unlike the π -calculus, where non-determinism arises due to multiple processes communicating on a single, shared channel. We can easily implement this operator in the π -calculus, using a nullary communication:

$$\begin{array}{c} (\bar{x}\langle \rangle.0 \mid x\langle \rangle.P \mid x\langle \rangle.Q) \\ \Downarrow_{\star} \\ (P \mid x\langle \rangle.Q) \quad \text{or} \quad (x\langle \rangle.P \mid Q) \end{array}$$

In this implementation, the process $\bar{x}\langle \rangle.0$ will “unlock” either P or Q , leaving the other process deadlocked. Or we could use input-guarded choice:

$$(\bar{x}\langle \rangle.0 \mid (x\langle \rangle.P + x\langle \rangle.Q))$$

However, there are many non-deterministic processes in the π -calculus which are awkward to encode using non-deterministic local choice. Let us look at our example:

$$\begin{array}{c} (\bar{x}\langle \text{🍰} \rangle.\bar{x}\langle \text{👏} \rangle.\text{🇧🇪} \mid x\langle y \rangle.\text{👤} \mid x\langle z \rangle.\text{👩}) \\ \Downarrow_{\star} \\ (\text{🇧🇪} \mid \text{👤}\{\text{🍰}/y\} \mid \text{👩}\{\text{👏}/z\}) \quad \text{or} \quad (\text{🇧🇪} \mid \text{👤}\{\text{👏}/y\} \mid \text{👩}\{\text{🍰}/z\}) \end{array}$$

This non-deterministic interaction involves communication. If we wanted to write down a process which reduced to the same result using non-deterministic local

choice, we would have to write the following process:

$$\begin{aligned}
 & (\text{👩} \mid \text{👨}\{\text{🍰}/y\} \mid \text{👧}\{\text{👏}/z\}) + (\text{👩} \mid \text{👨}\{\text{👏}/y\} \mid \text{👧}\{\text{🍰}/z\}) \\
 & \quad \Downarrow_{\star} \\
 & (\text{👩} \mid \text{👨}\{\text{🍰}/y\} \mid \text{👧}\{\text{👏}/z\}) \quad \text{or} \quad (\text{👩} \mid \text{👨}\{\text{👏}/y\} \mid \text{👧}\{\text{🍰}/z\})
 \end{aligned}$$

In essence, instead of modelling a non-deterministic interaction, we are enumerating the outcomes of such an interaction. This means non-deterministic local choice does not adequately model non-determinism in the way the π -calculus does. Enumerating all possible outcomes becomes worse the more processes are involved in an interaction. Imagine a scenario the following scenario:

Three customers, Alice, John and Mary, have a craving for cake. Should cake be sold out, however, well... a doughnut will do. They prepare to order their goods via an online store. Unfortunately, they all decide to use the same *shockingly* under-stocked store, which has only one slice of cake, and a single doughnut. After that, all it can deliver is disappointment.

We can model this scenario in the π -calculus, assuming 🧑, 🧑, 🧑, and 🏪 are four processes modelling Alice, John, Mary and the store, and 🍰, 🍩, and 👏 are three channels giving access to a slice of cake, a so-so doughnut, and disappointment, respectively.

$$\begin{aligned}
 & (\bar{x}\langle\text{🍰}\rangle.\bar{x}\langle\text{🍩}\rangle.\bar{x}\langle\text{👏}\rangle.\text{🏪} \mid x\langle y\rangle.\text{🧑} \mid x\langle z\rangle.\text{🧑} \mid x\langle w\rangle.\text{🧑}) \\
 & \quad \Downarrow_{\star} \\
 & (\text{🏪} \mid \text{🧑}\{\text{🍰}/y\} \mid \text{🧑}\{\text{🍩}/z\} \mid \text{🧑}\{\text{👏}/w\}) \text{ or } (\text{🏪} \mid \text{🧑}\{\text{🍰}/y\} \mid \text{🧑}\{\text{👏}/z\} \mid \text{🧑}\{\text{🍩}/w\}) \\
 & (\text{🏪} \mid \text{🧑}\{\text{🍩}/y\} \mid \text{🧑}\{\text{👏}/z\} \mid \text{🧑}\{\text{🍰}/w\}) \text{ or } (\text{🏪} \mid \text{🧑}\{\text{🍩}/y\} \mid \text{🧑}\{\text{🍰}/z\} \mid \text{🧑}\{\text{👏}/w\}) \\
 & (\text{🏪} \mid \text{🧑}\{\text{👏}/y\} \mid \text{🧑}\{\text{🍰}/z\} \mid \text{🧑}\{\text{🍩}/w\}) \text{ or } (\text{🏪} \mid \text{🧑}\{\text{👏}/y\} \mid \text{🧑}\{\text{🍩}/z\} \mid \text{🧑}\{\text{🍰}/w\})
 \end{aligned}$$

With the addition of one process, Alice, we have increased the number of possible outcomes enormously! In general, the number of outcomes for these types of scenarios is $n!$, where n is the number of processes. This means that if we wish to translate any non-deterministic process to one using non-deterministic local choice, we can expect a factorial growth in the size of the term!

$$\begin{array}{l}
(\leftrightarrow_1) \quad \frac{\frac{}{w \leftrightarrow x \vdash w : A, x : A^\perp} \text{Ax} \quad R \vdash x : A, \Gamma}{\nu x.(w \leftrightarrow x \mid R) \vdash w : A, \Gamma} \text{CUT} \\
\\
\Rightarrow \quad R\{w/x\} \vdash w : A, \Gamma \\
\\
(\beta \otimes \wp) \quad \frac{\frac{P \vdash \Gamma, x : A^\perp, y : B^\perp}{y(x).P \vdash \Gamma, y : A^\perp \wp B^\perp} \wp \quad \frac{Q \vdash \Delta, x : A \quad R \vdash \Theta, y : B}{y[x].(| Q \mid R) \vdash \Delta, \Theta, y : A \otimes B} \otimes}{\nu y.(y(x).P \mid y[x].(Q \mid R)) \vdash \Gamma, \Delta, \Theta} \text{CUT} \\
\\
\Rightarrow \quad \frac{\frac{P \vdash \Gamma, x : A^\perp, y : B^\perp \quad Q \vdash \Delta, x : A}{\nu x.(P \mid Q) \vdash \Gamma, \Delta, y : B^\perp} \text{CUT} \quad R \vdash \Theta, y : B}{\nu y.(\nu x.(P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta} \text{CUT} \\
\\
(\beta 1 \perp) \quad \frac{\frac{P \vdash \Gamma}{x().P \vdash \Gamma, x : \perp} \perp \quad \frac{}{x[] . 0 \vdash x : \mathbf{1}} \mathbf{1}}{\nu x.(x[] . 0 \mid x().P) \vdash \Gamma} \text{CUT} \\
\\
\Rightarrow \quad P \vdash \Gamma \\
\\
(\beta \oplus \&_1) \quad \frac{\frac{P \vdash \Gamma, x : A^\perp \quad Q \vdash \Gamma, x : B^\perp}{\text{case } x \{P; Q\} \vdash \Gamma, x : A^\perp \& B^\perp} \& \quad \frac{R \vdash \Delta, x : A}{x[\text{inl}].R \vdash \Delta, x : A \oplus B} \oplus_1}{\nu x.(\text{case } x \{P; Q\} \mid x[\text{inl}].R) \vdash \Gamma, \Delta} \text{CUT} \\
\\
\Rightarrow \quad \frac{P \vdash \Gamma, x : A^\perp \quad R \vdash \Delta, x : A}{\nu x.(P \mid R) \vdash \Gamma, \Delta} \text{CUT} \\
\\
(\beta \oplus \&_2) \quad (\text{as above})
\end{array}$$

Figure 2.5: Type preservation for the $(\leftrightarrow_1)$ and β -reduction rules of RCP

$$\begin{aligned}
& (\kappa \otimes_1) \quad \frac{\frac{P \vdash \Gamma, x:A, z:B \quad Q \vdash \Delta, y:C}{y[z].(P \mid Q) \vdash \Gamma, \Delta, y:B \otimes C} \otimes \quad R \vdash \Theta, x:A^\perp}{\nu x.(y[z].(P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta, y:B \otimes C} \text{CUT} \\
& \Rightarrow \quad \frac{\frac{P \vdash \Gamma, x:A, z:B \quad R \vdash \Theta, x:A^\perp}{\nu x.(P \mid Q) \vdash \Gamma, \Theta, z:B} \text{CUT} \quad Q \vdash \Delta, y:C}{y[z].(\nu x.(P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta, y:B \otimes C} \otimes \\
& (\kappa \otimes_2) \quad (\text{as above}) \\
& (\kappa \wp) \quad \frac{\frac{P \vdash \Gamma, x:A, z:B, y:C}{y(z).P \vdash \Gamma, x:A, y:B \wp C} \wp \quad R \vdash \Theta, x:A^\perp}{\nu x.(y(z).P \mid R) \vdash \Gamma, \Theta, y:B \wp C} \text{CUT} \\
& \Rightarrow \quad \frac{\frac{P \vdash \Gamma, x:A, z:B, y:C \quad R \vdash \Theta, x:A^\perp}{\nu x.(P \mid R) \vdash \Gamma, \Theta, z:B, y:C} \text{CUT}}{y(z).\nu x.(P \mid R) \vdash \Gamma, \Theta, y:B \wp C} \wp \\
& (\kappa \perp) \quad \frac{\frac{P \vdash \Gamma, x:A}{y().P \vdash \Gamma, x:A, y:\perp} \perp \quad R \vdash \Theta, x:A^\perp}{\nu x.(y().P \mid R) \vdash \Gamma, \Theta, y:\perp} \text{CUT} \\
& \Rightarrow \quad \frac{\frac{P \vdash \Gamma, x:A \quad R \vdash \Theta, x:A^\perp}{\nu x.(P \mid R) \vdash \Gamma, \Theta} \text{CUT}}{y().\nu x.(P \mid R) \vdash \Gamma, \Theta, y:\perp} \perp
\end{aligned}$$

Figure 2.6: Type preservation for the commuting conversions of CP

$$\begin{aligned}
(\kappa\oplus_1) \quad & \frac{\frac{P \vdash \Gamma, x:A, y:B}{y[\text{inl}].P \vdash \Gamma, x:A, y:B \oplus C} \oplus_1 \quad R \vdash \Theta, x:A^\perp}{\nu x.(y[\text{inl}].P \mid R) \vdash \Gamma, \Theta, y:B \oplus C} \text{CUT} \\
\Rightarrow \quad & \frac{\frac{P \vdash \Gamma, x:A, y:B \quad R \vdash \Theta, x:A^\perp}{\nu x.(P \mid R) \vdash \Gamma, \Theta, y:B} \text{CUT}}{y[\text{inl}].\nu x.(P \mid R) \vdash \Gamma, \Theta, y:B \oplus C} \oplus_1 \\
(\kappa\oplus_2) \quad & \text{(as above)} \\
(\kappa\&) \quad & \frac{\frac{P \vdash \Gamma, x:A, y:B \quad Q \vdash \Gamma, x:A, y:C}{\text{case } y \{P; Q\} \vdash \Gamma, x:A, y:B \& C} \& \quad R \vdash \Theta, x:A^\perp}{\nu x.(\text{case } y \{P; Q\} \mid R) \vdash \Gamma, \Theta, y:B \& C} \text{CUT} \\
\Rightarrow \quad & \frac{\frac{P \vdash \Gamma, x:A, y:B \quad R \vdash \Theta, x:A^\perp}{\nu x.(P \mid R) \vdash \Gamma, \Theta, y:B} \text{CUT} \quad \frac{Q \vdash \Gamma, x:A, y:C \quad R \vdash \Theta, x:A^\perp}{\nu x.(Q \mid R) \vdash \Gamma, \Theta, y:C} \text{CUT}}{\text{case } y \{\nu x.(P \mid R); \nu x.(Q \mid R)\} \vdash \Gamma, \Theta, y:B \& C} \& \\
(\kappa\top) \quad & \frac{\frac{\text{case } y \{\} \vdash \Gamma, x:A, y:\top}{\nu x.(\text{case } y \{\} \mid R) \vdash \Gamma, \Theta, y:\top} \top \quad R \vdash \Theta, x:A^\perp}{\text{case } y \{\} \vdash \Gamma, \Theta, y:\top} \text{CUT} \\
\Rightarrow \quad & \frac{}{\text{case } y \{\} \vdash \Gamma, \Theta, y:\top} \top
\end{aligned}$$

Figure 2.7: Type preservation for the commuting conversions of CP (cont'd)

Chapter 3

CP as a type system for the π -calculus

CP has a tight correspondence with classical linear logic. This has many advantages. It is deadlock free and terminating, and as seen in section 2.1.7, the proofs of its meta-theoretical properties are concise and elegant. The price paid for this is a weaker correspondence with the π -calculus. It would be useful to be able to think of CP as a type system for the π -calculus. However, as it stands there are many differences between these systems. For instance, in CP name restriction and parallel composition are considered a single, atomic construct, and it uses case statements instead of input-guarded choice. Most prominent among the differences, however, are the commuting conversions. These reduction rules are taken directly from the proof normalisation procedure of classical linear logic, and do not correspond to any reductions in the π -calculus.

Lindley and Morris [9] observed that, using a different reduction strategy, which more closely resembles that of the π -calculus, we can ensure that the commuting conversions are always applied *last*. That is to say, they define two separate reduction relations: $\longrightarrow_C$ for $(\leftrightarrow_1)$ and β -reductions, and $\longrightarrow_{CC}$ for commuting conversions, and show that any sequence of reductions is equivalent to one of the form:

$$P \longrightarrow_C \cdots \longrightarrow_C Q \longrightarrow_{CC} \cdots \longrightarrow_{CC} R$$

In this dissertation, we use a reduction strategy which follows Lindley and Morris [9], but drop the suffix of commutative conversions. Consequently, we can drop

the commuting conversions from our reduction system, which therefore more tightly corresponds to that of the π -calculus. The price we pay for this is a weaker correspondence to classical linear logic. This shows in our notion of canonical form, which is weaker, and in our proof of progress, which is much more involved.

Whenever we refer to CP or RCP, for the remainder of this dissertation, we refer to the variant *without* commuting conversions, i.e. which uses the following definition of term reduction.

Definition 3.1 (Term reduction)

A reduction $P \Rightarrow P'$ denotes that the process P can reduce to the process P' in a single step. Reductions can only be constructed using the rules in fig. 2.2.

The relation $\Rightarrow^*$ is the reflexive, transitive closure of $\Rightarrow$.

This chapter will proceed as follows. In section 3.1, we will describe what it means for a term to be in canonical form. In section 3.2, we will define evaluation contexts. In section 3.3, we will give a new proof of progress, which follows Lindley and Morris [9].

3.1 Canonical forms

The reduction strategy described by Lindley and Morris applies $(\leftrightarrow_1)$ and β -reductions until the process blocks on one or more external communications, and then applies the commuting conversions to bubble one of those external communications to the front of the term. This allows them to define canonical forms as any term which is not a cut. For us, terms in canonical form will be those terms which are blocked on an external communication, before any commuting conversions are applied. In this section, we will describe the form of such terms.

We have informally used the phrase “act on” in previous sections. It is time to formally define what it means when we say a process *acts on* some channel.

Definition 3.2 (Action)

A process P *acts on* a channel x if it is in one of the following forms:

- $x \leftrightarrow y$
- $x(y).P'$
- $x[\text{inl}].P'$
- $y \leftrightarrow x$
- $x[].0$
- $x[\text{inr}].P'$
- $x[y].(P' \mid Q')$
- $x().P'$
- $\text{case } x \{P'; Q'\}$

- $\text{case } x \{ \}$

We say a process P is an *action* if it acts on some channel x .

Furthermore, we will need the notion of an *evaluation prefix*. Intuitively, evaluation prefixes are multi-holed contexts consisting solely of cuts. We will use evaluation prefixes in order to have a view of every *action* in a process at once.

Definition 3.3 (Evaluation prefixes)

We define evaluation prefixes as:

$$G, H := \square \mid \nu x.(G \mid H)$$

The $\square$ construct represents a hole.

Definition 3.4 (Plugging)

We define plugging for an evaluation prefix with n holes as:

$$\begin{aligned} \square[R] &:= R \\ \nu x.(G \mid H)[R_1 \dots R_m, R_{m+1} \dots R_n] &:= \nu x.(G[R_1 \dots R_m] \mid H[R_{m+1} \dots R_n]) \end{aligned}$$

Note that in the second case, G is an evaluation prefix with m holes, and H is an evaluation prefix with $(n - m)$ holes.

Intuitively, we can say that every term of the form $G[P_1 \dots P_n]$ is equivalent to some term of the form $\nu x_1.(P_1 \mid \nu x_2.(P_2 \mid \dots \nu x_n.(P_{n-1} \mid P_n) \dots))$ where $x_1 \dots x_{n-1}$ are the channels bound in G . In fact, a similar equivalence was used by Lindley and Morris [9] in their semantics for CP.

Definition 3.5 (Maximum evaluation prefix)

We say that G is the evaluation prefix of P when there exist terms $P_1 \dots P_n$ such that $P = G[P_1 \dots P_n]$. We say that G is the maximum evaluation prefix if each P_i is an action.

Lemma 3.6 Every term P has a maximum evaluation prefix. □

Proof. By induction on the structure of P . ■

We can now define what it means for a term to be in canonical form. Intuitively, a process is in canonical form either when there is no top-level cut, or when it is blocked on an external communication. We state this formally as follows:

Definition 3.7 (Canonical forms)

A process P is in canonical form if it is an action, or if it is of the form $G[P_1 \dots P_n]$, where G is the maximum evaluation prefix of P , no P_i is a link, and no P_i and P_j act on the *same* channel.

3.2 Evaluation contexts

Intuitively, evaluation contexts are one-holed term contexts under which reduction can take place. For RCP, these consist solely of cuts.

Definition 3.8 (Evaluation contexts)

We define evaluation contexts as:

$$E := \square \mid \nu x.(E \mid P) \mid \nu x.(P \mid E)$$

Definition 3.9 (Plugging)

We define plugging for evaluation contexts as:

$$\begin{aligned} \square[R] &:= R \\ \nu x.(E \mid P)[R] &:= \nu x.(E[R] \mid P) \\ \nu x.(P \mid E)[R] &:= \nu x.(P \mid E[R]) \end{aligned}$$

We can prove that we can push any cut downwards under an evaluation context, as long as the channel it binds does not occur in the context itself. In proofs throughout this dissertation, we will leave uses of theorem 2.11 and theorem 2.12 implicit.

Lemma 3.10

If $\nu x.(E[P] \mid Q) \vdash \Gamma$ and $x \notin E$, then $\nu x.(E[P] \mid Q) \equiv E[\nu x.(P \mid Q)]$. $\square$

Proof. By induction on the structure of the evaluation context E .

- Case $\square$. By reflexivity.

- Case $\nu y.(E \mid R)$.

$$\begin{aligned}
\nu x.(\nu y.(E[P] \mid R) \mid Q) &\equiv \text{by } (\nu\text{-comm}) \\
\nu x.(\nu y.(R \mid E[P]) \mid Q) &\equiv \text{by } (\nu\text{-assoc}_2) \\
\nu y.(R \mid \nu x.(E[P] \mid Q)) &\equiv \text{by } (\nu\text{-comm}) \\
\nu y.(\nu x.(E[P] \mid Q) \mid R) &\equiv \text{by the induction hypothesis} \\
\nu y.(E[\nu x.(P \mid Q)] \mid R) &
\end{aligned}$$

- Case $\nu y.(R \mid E)$.

$$\begin{aligned}
\nu x.(\nu y.(R \mid E[P]) \mid Q) &\equiv \text{by } (\nu\text{-assoc}_2) \\
\nu y.(R \mid \nu x.(E[P] \mid Q)) &\equiv \text{by the induction hypothesis} \\
\nu y.(R \mid E[\nu x.(P \mid Q)]) &
\end{aligned}$$

In each case, the side conditions for $(\nu\text{-assoc}_2)$, $x \notin R$ and $y \notin Q$, can be inferred from $x \notin E$ and the fact that $\nu x.(E[P] \mid Q)$ is well-typed. ■

And vice versa. However, we will not use the following lemma in this dissertation, and leave its proof as an exercise to the reader.

Lemma 3.11

If $E[\nu x.(P \mid Q)] \vdash \Gamma$ and $x \notin E$, then $E[\nu x.(P \mid Q)] \equiv \nu x.(E[P] \mid Q)$. □

3.3 Progress

Progress is the fact that every term is either in some canonical form, or can be reduced further. There are two important lemmas which we will need in order to prove progress. These relate evaluation prefixes to evaluation contexts. Specifically, if a process under an evaluation prefix is a link, we can rewrite the entire process in such a way as to reveal the cut which introduced one of the channels acted upon by that link.

Lemma 3.12

If $G[P_1 \dots P_n] \vdash \Gamma$, and some P_i is a link $x \leftrightarrow y$, then either x and y are not bound by G , or there exist E , E' and Q such that $G[P_1 \dots P_n] \equiv E[\nu x.(E'[x \leftrightarrow y] \mid Q)]$. □

Proof. By induction on the structure of G .

- Case $\square$. Clearly x and y are not bound.
- Case $\nu z.(G'[P_1 \dots P_i \dots P_m] \mid G''[P_{m+1} \dots P_n])$.
Case $\nu z.(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_n])$.

We apply the induction hypothesis. There are two cases:

- If x and y were not bound, they remain unbound.
- If x or y is bound deeper in G , we prepend one of $\nu z.(\square \mid G''[P_{m+1} \dots P_n])$, $\nu z.(G'[P_1 \dots P_m] \mid \square)$, $(\square \mid G''[P_{m+1} \dots P_n])$, or $(G'[P_1 \dots P_m] \mid \square)$ to E . The desired equivalence follows by congruence.

- Case $\nu x.(G'[P_1 \dots P_i \dots P_m] \mid G''[P_{m+1} \dots P_n])$.
Case $\nu y.(G'[P_1 \dots P_i \dots P_m] \mid G''[P_{m+1} \dots P_n])$.
Case $\nu x.(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_n])$.
Case $\nu y.(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_n])$.

Let $E := \square$ and

let $E' := G'[P_1 \dots P_{i-1}, \square, P_{i+1}, \dots P_m]$
or $G''[P_{m+1} \dots P_{i-1}, \square, P_{i+1}, \dots P_n]$

By reflexivity and $(\nu\text{-comm})$. ■

If two processes under an evaluation prefix act on the same channel, then we can rewrite the entire process in such a way as to reveal the cut that introduced the channel.

Lemma 3.13

If $G[P_1 \dots P_n] \vdash \Gamma$, and some P_i and P_j , on different sides of at least one cut, act on the same bound channel x , then there exist E , E_i and E_j such that $G[P_1 \dots P_n] = E[\nu x.(E_i[P_i] \mid E_j[P_j])]$. □

Proof. By induction on the structure of G .

- Case $\nu x.(G'[P_1 \dots P_i \dots P_m] \mid G''[P_{m+1} \dots P_j \dots P_n])$.

Let $E := \square$,

$E_i := G'[P_1 \dots P_{i-1}, \square, P_{i+1} \dots P_m]$,

$E_j := G''[P_{m+1} \dots P_{j-1}, \square, P_{j+1} \dots P_n]$.

By reflexivity.

- Case $\nu x.(G'[P_1 \dots P_j \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_n])$.

As above.

- Case $\nu y.(G'[P_1 \dots P_i \dots P_j \dots P_m] \mid G''[P_{m+1} \dots P_n])$.
Case $\nu y.(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_j \dots P_n])$.

We obtain E , E_1 and E_2 from the induction hypothesis and theorem 2.11, and then prepend either $\nu y.(\Box \mid G''[P_{m+1} \dots P_n])$ or $\nu y.(G'[P_1 \dots P_m] \mid \Box)$ to E . The desired equality follows by congruence.

The case for $\Box$ is excluded because $n > 1$. The cases in which P_i and P_j are on the *same* side of the cut, but the cut binds x , and the cases in which P_i and P_j are on different sides of the cut, but the cut binds some other channel y , are excluded by the type system. ■

Finally, we are ready to prove progress. In proofs throughout this dissertation, we will leave uses of $(\gamma \equiv)$ and the congruence rules for reductions implicit.

Theorem 3.14 (Progress)

If $P \vdash \Gamma$, then either P is in canonical form, or there exists a P' such that $P \Rightarrow P'$. □

Proof. By induction on the structure of derivation for $P \vdash \Gamma$. The only interesting case is when the last rule of the derivation is CUT—in every other case, the typing rule constructs a term in which is in canonical form.

We consider the maximum evaluation prefix G of P , such that $P = G[P_1 \dots P_{n+1}]$ and each P_i is an action. The prefix G consists of n cuts, and introduces n channels, but composes $n + 1$ actions. Therefore, one of the following must be true:

- One of the processes is a link $x \leftrightarrow y$ acting on a bound channel. We have:

$$\begin{aligned} G[P_1 \dots x \leftrightarrow y \dots P_{n+1}] &\equiv \text{by lemma 3.12} \\ E[\nu z.(E'[x \leftrightarrow y] \mid Q)] &\equiv \text{by lemma 3.10} \\ E[E'[\nu z.(x \leftrightarrow y \mid Q)]] & \end{aligned}$$

Where $z = x$ or $z = y$. We then apply $(\leftrightarrow_1)$.

- Two of the processes, P_i and P_j , act on the same bound channel x . We have:

$$\begin{aligned} G[P_1 \dots P_i \dots P_j \dots P_{n+1}] &= \text{by lemma 3.13} \\ G[\nu x.(E_i[P_i] \mid E_j[P_j])] &\equiv \text{by lemma 3.10} \\ G[E_i[\nu x.(P_i \mid E_j[P_j])]] &\equiv \text{by lemma 3.10} \\ G[E_i[E_j[\nu x.(P_i \mid P_j)]]] & \end{aligned}$$

We then apply one of the β -reduction rules.

- Otherwise (at least) one of the processes acts on an external channel.
No process P_i is a link. No two processes P_i and P_j act on the same channel x . Therefore, P is canonical. ■

The proof of progress described in this section is novel, though it takes inspiration from the reduction system for CP described by Lindley and Morris [9]. The proof itself is somewhat involved. The reason for this is that we want our reduction strategy to match that of the π -calculus as closely as possible. It is also for this reason that our proof of progress involves some non-determinism. For instance, in the second case of the proof, we do not specify how to select the two processes P_i and P_j if there are multiple options available.

3.4 Rewriting versus commuting

Let's have a look at the differences between the reduction strategy we have defined in this chapter, and the reduction strategy which Wadler [14] defines. Let's imagine the following scenario:

Alice, John, and Mary went on a lovely trip together. However, John and Mary can be a bit scatterbrained sometimes, and it just so happens that they both forgot to bring their wallets. They both owe Alice some money, which they're now trying to pay back at the same time.

We can model this interaction in CP as the following term, assuming , , and  are three processes representing Alice, John and Mary, and  and  are two processes representing John and Mary's respective banks. To simplify notation, we will adopt an “anti-Barendregt” convention in our examples. This means that if two channel names will be identified after a reduction, we will preemptively give them the same name, for instance, z and w in the example below.

$$\nu x.(\nu y.(x(z).y(w)).\text{woman} \mid y[w].(\text{bank} \mid \text{man})) \mid x[z].(\text{bank} \mid \text{woman}))$$

In the above interaction, no β -reduction rule applies *immediately*. This means that using the reduction strategy described by Wadler [14], we apply a commuting

conversion.

$$\begin{aligned}
& \nu x.(\nu y.(x(z).y(w). \text{👩} \mid y[w].(\text{🏛️} \mid \text{👨}))) \mid x[z].(\text{🏛️} \mid \text{👩})) \implies \text{by } (\kappa\mathfrak{A}) \\
& \nu x.(x(z).\nu y.(y(w). \text{👩} \mid y[w].(\text{🏛️} \mid \text{👨}))) \mid x[z].(\text{🏛️} \mid \text{👩})) \implies \text{by } (\beta \otimes \mathfrak{A}) \\
& \nu z.(\nu x.(\nu y.(y(w). \text{👩} \mid y[w].(\text{🏛️} \mid \text{👨}))) \mid \text{👩} \mid \text{🏛️})) \implies \text{by } (\beta \otimes \mathfrak{A}) \\
& \nu z.(\nu x.(\nu w.(\nu y.(\text{👩} \mid \text{👨} \mid \text{🏛️})) \mid \text{👩} \mid \text{🏛️}))
\end{aligned}$$

On the other hand, using the reduction strategy described in this chapter, we will rewrite using structural congruence.

$$\begin{aligned}
& \nu x.(\nu y.(x(z).y(w). \text{👩} \mid y[w].(\text{🏛️} \mid \text{👨}))) \mid x[z].(\text{🏛️} \mid \text{👩})) \equiv \text{by lemma 3.10} \\
& \nu y.(\nu x.(x(z).y(w). \text{👩} \mid x[z].(\text{🏛️} \mid \text{👩}))) \mid y[w].(\text{🏛️} \mid \text{👨})) \implies \text{by } (\beta \otimes \mathfrak{A}) \\
& \nu y.(\nu z.(\nu x.(y(w). \text{👩} \mid \text{👩} \mid \text{🏛️})) \mid y[w].(\text{🏛️} \mid \text{👨})) \equiv \text{by lemma 3.10} \\
& \nu z.(\nu x.(\nu y.(y(w). \text{👩} \mid y[w].(\text{🏛️} \mid \text{👨}))) \mid \text{👩} \mid \text{🏛️})) \implies \text{by } (\beta \otimes \mathfrak{A}) \\
& \nu z.(\nu x.(\nu w.(\nu y.(\text{👩} \mid \text{👨} \mid \text{🏛️})) \mid \text{👩} \mid \text{🏛️}))
\end{aligned}$$

Note that the second reduction sequence is a *valid* reduction sequence in the either reduction system—it is simply not the sequence *chosen* by the reduction strategy described by Wadler [14]. In fact, the structural congruence is only used in a *single* instance in the original reduction strategy for CP—to derive the flipped version of $(\gamma\nu)$.

Chapter 4

Non-deterministic Classical Processes

In this section, we will discuss our main contribution: an extension of CP which allows for races while still excluding deadlocks. We have seen in section 2.1.6 how CP excludes deadlocks, but how exactly does CP exclude races? Let us return to our first example from chapter 1, to the interaction between John, Mary and the store.

$$\begin{array}{c}
 (\bar{x}\langle \text{🍰} \rangle . \bar{x}\langle \text{👏} \rangle . \text{🏪} \mid x\langle y \rangle . \text{👤} \mid x\langle z \rangle . \text{👩}) \\
 \Downarrow_{\star} \\
 (\text{🏪} \mid \text{👤}\{\text{🍰}/y\} \mid \text{👩}\{\text{👏}/z\}) \quad \text{or} \quad (\text{🏪} \mid \text{👤}\{\text{👏}/y\} \mid \text{👩}\{\text{🍰}/z\})
 \end{array}$$

Races occur when more than two processes attempt to communicate simultaneously over the *same* channel. However, the Cut rule of CP requires that *exactly two* processes communicate over each channel:

$$\frac{P \vdash \Gamma, x : A \quad Q \vdash \Delta, x : A^\perp}{\nu x. (P \mid Q) \vdash \Gamma, \Delta} \text{CUT}$$

We could attempt write down a protocol for our example, stating that the store has a pair of channels $x, y : \text{🍰}$ with which it communicates with John and Mary, taking 🍰 to be the type of interactions in which cake *may* be obtained, i.e. of both 🍰 and 👏 , and state that the store communicates with John *and* Mary over a channel of type $\text{🍰} \wp \text{🍰}$. However, this *only* models interactions such as the following:

$$\frac{\frac{\text{John} \vdash \Gamma, x: \text{cake}^\perp \quad \text{Mary} \vdash \Delta, y: \text{cake}^\perp}{y[x].(\text{John} \mid \text{Mary}) \vdash \Gamma, \Delta, y: \text{cake}^\perp \otimes \text{cake}^\perp} \otimes \quad \frac{\text{Store} \vdash \Theta, x: \text{cake}, y: \text{cake}}{y(x).\text{Store} \vdash \Theta, y: \text{cake} \wp \text{cake}} \wp}{\nu y.(y[x].(\text{John} \mid \text{Mary}) \mid y(x).\text{Store}) \vdash \Gamma, \Delta, \Theta} \text{CUT}$$

Note that in this interaction, John will get whatever the store decides to send on x , and Mary will get whatever the store decides to send on y . This means that this interactions gives the choice of who receives what *to the store*. This is not an accurate model of our original example, where the choice of who receives the cake is non-deterministic and depends on factors outside of any of the participants' control! And to make matters worse, the term which models our example is entirely different from the one we initially wrote down in the π -calculus!

The ability to model racy behaviour, such as that in our example, is essential to describing the interactions that take place in realistic concurrent systems. Therefore, we would like to extend CP to allow such races. Specifically, we would like to do it in a way which mirrors the way in which the π -calculus handles non-determinism. We will base our extension on RCP, a subset of CP which we introduced in chapter 2. We have chosen to do this to keep our discussion as simple as possible. Furthermore, as compatibility with the π -calculus is of interest, we will use the reduction system without commuting conversions, which we introduced in chapter 3.

This chapter proceeds as follows. In section 4.1, we introduce the extensions to the terms, structural congruence and types of CP. In section 4.2, we introduce the typing rules for CP_{ND} . In section 4.3, we introduce the reduction rules for CP_{ND} . In section 4.4, we prove that our extension preserves the meta-theoretical of CP. Finally, in section 4.5, we discuss the relation between non-determinism in CP_{ND} and the non-determinism introduced by the addition of non-deterministic local choice.

4.1 Terms and types

Let us return, briefly, to our example.

$$(\bar{x}\langle \text{cake} \rangle.\bar{x}\langle \text{clap} \rangle.\text{Store} \mid x\langle y \rangle.\text{John} \mid x\langle z \rangle.\text{Mary})$$

In this interaction, we see that the channel x is used only as a way to connect the various clients, John and Mary, to the store. The *real* communication, sending

the slice of cake and disappointment, takes places on the channels , , y and z .

Inspired by this, we add two new constructs to the term language of CP: sending and receiving on a *shared* channel. These actions are marked with a $\star$ in order to distinguish them syntactically from ordinary sending and receiving. To group clients, we add another form of parallel composition, which we refer to as *pooling*.

Definition 4.1 (Terms)

We extend definition 2.1 with the following constructs:

$$\begin{aligned}
 P, Q, R &:= \dots \\
 &| \star x[y].P \quad \text{client creation} \\
 &| \star x(y).P \quad \text{server interaction} \\
 &| (P \mid Q) \quad \text{parallel composition of clients}
 \end{aligned}$$

As before, round brackets denote input, square brackets denote output. Note that $\star x[y].P$, much like $x[y].(P \mid Q)$, is a bound output—this means that both client creation and server interaction bind a new name.

In RCP, we terms are identified up to the commutativity and associativity of parallel composition. In CP_{ND} , we add another form of parallel composition, and therefore must extend our structural congruence:

Definition 4.2 (Structural congruence)

We extend definition 2.2 with the following equivalences:

$$\begin{aligned}
 (|-comm) \quad (P \mid Q) &\equiv (Q \mid P) \\
 (|-assoc_1) \quad (P \mid (Q \mid R)) &\equiv ((P \mid Q) \mid R) \\
 (|-extrusion_1) \quad \nu x.((P \mid Q) \mid R) &\equiv (P \mid \nu x.(Q \mid R)) \quad \text{if } x \notin P \\
 (|-extrusion_2) \quad (P \mid \nu x.(Q \mid R)) &\equiv \nu x.((P \mid Q) \mid R)
 \end{aligned}$$

We add axioms for the commutativity and associativity of pooling. We do not add an axiom for $(|-assoc_2)$, as it follows from definition 4.2, see lemma 4.3. It should be noted that $\nu x.(P \mid Q)$ is considered a single, *atomic* construct. Therefore you *cannot* use $(|-assoc_1)$ to rewrite $\nu x.(P \mid (Q \mid R))$ to $\nu x.((P \mid Q) \mid R)$. We do, however, add two axioms which relate cuts and pool. We call these *extrusion*, because they closely resemble the π -calculus axiom for scope extrusion. We add both $(|-extrusion_1)$ and $(|-extrusion_2)$, as these relate two different constructs, and therefore we cannot use the one to derive the other.

Lemma 4.3 ($|-assoc_2$)

We have $((P \mid Q) \mid R) \equiv (P \mid (Q \mid R))$. □

Proof.

$$((P \mid Q) \mid R) \equiv \text{by } (|-comm)$$

$$((Q \mid P) \mid R) \equiv \text{by } (|-comm)$$

$$(R \mid (Q \mid P)) \equiv \text{by } (|-assoc_1)$$

$$((R \mid Q) \mid P) \equiv \text{by } (|-comm)$$

$$(P \mid (R \mid Q)) \equiv \text{by } (|-comm)$$

$$(P \mid (Q \mid R))$$
■

Furthermore, the extensions to structural congruence preserve symmetry.

Theorem 4.4 (Symmetry)

If $P \equiv Q$, then $Q \equiv P$. □

Proof. By induction on the structure of the equivalence proof. ■

We can make another observation from our examples. In every example in which a server interacts with a pool of clients, and which does not deadlock, there are *exactly* as many clients as there are server interactions. Therefore, we add two new *dual* types for client pools and servers, which track how many clients or server interactions they represent.

Definition 4.5 (Types)

We extend definition 2.5 with the following types:

$$\begin{aligned} A, B, C &:= \dots \\ &| \textcolor{blue}{!}_n A \quad \text{pool of } n \text{ clients} \\ &| \textcolor{blue}{?}_n A \quad n \text{ server interactions} \end{aligned}$$

Definition 4.6 (Duality)

We extend definition 2.6 with the following cases:

$$(\textcolor{blue}{!}_n A)^\perp = \textcolor{blue}{?}_n A^\perp \quad (\textcolor{blue}{?}_n A)^\perp = \textcolor{blue}{!}_n A^\perp$$

With these new types, duality remains an involutive function.

Lemma 4.7 (Duality is involutive) We have $A^{\perp\perp} = A$. □

Proof. By induction on the structure of the type A . ■

4.2 Typing clients and servers

We have to add typing rules to associate our new client and server interactions with their types. The definition for environments will remain unchanged, but we will extend the definition for the typing judgement. To determine the new typing rules, we essentially have to answer the question “What typing constructs do we need to complete the following proof?”

$$\begin{array}{c} \text{👤} \vdash \Gamma, y : \text{🍰}^\perp \quad \text{👩} \vdash \Delta, y' : \text{🍰}^\perp \quad \text{🏠} \vdash \Theta, z : \text{🍰}, z' : \text{🍰} \\ \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\ \nu x. ((\star x[y].\text{👤} \mid \star x[y'].\text{👩}) \mid \star x(z).\star x(z').\text{🏠}) \vdash \Gamma, \Delta, \Theta \end{array}$$

Ideally, we would still like the composition of the client pool and the server to be a cut. This seems reasonable, as the left-hand side of the term above has 2 clients, and the right-hand side has two server interactions, so x is used at type $!_2 \text{🍰}^\perp$ on the left, and as $?_2 \text{🍰}$ on the right.

$$\frac{\begin{array}{c} \text{👤} \vdash \Gamma, y : \text{🍰}^\perp \quad \text{👩} \vdash \Delta, y' : \text{🍰}^\perp \quad \text{🏠} \vdash \Theta, z : \text{🍰}, z' : \text{🍰} \\ \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \\ (\star x[y].\text{👤} \mid \star x[y'].\text{👩}) \vdash \Gamma, \Delta, x : !_2 \text{🍰}^\perp \quad \star x(z).\star x(z').\text{🏠} \vdash \Theta, x : ?_2 \text{🍰} \end{array}}{\nu x. ((\star x[y].\text{👤} \mid \star x[y'].\text{👩}) \mid \star x(z).\star x(z').\text{🏠}) \vdash \Gamma, \Delta, \Theta} \text{CUT}$$

We will define the typing judgement, and then discuss servers and clients, the two sides of the above cut, describe the rules we add, and show how they allow us to complete our proof.

Definition 4.8 (Typing judgements)

A typing judgement $P \vdash x_1 : A_1 \dots x_n : A_n$ denotes that the process P communicates along channels $x_1 \dots x_n$ following protocols $A_1 \dots A_n$. Typing judgements can be constructed using the inference rules in figs. 2.1 and 4.1.

4.2.1 Clients and pooling

A client pool represents a number of independent processes, each wanting to interact with the server. Examples of such a pool include John and Mary from our example, customers for online stores in general, and any number of processes which interact with a single, centralised server.

We introduce two new rules: one to construct clients, and one to pool them together. The first rule, $(!_1)$, marks interaction over some channel as a client interaction. It does this by receiving a channel y over a *shared* channel x . The channel y is the channel across which the actual interaction will eventually take place. The second rule, POOL, allows us to pool together clients. This is implemented, as in the π -calculus, using parallel composition.

$$\frac{P \vdash \Gamma, y : A}{\star x[y].P \vdash \Gamma, x : !_1 A} (!_1) \quad \frac{P \vdash \Gamma, x : !_m A \quad Q \vdash \Delta, x : !_n A}{(P \mid Q) \vdash \Gamma, \Delta, x : !_{m+n} A} \text{POOL}$$

Using these rules, we can derive the left-hand side of our proof by marking John and Mary as clients, and pooling them together.

$$\frac{\frac{\text{John} \vdash \Gamma, y : \text{cake}^\perp}{\star x[y].\text{John} \vdash \Gamma, z : !_1 \text{cake}^\perp} (!_1) \quad \frac{\text{Mary} \vdash \Delta, y' : \text{cake}^\perp}{\star x[y'].\text{Mary} \vdash \Delta, y' : !_1 \text{cake}^\perp} (!_1)}{(\star x[y].\text{John} \mid \star x[y'].\text{Mary}) \vdash \Gamma, \Delta, x : !_2 \text{cake}^\perp} \text{POOL}$$

4.2.2 Servers and contraction

Dual to a pool of clients is a server. Our interpretation of a server is a process which offers up some number of interdependent interactions of the same type.

$$\frac{P \vdash \Gamma, y : A}{\star x[y].P \vdash \Gamma, x : !_1 A} (!_1) \quad \frac{P \vdash \Gamma, y : A}{\star x(y).P \vdash \Gamma, x : ?_1 A} (?_1)$$

$$\frac{P \vdash \Gamma, x : !_m A \quad Q \vdash \Delta, x : !_n A}{(P \mid Q) \vdash \Gamma, \Delta, x : !_{m+n} A} \text{POOL}$$

$$\frac{P \vdash \Gamma, x : ?_m A, y : ?_n A}{P\{x/y\} \vdash \Gamma, x : ?_{m+n} A} \text{CONT}$$

Figure 4.1: Typing judgement for CP_{ND} extending that of Figure 2.1

Examples include the store from our example, which gives out slices of cake and disappointment, online stores in general, and any central server which interacts with some number of client processes.

We introduce two new rules to construct servers. The first rule, $(?_1)$, marks a interaction over some channel as a server interaction. It does this by sending a channel y over a *shared* channel x . The channel y is the channel across which the actual interaction will eventually take place. The second rule, CONT , short for contraction, allows us to contract several server interactions into a single server. This allows us to construct a server which has multiple interactions of the same type, across the same shared channel.¹

$$\frac{P \vdash \Gamma, y : A}{\star x(y).P \vdash \Gamma, x : ?_1 A} (?_1) \quad \frac{P \vdash \Gamma, x : ?_m A, y : ?_n A}{P\{x/y\} \vdash \Gamma, x : ?_{m+n} A} \text{CONT}$$

Using these rules, we can derive the right-hand side of our proof, by marking each of the store's interactions as server interactions, and then contracting them.

$$\frac{\frac{\frac{\frac{\text{store} \vdash \Theta, z : \text{cake}, z' : \text{disappointment}}{\star x'(z').\text{store} \vdash \Theta, z : \text{cake}, x' : ?_1 \text{cake}} (?_1)}{\star x(z).\star x'(z').\text{store} \vdash \Theta, x : ?_1 \text{cake}, x' : ?_1 \text{cake}} (?_1)}{\star x(z).\star x'(z').\text{store} \vdash \Theta, x : ?_2 \text{cake}} \text{CONT}$$

Thus, we complete the typing derivation of our example.

4.3 Running clients and servers

Once we have a client/server interaction, how do we run it? Ideally, we would simply use the reduction rule closest to the one used in the π -calculus.

$$\nu x.(\star x[y].P \mid \star x(z).R) \Longrightarrow \nu y.(P \mid R\{y/z\})$$

However, our case is complicated by the fact that in $\nu x.(P \mid Q)$ the name restriction is an inseparable part of the composition, and therefore has to be part of our reduction rule. Because of this, the above reduction can only apply in the singleton case. If the client pool contains more than one client, such as in the term below,

¹While it ultimately does not matter whether $(?_1)$ and $(!_1)$ are implemented with a send or a receive action, it feels more natural to have the server do the sending. Clients indicate their interest in interacting with the server by connecting to the shared channel, but it is up to the server to decide *when* to interact with each channel.

then there is no way to isolate a single client together with the server, because x occurs in both $\star x[y].P$ and $\star x[z].Q$.

$$\nu x.((\star x[y].P \mid \star x[z].Q) \mid \star x(w).R) \not\Rightarrow$$

Therefore, we add a second reduction rule, which handles communication between a one client in a pool of multiple clients and a server.

$$\nu x.((\star x[y].P \mid Q) \mid \star x(z).R) \Rightarrow \nu x.(Q \mid \nu y.(P \mid R\{y/z\}))$$

Lastly, because we have added another form of parallel composition, we add another congruence rule, to allow for reduction inside client pools.

Definition 4.9 (Term reduction)

We extend definition 3.1 with the following reductions:

$$\begin{aligned} (\beta\star_1) \quad & \nu x.(\star x[y].P \mid \star x(z).R) \Rightarrow \nu y.(P \mid R\{y/z\}) \\ (\beta\star_{n+1}) \quad & \nu x.((\star x[y].P \mid Q) \mid \star x(z).R) \Rightarrow \nu x.(Q \mid \nu y.(P \mid R\{y/z\})) \end{aligned}$$

$$\frac{P \Rightarrow P'}{(P \mid Q) \Rightarrow (P' \mid Q)} (\gamma|)$$

The rules $(\beta\star_1)$ and $(\beta\star_{n+1})$ seem like the elimination rules for a list-like construct. This may come as a surprise, as our client pools are built up like binary trees, and the typing rules for both sides are tree-like, with $(!_1)$ and $(?_1)$ playing the role of leaves, and POOL and CONT merging two trees with m and n leaves into one with $m + n$ leaves. However, the server process imposes a sequential ordering on its interactions, and it is because of this that we have to use list-like elimination rules.

So where does the non-determinism in CP_{ND} come from? Let us say we have a term of the following form:

$$\nu x.((\star x[y_1].P_1 \mid \cdots \mid \star x[y_n].P_n) \mid \star x(y).Q)$$

Because pooling is commutative and associative, we can rewrite this term to bring any client in the pool to the front, before applying $(\beta\star_{n+1})$. Thus, like in the π -calculus, the non-determinism is introduced by the structural congruence.

Does this mean that, for an arbitrary client pool P in $\nu x.(P \mid \star x(z).Q)$, every client in that pool is competing for the server interaction on x ? Not necessarily,

as some portion of the clients can be blocked on an external communication. For instance, in the term below, clients $\star x[y_{n+1}].P_{n+1} \dots \star x[y_m].P_m$ are blocked on a communication on the external channel a .

$$\begin{aligned} & \nu x.((\star x[y_1].P_1 \mid \dots \mid \star x[y_n].P_n) \\ & \quad \mid a().(\star x[y_{n+1}].P_{n+1} \mid \dots \mid \star x[y_m].P_m)) \\ & \quad \mid \star x(y_1) \dots \star x(y_m).Q) \end{aligned}$$

If we reduce this term, then only the clients $\star x[y_1].P_1 \dots \star x[y_n].P_n$ will be assigned server interactions, and we end up with the following canonical form term.

$$\begin{aligned} & \nu x.(a().(\star x[y_{n+1}].P_{n+1} \mid \dots \mid \star x[y_m].P_m) \\ & \quad \mid \star x(y_{n+1}) \dots \star x(y_m).Q) \end{aligned}$$

This matches the reduction behaviour of the π -calculus, and it fits with our notion of computation with processes.

4.4 Properties of CP_{ND}

In this section, we will revisit the proofs for three important properties of RCP, namely preservation, progress, and termination, and show that our extensions preserve these properties.

4.4.1 Preservation

Preservation is the fact that term reduction preserves typing. There are two proofs involved in this. First, we show that structural congruence preserves typing.

Theorem 4.10 (Preservation for $\equiv$)

If $P \vdash \Gamma$ and $P \equiv Q$, then $Q \vdash \Gamma$. □

Proof. By induction on the structure of the equivalence. The cases for reflexivity, transitivity and congruence are trivial. The cases for $(\nu\text{-comm})$ and $(\nu\text{-assoc}_1)$ are given in fig. 2.4. The cases for $(|\text{-comm})$ and $(|\text{-assoc}_1)$ are given in fig. 4.2. ■

Secondly, we prove that term reduction preserves typing.

Theorem 4.11 (Preservation)

If $P \vdash \Gamma$ and $P \Longrightarrow Q$, then $Q \vdash \Gamma$. □

Proof. By induction on the structure of the reduction. See fig. 2.5 for $(\leftrightarrow_1)$, $(\leftrightarrow_2)$, and the β -reduction rules from CP. See fig. 4.3 for the β -reduction rules of CP_{ND} . The cases for $(\gamma\nu)$ and $(\gamma|)$ are trivial by call to the induction hypothesis, and the case for $(\gamma\equiv)$ is trivial by call to the induction hypothesis and theorem 4.10. ■

4.4.2 Canonical forms and progress

In this section, we will extend the definition of canonical forms and the proof of progress progress given in chapter 3.

4.4.2.1 Canonical forms

First, we extend the definitions of actions with our actions for client and server creation.

Definition 4.12 (Action)

We extend definition 3.2 with the following cases:

- $\star x[y].P'$
- $\star x(y).P'$

Secondly, as we can reduce inside client pools, we will add pooling to our definition of evaluation prefixes.

Definition 4.13 (Evaluation prefixes)

We extend definition 3.3 with the following constructs:

$$G, H := \cdots \mid (G \mid H)$$

We also define a special case of evaluation prefixes, which we will refer to as *pooling prefixes*. These are evaluation prefixes which consist solely of pooling operators and holes.

Definition 4.14 (Plugging)

We extend definition 3.4 with the following case:

$$(G \mid H)[R_1 \dots R_m, R_{m+1} \dots R_n] := (G[R_1 \dots R_m] \mid H[R_{m+1} \dots R_n])$$

Note that in this case, G is an evaluation prefix with m holes, and H is an evaluation prefix with $(n - m)$ holes.

The definition for the maximum evaluation prefix is unchanged.

There are some subtleties to our definition of canonical forms. The type system for CP guarantees that all links directly under an evaluation context act on a bound channel. Not so for CP_{ND} .

$$\frac{\overline{x \leftrightarrow y \vdash x : !_m A, y : ?_m A^\perp} \text{ AX} \quad P \vdash \Gamma, x : !_n A}{(x \leftrightarrow y \mid P) \vdash \Gamma, x : !_{m+n} A, y : ?_m A^\perp} \text{ POOL}$$

There is no way to sensibly reduce this link. Furthermore, in CP, if two processes act on the same channel, then they must be on different sides of the cut introducing that channel. The addition of shared channels and client pools invalidates this property. Therefore, we will have to be more careful about the way we define canonical forms. We restate the definition of canonical forms below. The additions have been italicised.

Definition 4.15 (Canonical forms)

A process P is in canonical form if it is an action, or if it is of the form $G[P_1 \dots P_n]$, where G is the maximum evaluation prefix of P , no P_i is a link *which acts on a bound channel*, and no P_i and P_j , *on different sides of at least one cut*, act on the same channel.

4.4.3 Evaluation contexts

Evaluation contexts are one-holed term contexts under which reduction can take place. Since we have added another congruence rule, stating that reduction can take place inside client pools, we extend our definition of evaluation contexts to match this.

Definition 4.16 (Evaluation contexts)

We extend definition 3.8 with the following constructs:

$$E := \dots \mid (E \mid P) \mid (P \mid E)$$

We also define a special case of evaluation contexts, which we will refer to as *pooling contexts*. These are evaluation contexts which consist solely of pooling operators and holes.

Definition 4.17 (Plugging)

We extend definition 3.9 with the following cases:

$$\begin{aligned} (E \mid P)[R] &:= (E[R] \mid P) \\ (P \mid E)[R] &:= (P \mid E[R]) \end{aligned}$$

We also restate lemma 3.10, and prove that our extension preserves the property.

Lemma 4.18

If $\nu x.(E[P] \mid Q) \vdash \Gamma$ and $x \notin E$, then $\nu x.(E[P] \mid Q) \equiv E[\nu x.(P \mid Q)]$. $\square$

Proof. By induction on the structure of the evaluation context E .

- Case $\square$, $\nu y.(H \mid R)$, and $\nu y.(R \mid H)$. See lemma 3.10.
- Case $(E \mid R)$.

$$\begin{aligned} \nu x.((E[P] \mid R) \mid Q) &\equiv \text{by } (|-comm) \\ \nu x.((R \mid E[P]) \mid Q) &\equiv \text{by } (|-extrusion_1) \\ (R \mid \nu x.(E[P] \mid Q)) &\equiv \text{by } (|-comm) \\ (\nu x.(E[P] \mid Q) \mid R) &\equiv \text{by the induction hypothesis} \\ (E[\nu x.(P \mid Q)] \mid R) & \end{aligned}$$

- Case $(R \mid E)$.

$$\begin{aligned} \nu x.((R \mid E[P]) \mid Q) &\equiv \text{by } (|-extrusion_1) \\ (R \mid \nu x.(E[P] \mid Q)) &\equiv \text{by the induction hypothesis} \\ (R \mid E[\nu x.(P \mid Q)]) & \end{aligned}$$

In each case, the side condition for $(|-extrusion_1)$, $x \notin R$, can be inferred from $x \notin E$, and the side conditions for the induction hypothesis can be inferred from theorem 4.10 and $x \notin E$. $\blacksquare$

Furthermore, it will be useful to prove a similar lemma, which shows that we can push any pooling downwards under an evaluation context.

Lemma 4.19

If $(E[P] \mid Q) \vdash \Gamma$, then $(E[P] \mid Q) \equiv E[(P \mid Q)]$. $\square$

Proof. By induction on the structure of the evaluation context E .

- Case $\square$. By reflexivity.
- Case $\nu y.(E \mid R)$.

$$\begin{aligned}
 (\nu y.(E[P] \mid R) \mid Q) &\equiv \text{by } (|-comm) \\
 (Q \mid \nu y.(E[P] \mid R)) &\equiv \text{by } (|-extrusion_2) \\
 \nu y.((Q \mid E[P]) \mid R) &\equiv \text{by } (\nu-comm) \\
 \nu y.((E[P] \mid Q) \mid R) &\equiv \text{by the induction hypothesis} \\
 \nu y.(E[(P \mid Q)] \mid R) &
 \end{aligned}$$

- Case $\nu y.(R \mid E)$.

$$\begin{aligned}
 (\nu y.(R \mid E[P]) \mid Q) &\equiv \text{by } (|-comm) \\
 (Q \mid \nu y.(E[P] \mid R)) &\equiv \text{by } (|-extrusion_2) \\
 \nu y.((Q \mid E[P]) \mid R) &\equiv \text{by } (\nu-comm) \\
 \nu y.(R \mid (Q \mid E[P])) &\equiv \text{by the induction hypothesis} \\
 \nu y.(R \mid E[(P \mid Q)]) &
 \end{aligned}$$

- Case $(E \mid R)$.

$$\begin{aligned}
 ((E[P] \mid R) \mid Q) &\equiv \text{by } (|-comm) \\
 ((R \mid E[P]) \mid Q) &\equiv \text{by } (|-assoc_2) \\
 (R \mid (E[P] \mid Q)) &\equiv \text{by } (|-comm) \\
 ((E[P] \mid Q) \mid R) &\equiv \text{by the induction hypothesis} \\
 (E[(P \mid Q)] \mid R) &
 \end{aligned}$$

- Case $(R \mid E)$.

$$\begin{aligned}
 ((R \mid E[P]) \mid Q) &\equiv \text{by } (|-assoc_2) \\
 (R \mid (E[P] \mid Q)) &\equiv \text{by the induction hypothesis} \\
 (R \mid E[(P \mid Q)]) &
 \end{aligned}$$

In each case, the side conditions for $(|-extrusion_2)$, $y \notin Q$, can be inferred from the fact that $(Q \mid \nu y.(E[P] \mid R))$ is well-typed; the side conditions for $(\nu-assoc_2)$, $x \notin R$ and $y \notin Q$, can be inferred from $x \notin E$ and the fact that $(E[P] \mid Q)$ is well-typed; and the side conditions for the induction hypothesis can be inferred from the fact that $(E[P] \mid Q)$ is well-typed, theorem 4.10 and $x \notin E$. ■

4.4.4 Progress

Progress is the fact that every term is either in some canonical form, or can be reduced further. First, we will restate lemma 3.12 and lemma 3.13, which relate evaluation prefixes and evaluation contexts, and show that our extension preserves these properties.

Lemma 4.20

If $G[P_1 \dots P_n] \vdash \Gamma$, and some P_i is a link $x \leftrightarrow y$, then either x and y are not bound by G , or there exist E , E' and Q such that $G[P_1 \dots P_n] \equiv E[\nu x.(E'[x \leftrightarrow y] \mid Q)]$. $\square$

Proof. As lemma 3.12. The two cases for $(G'[P_1 \dots P_i \dots P_m] \mid G''[P_{m+1} \dots P_n])$ and $(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_n])$ are handled exactly as the cases for cuts which do not bind x or y . $\blacksquare$

Lemma 4.21

If $G[P_1 \dots P_n] \vdash \Gamma$, and some P_i and P_j , on different sides of at least one cut, act on the same bound channel x , then there exist E , E_i and E_j such that $G[P_1 \dots P_n] = E[\nu x.(E_i[P_i] \mid E_j[P_j])]$. $\square$

Proof. As lemma 3.12. The two cases for $(G'[P_1 \dots P_i \dots P_j \dots P_m] \mid G''[P_{m+1} \dots P_n])$ and $(G'[P_1 \dots P_m] \mid G''[P_{m+1} \dots P_i \dots P_j \dots P_n])$ are handled exactly as the cases for cuts which do not bind x . The case where P_i and P_j are on different sides of a pool is excluded by the type system. $\blacksquare$

In essence, lemma 4.20 and lemma 4.21 cover the cases in which either $(\leftrightarrow_1)$ or a β -reduction rule will be applied. However, after applying lemma 4.21, we cannot immediately apply $(\beta\star_{n+1})$. For that, we must uncover at least one layer of pooling. We prove a lemma which states that if we have an interaction on a shared channel x , we can push all pooling rules which pool clients communicating on x inwards.

Lemma 4.22

If $G[P] \vdash \Gamma, x : !_n A$ and $x \in P$, then there exists an E and $R_1 \dots R_{n-1}$ such that $G[P] \equiv E[(P \mid (R_1 \mid (\dots \mid R_{n-1}) \dots))]$, where $x \notin E$ and $x \in R_1, \dots, x \in R_{n-1}$. $\square$

Proof. By induction on the structure of the evaluation context G .

- Case $\Box$. By reflexivity.
- Case $\nu y.(G \mid R)$.
Case $\nu y.(R \mid G)$.
By the induction hypothesis.
- Case $(G \mid R)$. There are two cases:
 - Case $x \in R$.

$$\begin{aligned}
 & (G[P] \mid R_{n-1}) \\
 & \equiv \quad \text{by the induction hypothesis} \\
 & (E[(P \mid (R_1 \mid (\dots \mid R_{n-2}) \dots))] \mid R_{n-1}) \\
 & \equiv \quad \text{by lemma 4.19} \\
 & E[(P \mid (R_1 \mid (\dots \mid (R_{n-2} \mid R_{n-1})) \dots))]
 \end{aligned}$$

- Case $x \notin R$. By the induction hypothesis.

- Case $(R \mid G)$. There are two cases:
 - Case $x \in R$.

$$\begin{aligned}
 & (R_{n-1} \mid G[P]) \\
 & \equiv \quad \text{by } (|-comm) \\
 & (G[P] \mid R_{n-1}) \\
 & \equiv \quad \text{by the induction hypothesis} \\
 & (E[(P \mid (R_1 \mid (\dots \mid R_{n-2}) \dots))] \mid R_{n-1}) \\
 & \equiv \quad \text{by lemma 4.19} \\
 & E[(P \mid (R_1 \mid (\dots \mid (R_{n-2} \mid R_{n-1})) \dots))]
 \end{aligned}$$

- Case $x \notin R$. By the induction hypothesis. ■

Finally, we are ready to extend our proof of progress. The overall structure of the proof remains the same, though the addition of pooling makes the wording slightly more subtle.

Theorem 4.23 (Progress)

If $P \vdash \Gamma$, then either P is in canonical form, or there exists a P' such that $P \Rightarrow P'$. □

Proof. By induction on the structure of derivation for $P \vdash \Gamma$. The only interesting cases are when the last rule of the derivation is CUT or POOL. In

every other case, the typing rule constructs a term in which is in canonical form.

If the last rule in the derivation is CUT or POOL, we consider the maximum evaluation prefix G of P , such that $P = G[P_1 \dots P_{m+n+1}]$ and each P_i is an action. The prefix G consists of m pools, n cuts, and introduces n channels, but composes $m + n + 1$ actions, at most $m + 1$ of which are on the same side of all cuts. Therefore, one of the following must be true:

- One of the processes is a link $x \leftrightarrow y$ acting on a bound channel.
We proceed as in theorem 3.14, replacing lemma 3.10 and lemma 3.12 with lemma 4.18 and lemma 4.20.
- Two of the processes, P_i and P_j , on different sides of at least one cut, act on the same bound channel x . We have:

$$\begin{aligned} G[P_1 \dots P_i \dots P_j \dots P_{m+n+1}] &\equiv \text{by lemma 4.21} \\ E[\nu x.(E_i[P_i] \mid E_j[P_j])] \end{aligned}$$

There are two cases:

- If x is a shared channel, we have $x : !_n A$ with $n > 1$ in either $E_i[P_i]$ or $E_j[P_j]$. Assume the former. We can infer $x \notin E_j$. We have:

$$\begin{aligned} E[\nu x.(E_i[P_i] \mid E_j[P_j])] &\equiv \text{by lemma 4.19} \\ E[E_j[\nu x.(E_i[P_i] \mid P_j)]] &\equiv \text{by lemma 4.22} \\ E[E_j[E'_i[\nu x.((P_i \mid (R_1 \mid (\dots \mid R_{n-1}) \dots)) \mid P_j)]]] \end{aligned}$$

We apply $(\beta\star_{n+1})$. Similarly if $x : !_n A$ in $E_j[P_j]$.

- Otherwise, we can infer $y \notin E_i$ and $y \notin E_j$.

We proceed as in theorem 3.14, including $(\beta\star_1)$ in the β -reduction rules, and replacing lemma 3.10 and lemma 3.13 with lemma 4.18 and lemma 4.21.

- Otherwise (at least) one of the actions acts on a free variable.
No process P_i is a link acting on a bound channel. No two processes P_i and P_j act on the same channel x . Therefore, P is canonical. ■

4.4.5 Termination

Termination is the fact that if we iteratively apply progress to obtain a reduction, and apply that reduction, we will eventually end up with a term in canonical form. We restate its proof here for the sake of completeness, but its wording is unchanged, modulo references to figures.

Theorem 4.24 (Termination)

If $P \vdash \Gamma$, then there are no infinite $\Rightarrow$ reduction sequences. $\square$

Proof. Every reduction reduces a single cut to zero, one or two cuts. However, each of these cuts is *smaller*, in the sense that the type of the channel on which the communication takes place is smaller. Each reduction either eliminates a connective, or decreases a resource index on the type of a shared channel. See figs. 2.5 and 4.3. Furthermore, each instance of the structural congruence preserves the size of the cut—see figs. 2.4 and 4.2. Therefore, there cannot be an infinite $\Rightarrow$ reduction sequence. $\blacksquare$

4.5 CP_{ND} and non-deterministic local choice

In section 2.2, we discussed the non-deterministic local choice operator, which is used in several extensions of π DILL and CP [1, 3, 4]. This operator is admissible in CP_{ND} . We can derive the non-deterministic choice $P + Q$ by constructing the following term:

$$\begin{aligned} & \nu x. ((\star x[y].y[\text{inr}].y[] .0 \\ & \quad | \star x[z].z[\text{inr}].z[] .0) \\ & \quad | \star x(y).\star x(z).\text{case } y \{ y().\text{case } y \{ y().P; y().P \} \\ & \quad \quad ; y().\text{case } y \{ y().Q; y().Q \} \}) \end{aligned}$$

The term is a cut between two processes. Both sides are well-typed, see fig. 4.4. Let us unpack what each side is doing. On the left-hand side, we have a pool of two processes, $\star x[y].y[\text{inr}].y[] .0$ and $\star x[z].z[\text{inr}].z[] .0$. Each makes a choice—the first sends **inl**, the second sends **inr**. On the right-hand side, we have a server with both P and Q . This server has two channels on which a choice is offered, y and z . However, the choice on z does not affect the outcome of the process. When these clients and the server are put together, the choices offered by the server will be

non-deterministically lined up with the clients which make choices, and either P or Q will run.

While there is a certain amount of overhead involved in this encoding, it scales linearly in terms of the number of processes. The reverse—encoding the non-determinism present in CP_{ND} using non-deterministic local choice—scales exponentially, as with the π -calculus.

Nonetheless, it is worrying that we duplicate each program P and Q in order to encode non-deterministic local choice. However, we can replace the each term of the form $\text{case } y \{y().P; y().P\}$ with $\nu w.(\text{case } z \{z().w[].0; z().w[].0\} \mid w().P)$. This process is also well-typed, see below.

$$\frac{
 \frac{
 \frac{}{w[] \vdash w : \mathbf{1}}{(\mathbf{1})}
 }{z().w[] \vdash w : \mathbf{1}, z : \perp} (\perp)
 \quad
 \frac{
 \frac{}{w[] \vdash w : \mathbf{1}}{(\mathbf{1})}
 }{z().w[] \vdash w : \mathbf{1}, z : \perp} (\perp)
 }{
 \text{case } z \{z().w[] \vdash w : \mathbf{1}, z : \perp \& \perp\}
 } (\&)
 \quad
 \frac{
 P \vdash \Gamma
 }{w().P \vdash \Gamma, w : \perp} (\perp)
 }{
 \nu w.(\text{case } z \{z().w[] \vdash w : \mathbf{1}, z : \perp \& \perp\} \mid w().P) \vdash \Gamma, z : \perp \& \perp
 } \text{CUT}$$

$$\begin{aligned}
& (|-comm) \quad \frac{P \vdash \Gamma, x : !_m A \quad Q \vdash \Delta, x : !_n A}{(P \mid Q) \vdash \Gamma, \Delta, !_{m+n} A} \text{POOL} \\
& \equiv \quad \frac{Q \vdash \Delta, x : !_n A \quad P \vdash \Gamma, x : !_m A}{(Q \mid P) \vdash \Gamma, \Delta, x : !_{m+n} A} \text{POOL} \\
& (|-assoc_1) \quad \frac{P \vdash \Gamma, x : !_l A \quad \frac{Q \vdash \Delta, x : !_m A \quad R \vdash \Theta, x : !_n A}{(Q \mid R) \vdash \Delta, \Theta, x : !_{m+n} A} \text{POOL}}{(P \mid (Q \mid R)) \vdash \Gamma, \Delta, \Theta, x : !_{l+m+n} A} \text{POOL} \\
& \equiv \quad \frac{\frac{P \vdash \Gamma, x : !_l A \quad Q \vdash \Delta, x : !_m A}{(P \mid Q) \vdash \Gamma, \Delta, x : !_{l+m} A} \text{POOL} \quad R \vdash \Theta, x : !_n A}{((P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta, x : !_{l+m+n} A} \text{POOL} \\
& \text{or} \\
& (|-assoc_1) \quad \frac{P \vdash \Gamma, x : !_k A \quad \frac{Q \vdash \Delta, x : !_l A, y : !_m B \quad R \vdash \Theta, x : !_n B}{(Q \mid R) \vdash \Delta, \Theta, x : !_l A, y : !_{m+n} B} \text{POOL}}{(P \mid (Q \mid R)) \vdash \Gamma, \Delta, \Theta, x : !_{k+l} A, y : !_{m+n} B} \text{POOL} \\
& \equiv \quad \frac{\frac{P \vdash \Gamma, x : !_k A \quad Q \vdash \Delta, x : !_l A, y : !_m B}{(Q \mid R) \vdash \Gamma, \Delta, x : !_{k+l} A, y : !_m B} \text{POOL} \quad R \vdash \Theta, x : !_n B}{(P \mid (Q \mid R)) \vdash \Gamma, \Delta, \Theta, x : !_{k+l} A, y : !_{m+n} B} \text{POOL} \\
& (|-extrusion_1) \quad \frac{P \vdash \Gamma, y : !_m B \quad \frac{Q \vdash \Delta, x : A, y : !_n B \quad R \vdash \Theta, x : A^\perp}{\nu x.(Q \mid R) \vdash \Gamma, \Delta, \Theta, y : !_n B} \text{CUT}}{(P \mid \nu x.(Q \mid R)) \vdash \Gamma, \Delta, \Theta, y : !_{m+n} B} \text{POOL} \\
& \equiv \quad \frac{\frac{P \vdash \Gamma, y : !_m B \quad Q \vdash \Delta, x : A, y : !_n B}{(P \mid Q) \vdash \Gamma, \Delta, y : !_{m+n} B} \text{POOL} \quad R \vdash \Theta, x : A^\perp}{\nu x.((P \mid Q) \mid R) \vdash \Gamma, \Delta, \Theta, y : !_{m+n} B} \text{CUT} \\
& (|-extrusion_2) \quad (\text{as above})
\end{aligned}$$

Figure 4.2: Type preservation for the structural congruence of CP_{ND}

$$\begin{aligned}
(\beta\star_1) \quad & \frac{\frac{P \vdash \Gamma, y:A}{\star x[y].P \vdash \Gamma, x:!_1 A} !_1 \quad \frac{Q \vdash \Delta, z:A^\perp}{\star x(z).Q \vdash \Delta, x:?_1 A} ?_1}{\nu x.(\star x[y].P \mid \star x(z).Q) \vdash \Gamma, \Delta} \text{CUT} \\
\Rightarrow \quad & \frac{P \vdash \Gamma, y:A \quad Q \vdash \Delta, z:A^\perp}{\nu y.(P \mid Q\{y/z\}) \vdash \Gamma, \Delta} \text{CUT} \\
(\beta\star_{n+1}) \quad & \frac{\frac{P \vdash \Gamma, y:A}{\star x[y].P \vdash \Gamma, x:!_1 A} !_1 \quad \frac{Q \vdash \Delta, x:!_n A}{(\star x[y].P \mid Q) \vdash \Gamma, \Delta, x:!_{n+1} A} \text{POOL} \quad \frac{\frac{R \vdash \Theta, z:A^\perp, x:?_n A}{\star x'(z).Q \vdash \Theta, x':?_1 A, x:?_n A} ?_1}{\star x(z).Q \vdash \Theta, x:?_{n+1} A} \text{CONT}}{\nu x.((\star x[y].P \mid Q) \mid \star x(z).Q) \vdash \Gamma, \Delta, \Theta} \text{CUT} \\
\Rightarrow \quad & \frac{Q \vdash \Delta, x:!_n A \quad \frac{P \vdash \Gamma, y:A \quad R \vdash \Theta, z:A^\perp, x:?_n A}{\nu y.(P \mid R\{y/z\}) \vdash \Gamma, \Theta, x:?_n A} \text{CUT}}{\nu x.(Q \mid \nu y.(P \mid R\{y/z\})) \vdash \Gamma, \Delta, \Theta} \text{CUT}
\end{aligned}$$

Figure 4.3: Type preservation for the β -reduction rules of CP_{ND}

Chapter 5

Conclusions and future work

We have presented CP_{ND} , an extension of CP [14] which permits non-deterministic communication. We have given proofs for preservation, progress, and termination for the term reduction system of CP_{ND} . We have shown that we can define non-deterministic local choice in CP_{ND} .

We have also presented an alternative reduction system for CP, based on the work by Lindley and Morris [9], which more closely resembles reduction in the π -calculus.

5.1 Mechanisation of CP_{ND}

We have mechanised a variant of CP_{ND} , which has the same terms and types, but has a different reduction system for the non-deterministic terms. While the existence of this mechanisation gives us some confidence in the correctness of the proofs given in this dissertation, and shows that the type system of CP_{ND} corresponds to a sound logical system, it would be worthwhile to extend this mechanisation to cover the work described in this dissertation.

5.2 Relation to bounded linear logic

We mentioned in chapter 1 that CP_{ND} was inspired by bounded linear logic (BLL) [7]. BLL is a typed lambda calculus based on intuitionistic linear logic which guarantees

that its programs are polynomial-time functions. It too uses resource-indexed exponentials. However, instead of interpreting these as client and server interactions, BLL interprets them as accesses to a memory cell, as is a common interpretation in linear logic [6]. There are some superficial differences between BLL and CP_{ND} , e.g. the former is intuitionistic while the latter is classical, but the main difference between the two lies in storage versus pooling. In BLL, $!_n A$ denotes a memory cell which can be accessed n times, whereas in CP_{ND} , $!_n A$ represents a pool of n different values, computed independently by n different processes.

5.3 Name restriction and parallel composition

It would be worthwhile to decouple the name restriction from the parallel composition in CP, as this would greatly simplify our reduction system. We could do this, for instance, using a two-layered environment, which tracks which sets of channels are interdependent.

$$\begin{array}{c}
 X, Y, Z := \Gamma_1; \dots; \Gamma_n \\
 \frac{P \vdash X \quad Q \vdash Y}{P \mid Q \vdash X; Y} \quad \frac{P \vdash x : A, \Gamma; x : A^\perp, \Delta; X}{(\nu x)P \vdash \Gamma, \Delta; X}
 \end{array}$$

This decoupling would allow us to reduce the number of reduction rules for servers and clients, and strengthen our correspondence to the π -calculus.

5.4 Recursion and resource variables

Our formalism so far has only captured servers that provide for a fixed number of clients. More realistically, we would want to define servers that provide for arbitrary numbers of clients. This poses two problems: how would we define arbitrarily-interacting stateful processes, and how would we extend the typing discipline of CP_{ND} to account for them without losing its static guarantees.

One approach to defining server processes would be to combine CP_{ND} with structural recursion and corecursion, following the μCP extension of Lindley and Morris [10]. Their approach can express processes which produce streams of

A channels. Such a process would expose a channel with the corecursive type $\nu X.A \wp (1 \oplus X)$. Given such a process, it is possible to produce a channel of type $A \wp A \wp \dots \wp A$ for any number of A s, allowing us to satisfy the type $?_n A$ for an arbitrary n .

We would also need to extend the typing discipline to capture arbitrary use of shared channels. One approach would be to introduce resource variables and quantification. Following this approach, in addition to having types $?_n A$ and $!_n A$ for concrete n , we would also have types $?_x A$ and $!_x A$ for resource variables x . These variables would be introduced by quantifiers $\forall x A$ and $\exists x A$. Defining terms corresponding to $\forall x A$, and its relationship with structured recursion, presents an interesting area of further work.

5.5 Cuts with leftovers

So far, our account of non-determinism in client/server interactions only allows interactions between equal numbers of clients and server interactions. A natural extension of this would be to investigate if we could define a special case of cut on a client/server interaction, such that e.g. the clients only consume part of the server resources.

$$\frac{P \vdash \Gamma, x : !_n A \quad Q \vdash \Delta, x : ?_m A^\perp \quad n < m}{\nu x.(P \mid Q) \vdash \Gamma, \Delta, x : ?_{m-n} A^\perp}$$

Such an extension would work well together with an extension allowing clients and servers to provide an arbitrary number of interactions.

5.6 Relation to exponentials in CP

Our account of CP has not included the exponentials $?A$ and $!A$. The type $!A$ denotes arbitrarily many independent instances of A , while the type $?A$ denotes a concrete (if unspecified) number of potentially-dependent instances of A . Existing interpretations of linear logic as session types have taken $!A$ to denote A -servers, while $?A$ denotes A -clients. However, the analogy is imperfect: while we expect servers to provide arbitrarily many instances of their behavior, we also expect those instances to be interdependent.

With quantification over resource variables, we can give precise accounts of both CP's exponentials and idealised servers and clients. CP exponentials could be embedded into this framework using the definitions $!A := \forall n!_n A$ and $?A := \exists n?_n A$. We would also have types that precisely matched our intuitions for server and client behavior: an A server is of type $\forall n?_n A$, being unbounded but dependent, while a collection of A clients is of type $\exists n!_n A$, being definitely sized by independent.

Bibliography

- [1] Robert Atkey, Sam Lindley and J. Garrett Morris. ‘Conflation Confers Concurrency’. In: *A List of Successes That Can Change the World: Essays Dedicated to Philip Wadler on the Occasion of His 60th Birthday*. Ed. by Sam Lindley et al. Lecture Notes in Computer Science. 2016. DOI: 10.1007/978-3-319-30936-1_2. URL: http://dx.doi.org/10.1007/978-3-319-30936-1_2.
- [2] Michele Boreale. ‘On the expressiveness of internal mobility in name-passing calculi’. In: *Theoretical Computer Science* 195.2 (Mar. 1998), pp. 205–226. DOI: 10.1016/S0304-3975(97)00220-X. URL: [https://doi.org/10.1016/S0304-3975\(97\)00220-X](https://doi.org/10.1016/S0304-3975(97)00220-X).
- [3] Luís Caires. ‘Types and Logic, Concurrency and Non-Determinism’. In: *Essays for the Luca Cardelli Fest*. Ed. by Martin Abadi et al. Microsoft Research, Sept. 2014. URL: <https://www.microsoft.com/en-us/research/publication/essays-for-the-luca-cardelli-fest/>.
- [4] Luís Caires and Jorge A. Pérez. ‘Linearity, Control Effects, and Behavioral Types’. In: *Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings*. Springer, Apr. 2017.
- [5] Luís Caires and Frank Pfenning. ‘Session Types as Intuitionistic Linear Propositions’. In: *CONCUR 2010 - Concurrency Theory: 21th International Conference, CONCUR 2010, Paris, France, August 31-September 3, 2010. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 222–236. ISBN: 978-3-642-15375-4. DOI: 10.1007/978-3-642-15375-4_16. URL: http://dx.doi.org/10.1007/978-3-642-15375-4_16.

- [6] Jean-Yves Girard. ‘Linear logic’. In: *Theoretical Computer Science* 50.1 (1987), pp. 1–101. DOI: 10.1016/0304-3975(87)90045-4. URL: [http://dx.doi.org/10.1016/0304-3975\(87\)90045-4](http://dx.doi.org/10.1016/0304-3975(87)90045-4).
- [7] Jean-Yves Girard, Andre Scedrov and Philip J. Scott. ‘Bounded Linear Logic: A Modular Approach to Polynomial-time Computability’. In: *Theor. Comput. Sci.* 97.1 (Apr. 1992), pp. 1–66. ISSN: 0304-3975. DOI: 10.1016/0304-3975(92)90386-T. URL: [http://dx.doi.org/10.1016/0304-3975\(92\)90386-T](http://dx.doi.org/10.1016/0304-3975(92)90386-T).
- [8] Kohei Honda. ‘Types for dyadic interaction’. In: *CONCUR’93*. Springer Nature, 1993, pp. 509–523. DOI: 10.1007/3-540-57208-2_35. URL: http://dx.doi.org/10.1007/3-540-57208-2_35.
- [9] Sam Lindley and J. Garrett Morris. ‘A Semantics for Propositions as Sessions’. In: *Programming Languages and Systems*. Springer Nature, 2015, pp. 560–584. DOI: 10.1007/978-3-662-46669-8_23. URL: http://dx.doi.org/10.1007/978-3-662-46669-8_23.
- [10] Sam Lindley and J. Garrett Morris. ‘Talking Bananas: Structural Recursion for Session Types’. In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*. ICFP 2016. Nara, Japan: ACM, 2016, pp. 434–447. ISBN: 978-1-4503-4219-3. DOI: 10.1145/2951913.2951921. URL: <http://doi.acm.org/10.1145/2951913.2951921>.
- [11] Robin Milner. ‘The polyadic π -calculus: a tutorial’. In: *Logic and algebra of specification*. Springer, 1993, pp. 203–246.
- [12] Robin Milner, Joachim Parrow and David Walker. ‘A calculus of mobile processes, II’. In: *Information and Computation* 100.1 (Sept. 1992), pp. 41–77. DOI: 10.1016/0890-5401(92)90009-5. URL: [https://doi.org/10.1016/0890-5401\(92\)90009-5](https://doi.org/10.1016/0890-5401(92)90009-5).
- [13] Davide Sangiorgi. ‘ π -calculus, internal mobility, and agent-passing calculi’. In: *Theoretical Computer Science* 167.1-2 (1996), pp. 235–274. DOI: 10.1016/0304-3975(96)00075-8. URL: [https://doi.org/10.1016/0304-3975\(96\)00075-8](https://doi.org/10.1016/0304-3975(96)00075-8).
- [14] Philip Wadler. ‘Propositions As Sessions’. In: *Proceedings of the 17th ACM SIGPLAN International Conference on Functional Programming*. ICFP ’12. Copenhagen, Denmark: ACM, 2012, pp. 273–286. ISBN: 978-1-4503-1054-3. DOI: 10.1145/2364527.2364568. URL: <http://doi.acm.org/10.1145/2364527.2364568>.