Raymond Hu

Concurrent Combinators for Mobile Processes

MSc Advanced Computing

Imperial College, 2006

Project Originator and Supervisor: Dr Nobuko Yoshida

Contents

1 Introduction **1**
- 1.1 Process Calculi . . . 1
- 1.2 Concurrent Combinators [2] . . . 2
- 1.3 Application of Concurrent Combinators . . . 5

2 The Asynchronous π-Calculus **7**
- 2.1 Syntax . . . 8
- 2.2 Reduction . . . 10
- 2.3 Transition . . . 12
- 2.4 Behavioural Equivalence . . . 15
- 2.5 Expressiveness . . . 22
  - 2.5.1 Polyadicity . . . 23
  - 2.5.2 Synchronised Communication . . . 23
- 2.6 Asynchronous Bisimilarity . . . 24

3 Concurrent Combinators **25**
- 3.1 The **cc** [2] . . . 25
- 3.2 Embedding the **cc** within the π-calculus [2] . . . 27
- 3.3 Encoding the π-Calculus in the $\mathbf{cc}_\pi$[9] . . . 30

4 Related Work **37**
- 4.1 Replication in Concurrent Combinators [3] . . . 37
  - 4.1.1 Generalised Concurrent Combinators . . . 37
  - 4.1.2 Encoding Replication . . . 38
- 4.2 Solos in Concert [4] . . . 38
  - 4.2.1 The Fusion Calculus [7] . . . 39
  - 4.2.2 The Fusion Calculus of Solos [4] . . . 39

5 Conclusions and Further Work **43**
- 5.1 Conclusions . . . 43
- 5.2 Further Work . . . 43

A Details of Proofs from Chapter 3 **45**
- A.1 Proposition 3.2.4 . . . 45
- A.2 Proposition 3.3.1 . . . 51
- A.3 Proposition 3.3.4 . . . 52
- A.4 Lemma 3.3.8 . . . 58

List of Tables **59**

List of Figures **60**

Bibliography **63**

Chapter 1

Introduction

This report is a study of the asynchronous π-calculus and the theory of *concurrent combinators* as models of concurrent computation. The primary aim of this report is to show that the **cc** combinator system is as expressive as the monadic asynchronous π-calculus without summation or matching [9].

This chapter contains some introductory words on process calculi and an informal overview of the **cc**. Chapter 2 examines the asynchronous π-calculus in greater detail and asserts the necessary definitions and results. Chapter 3 presents a formal account of the **cc** and establishes the main result. Chapter 4 discusses some related work and Chapter 5 concludes.

1.1 Process Calculi

Process calculi model concurrent systems as a network of interacting parts, or *agents*, and communication within systems as the interaction of constituent agents. CCS [6] and the π-calculus [8] are based on interaction via atomic synchronisation as a computational primitive, and on the notion of *observational equivalence* between systems. The idea is that a system can be characterised by its interactive behaviour as defined by its capability for communication: to observe the behaviour of a system is to communicate with it.

CCS and π-calculus agents are represented as computational processes that describe behaviour through the capability for action. The actions which an agent may perform are abstracted to internal progression, which has no external effects, and the two fundamental operations of communication, sending and receiving a message (output and input). Systems consist of agents connected by named links, and two agents can interact if they have complementary capabilities for communication over the same link. Processes are formed by composing actions using the basic constructs of parallel composition (concurrency), prefix (for synchronisation), summation (for representing choice) and restriction (of access to links, and thus of capability for action).

The π-calculus is an evolution of CCS that incorporates the notion of system mobility in terms of link dynamics (and also mobile processes). The names of links themselves can be passed as messages allowing new, possibly private, connections to be generated. The model of computation for interactive systems provided by the π-calculus is analogous to that of the λ-calculus for general computation. However, there is no one definitive π-calculus; many variants have emerged as it has been adapted and applied in practise.

The asynchronous π-calculus is a sub-calculus of the π-calculus which models asynchronous communication by precluding synchronisation over output actions. Asynchronous communication can be typified by the separation of the act of sending a message, which can be thought of as putting a message into an intermediary medium, from that of receiving a message, which retrieves the message from the medium. In this way, isolated output actions can be seen as representing such messages in an implicit communication medium; a message is "sent" when the corresponding output action becomes active within the sender process. Chapter 2 examines the asynchronous π-calculus in greater detail.

1.2 Concurrent Combinators [2]

The main content of this report relates to the **cc**, a calculus of concurrent combinators, which we shall show to be capable of encoding, and thus is as expressive as, the monadic asynchronous π-calculus without summation or matching. Note that the **cc** as defined in this report differs slightly from that in [2] due to the inclusion of the replication operator. We first give an informal overview of the **cc** in this introduction, and present a formal account in Chapter 3. See Section 4.1.1 for a description of generalised concurrent combinators.

The **cc** consists of a set of atoms, a grammar by which compound combinators can be constructed from these atoms, the basic algebra over combinator terms, and a reduction relation, again over terms, which formally describes combinator behaviour. One purpose of the **cc**, as with other combinatory systems [1], is to provide a platform from which the properties of other models of computation may be studied, namely the π-calculus and its variants.

The **cc**-atoms are designed to capture the fundamental essence of concurrent computation in the π-calculus. Each atom represents a fixed unit of computational behaviour formed from the decomposition of the name passing and synchronisation operations on which the π-calculus is based. To reflect the specific actions that a π-process may perform, the behaviour that an atom embodies must be instantiated over a particular set of names, forming an *atomic agent*.

Atomic agents can be seen as atoms connected to names in the outside world through *polarised* interaction *ports*. The bindings of ports to names are specified as parameters to the atomic agent, of which the first (leftmost) is called the *subject*. Subject port polarity indicates whether the function of an atom is to perform an input ($-$) or output ($+$) action at the named "location", and the polarity of other ports indicate how descendents of the agent will use the corresponding names.

Computation in the **cc** is based on dyadic interaction of atomic agents as governed by the behaviour that each corresponding atom encapsulates. Two agents may interact if they are connected through complementary (in terms of polarity) subject ports to the same name. The subsequent reduction can be seen as a reaction that consumes the reagents, generating new agents in their stead. π-calculus name passing manifests itself in the communication topology dynamics resulting from interactive reduction. Figure 1.1 illustrates these concepts.

More specifically, **cc** computation is given by the reduction relation on **cc**-terms. **cc**-reduction is defined by a set of atomic *interaction rules* (operational semantics) that specify how a pair of compatible atoms can be reduced; this is the only source of computation

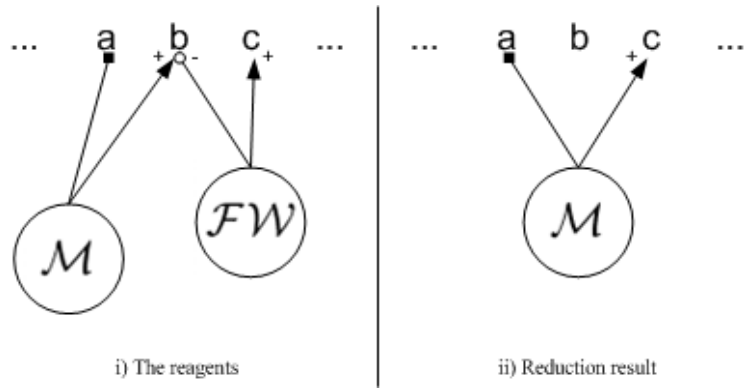

Figure 1.1: Combinator interaction.

within a **cc**-system. Each interaction rule is actually a rule schema where each parameter is a universally-quantified meta-name, and there is only one rule for each pair of compatible atoms. Thus, the behaviour encapsulated by each atom is best understood in terms of how the atoms interact. Following is a summary of the **cc** atoms and their respective interaction rules.

1. ***Message*** $\mathcal{M}(x^+y^\pm)$ The Message delivers the name connected to its value port to the target location, specified by the second and first parameters respectively. The Message is the only atom with a positive subject port, and thus all reductions occur between a Message and one other of the atoms below. The neutral polarity of the second parameter, denoted by $\pm$, reflects that it cannot known without further context how the name being carried will be used in the future. No atoms other than the Message have a neutral port.

2. ***Duplicator*** $\mathcal{D}(x^-u^+v^+) \mid \mathcal{M}(x^+y) \longrightarrow \mathcal{M}(uy) \mid \mathcal{M}(vy)$ The Duplicator creates two new Message atoms, carrying the name received over the first parameter, with targets as specified by its second and third parameters.

3. ***Killer*** $\mathcal{K}(x^-) \mid \mathcal{M}(xy) \longrightarrow \mathbf{0}$ The Killer destroys a Message.

4. ***Forwarder*** $\mathcal{FW}(x^-u^+) \mid \mathcal{M}(x^+y) \longrightarrow \mathcal{M}(uy)$ The Forwarder forwards a message by creating a new Message, carrying the name received over the first parameter, to the target specified by the second parameter.

5. ***Left Binder*** $\mathcal{B}_l(x^-u^+) \mid \mathcal{M}(x^+y) \longrightarrow \mathcal{FW}(yu)$ The Left Binder creates a new Forwarder with the name received over the first parameter as subject and target as specified by the second parameter.

6. ***Right Binder*** $\mathcal{B}_r(x^-u^-) \mid \mathcal{M}(x^+y) \longrightarrow \mathcal{FW}(uy)$ The Right Binder creates a new Forwarder with subject as specified by the second parameter and target set to the name received over the first parameter .

7. ***Synchroniser*** $\mathcal{S}(x^-u^-v^+) \mid \mathcal{M}(x^+y) \longrightarrow \mathcal{FW}(uv)$ The Synchroniser creates a new Forwarder upon receipt of a name over the first parameter between the locations as

specified by the second and third parameters. The received name is discarded; the interaction between a Synchronizer and a Message is of pure synchronisation with no name passing.

The Duplicator and Forwarder perform straightforward name passing operations by generating new message atoms that deliver the object name to the specified target(s). The Left Binder, Right Binder and Synchroniser perform more complicated operations that "use" rather than simply pass the received name. These atoms modify the topology of communication, that is, the links within a combinator network, by creating new Forwarders, and thus could be described as performing name passing meta-operations.

cc-term grammar allows combinator terms, or *agents*, to be constructed from atoms and other combinators through *parallel composition*, *name hiding* and *replication*; these operators are essentially identical to their π-calculus counterparts. Parallel composition is perhaps the definitive construct of any model of concurrent systems, allowing the constituent agents of a system to act and interact independently of, and concurrently with, other agents.

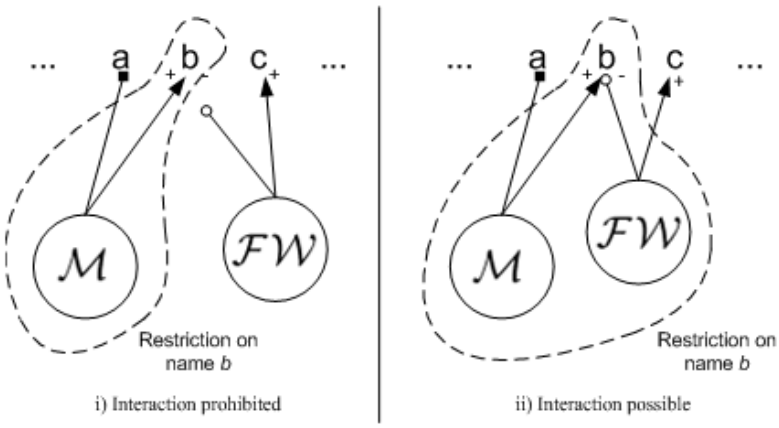

Figure 1.2: Combinator interaction with name hiding.

Name hiding is the division of access to a restricted location between those agents inside and those outside the scope of restriction. In other words, agents bound under the restriction of a name may not interact with atoms outside the restriction over that name (Figure 1.2). Although a restricted agent and a non-restricted agent may have bindings to the same literal identifier, the separate instances of the identifier are conceptually distinct.

The final operator, replication, provides a mechanism for infinite computation. In terms of combinator interaction, replication can be viewed as denoting persistent agents that are not consumed by reaction, enabling them to participate in any number of further reactions. As can be seen from the interaction rules given above, **cc**-reduction always decreases the number of non-Message atoms; thus, unbounded behaviour cannot be expressed in the **cc** without replication. However, Section 4.1 discusses an extension of the **cc** that can represent infinite computation without an explicit replication operator by introducing some additional atoms.

The basic algebra over **cc**-terms defines the property of structural congruence, which relates those terms expected to have the same essential behaviour. As in the π-calculus,

the rules for structural congruence are derived from the meanings of the syntactic constructs; indeed, the rules for structural congruence in the **cc** are identical to those for the asynchronous π-calculus as presented in this report.

1.3 Application of Concurrent Combinators

As mentioned in the preceding section, the **cc** can be used to investigate the properties of the π-calculus and its variants. The result that any asynchronous monadic π-term (without summation or matching) can be expressed as a behaviourally-equivalent **cc**-term (Chapter 3) is reminiscent of the fact that any λ-expression can be expressed using only the **S** and **K** combinators [1]. In the latter, the decomposition of β-reduction into the primitive higher-order functions denoted by **S** and **K** allows λ-abstraction to be replaced; in the former, the decomposition of the asynchronous message receipt operation into the seven basic **cc**-atoms replaces the input prefix.

The properties of combinatory logic, namely its expressiveness despite the small set of combinators, the relative simplicity of combinator reduction and the absence of formal variables, make it of interest for both theoretical and practical reasons. Similarly, the analysis of π-interaction from which the **cc** is derived offers insight into some fundamental aspects of concurrent computation in the π-calculus, such as the relationship between atoms due to name sharing as well as term-level composition [2].

It can intuitively be seen how the compositional, binding and synchronisation components of the π-calculus name passing primitive have been devolved into the **cc**-atoms. The close relationship between the **cc** and the π-calculus makes embedding the former within the latter as a sub-calculus straightforward: after defining a mapping that preserves the essential behaviour of each atom, it is then a matter of verifying that π-reduction between the translated atoms corresponds to **cc**-reduction. Figure 1.3 illustrates an example of this.

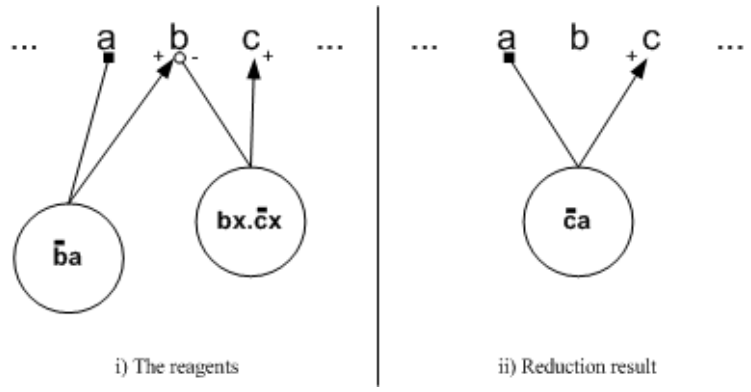

Figure 1.3: Interaction of **cc** combinators embedded into the π-calculus.

By bringing the **cc** within the π-calculus, we are now able to use machinery provided by the latter, namely bisimulation, to analyse the properties of the former. This facilitates the verification of the main result stated above, that the **cc** is capable of encoding the asynchronous π-calculus without loss of expressiveness. Chapter 3 covers the relationship between the π-calculus and the **cc** (illustrated in Figure 1.4) in more detail.

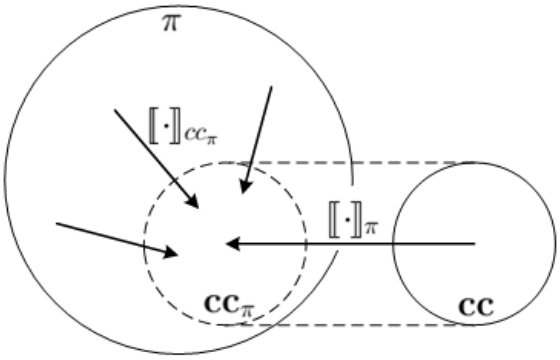

Figure 1.4: Encoding of the **cc** within the π-calculus and vice versa.

Establishing the expressiveness of the **cc**, and the roles of the **cc**-atoms, provides the aforementioned platform for investigating other sub-calculi of the π-calculus. In general, the technique of encoding, or being unable to encode, a language or calculus within another can reveal information regarding their comparative expressiveness. The encapsulation of computational behaviour within atomic combinatory units enables experimentation with the inclusion and exclusion of particular atoms, and can be used to characterise an encoded language.

The mapping from asynchronous π-terms to their equivalent counterparts in the **cc**, which will be defined in Chapter 3, serves as a template, and can sometimes be reused, for encoding such calculi in the **cc**. The **cc** can thus be used as a tool for analysing the properties of a particular calculus with respect to modelling concurrent systems, if it differs from others and how, and to discover what (real-world) situations they can be applied to.

Chapter 2

The Asynchronous π-Calculus

The π-calculus and the asynchronous π-calculus were briefly introduced in Section 1.1. This chapter examines the latter and its properties in greater detail, and prepares the necessary background material for Chapter 3. The primary reference source for this chapter is [8], and many of the definitions presented in this chapter, particularly in Section 2.4, are cited from or based on this source. The other main reference sources are [9] and [5].

In the asynchronous π-calculus, processes independently act and interact through asynchronous communication. Names are the simplest entities, and typically denote communication links, or channels. Processes are defined with respect to the names over which the basic actions, sending and receiving a name, can be performed. When applied to specific situations, names may represent values such as locations, passwords, and processes themselves.

Process interaction, which in the π-calculus is the same phenomenon as inter-process communication, is essentially a name-passing operation; the names stored within a process may be updated as a result of interaction and, together with the synchronisation mechanism, may enable further action. As will be explained in Section 2.1, asynchronicity is modelled by precluding synchronisation over output actions.

There are many variants of the asynchronous π-calculus differing in the abstractions that each captures: single or multiple name-passing, choice between input actions, explicit internal action, the capabilities that may be activated by communication, internal and external mobility, and many others. In this report, we consider the monadic asynchronous π-calculus without summation or matching as presented in [9], which will simply be referred to as the asynchronous π-calculus, or just the π-calculus, if there is no confusion. Methods for encoding certain behaviours that are inherent to more expressive calculi are discussed briefly in Section 2.5.

Firstly, the syntax and structure of the asynchronous π-calculus is defined. Two relations on asynchronous π-terms that govern their behaviour, reduction and transition, and their respective properties are then examined. Following this is the main purpose of this chapter, the identification of a suitable process equivalence that relates processes with the same essential behaviour. This forms the foundations over which the theory of concurrent combinators for the π-calculus is developed in Chapter 3.

As will be seen, several useful process equivalences exhibit special properties for the asynchronous π-calculus that do not hold for the full π-calculus. These theoretical results complement the practical motivations for the asynchronous π-calculus: many real-world

systems consist of components that communicate asynchronously, and many implementations of synchronous communication are built from asynchronous primitives.

2.1 Syntax

Definition 2.1.1 (names and processes) The set of terms, or *processes*, in the monadic asynchronous π-calculus without summation or matching, P_π, ranged over by $P, Q, R, ...$ is given by the syntax

$$P \quad ::= \quad P \,|\, Q \quad | \quad xz.P \quad | \quad (\boldsymbol{\nu}\, z)\, P \quad | \quad !\, P \quad | \quad \overline{x}y \quad | \quad \mathbf{0}\,.$$

where the lower-case letters $x, y, z, ...$ range over a countably-infinite set of *names*. Given a name x, the corresponding *co-name* is $\overline{x}$. We abbreviate $(\boldsymbol{\nu}\, z_1)\,, ..., (\boldsymbol{\nu}\, z_n)$ to $(\boldsymbol{\nu}\, z_1, ..., z_n)$ or just $(\boldsymbol{\nu}\, \tilde{z})\,$.

Occurrences of a name z in $xz.P$ and $(\boldsymbol{\nu}\, z)\, P$ are *binding* with *scope* P. The binding occurrence and all other occurrences within P of z are *bound*, and the occurrence of a name is *free* if not bound. The sets of bound and free names [5] occurring within P are denoted by $\mathsf{bn}(P)$ and $\mathsf{fn}(P)$ respectively. $xz.P$ will be abbreviated to $x.P$ if $z \notin \mathsf{fn}(P)$, and $(\boldsymbol{\nu}\, z)\, \overline{x}z$ to $\overline{x}$. The occurrence of a process Q within a process P is *guarded* if it occurs under a xz prefix, and *unguarded* otherwise. □

The first (leftmost) operator denotes parallel composition. The following three operators, input prefix, name restriction and replication respectively, bind more tightly than parallel composition, and parentheses are used to disambiguate processes when written as linear expressions. The remaining two terms denote the output and nil processes respectively. It is important to note that, under this definition, a name that does not occur in a process is neither bound nor free in that process.

These syntactical constructs from which processes are assembled represent abstracted aspects of communication and computation within a concurrent system. In general, names can be thought of as communication channel identifiers. The basic actions that processes may perform are to receive and send messages on the channels specified by the leftmost names of the input prefix and output term respectively. Input prefixing abstracts the receipt of a message by substituting all bound occurrences of the binding name with the received name. An asynchronous message send can be seen as occurring when an output term becomes unguarded, that is, when all prefixes have been removed by performing input actions; the resulting output particle then represents a message in an implicit communication medium. In this way, the input prefix enables synchronisation over input actions.

There is no syntactic distinction between instantiable names and non-instantiable names, or in other words, between variables and constants. The free names of a process define its capabilities for action: a process can only receive or send via a name if the name is free in that process. For two processes to interact, one must have the capability to send, and the other to receive, a message over a name free in both. Thus, name passing forms the basic computational primitive of the π-calculus, and process interaction and inter-process communication become equivalent under this abstraction.

By controlling the binding of names, restriction determines which processes can interact: processes within the scope of restriction are able to interact via the restricted name

$$(P \mid Q)\sigma \stackrel{\text{def}}{=} P\sigma \mid Q\sigma$$
$$(xz.P)\sigma \stackrel{\text{def}}{=} (xv)\sigma.P\sigma$$
$$((\boldsymbol{\nu}\, z)\, P)\sigma \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\, P\sigma$$
$$(!\, P)\sigma \stackrel{\text{def}}{=} !\, P\sigma$$
$$\mathbf{0}\sigma \stackrel{\text{def}}{=} \mathbf{0}$$

Table 2.1: Application of substitution.

but not with outside processes. Composition enables processes to proceed and interact independently of other processes within the system, and replication can be thought of as the infinite composition of a particular process, thus allowing infinite computation to be represented. Replication is a fundamental element of the π-calculus as unbounded behaviour cannot be expressed without it. Section 2.2 provides a more formal definition of process behaviour. The above syntactic constructs were also introduced for the **cc** in Section 1.2.

As mentioned above, the absence of output prefixing is the source of asynchronicity in the asynchronous π-calculus: the inability to synchronise over output actions means that a process cannot directly determine when another has received a message. As any number of unguarded outputs may appear within a process, the implicit communication medium has unbounded capacity, and, due to the commutativity of parallel composition (Section 2.2), no ordering of "in-transit" messages is preserved. For these reasons the summation operator, which represents choice of action within a process, is disallowed: if given the choice between performing input and output actions, a process can itself control whether to discard messages by performing an input. Although it may be desirable to model a lossy medium over which sent messages may not be received, this is better achieved by adding to the system terms such as $!\, xz.\mathbf{0}$ which can non-deterministically consume messages on the channel x. The match operator is excluded from the asynchronous π-calculus as a trade-off for certain proof techniques at the cost of expressiveness [8].

Definition 2.1.2 (substitution and α-equivalence)

(i) A name is *fresh* in P if the name does not occur in P.

(ii) The application of a *substitution* $\{y_1, ..., y_n/x_1, ..., x_n\}$ to a process P, denoted by $P\{y_1, ..., y_n/x_1, ..., x_n\}$, replaces each free occurrence of $x_1, ..., x_n$ with $y_1, ..., y_n$ respectively according to the rules given in Table 2.1. Names bound within P that clash with any of $y_1, ..., y_n$ are implicitly α-converted, as defined below, to fresh names. Substitutions are ranged over by σ.

(iii) *α-conversion* of a process is the replacement of a subterm $xz.Q$ by $xw.Q\{w/z\}$, or $(\boldsymbol{\nu}\, z)\, Q$ by $(\boldsymbol{\nu}\, w)\, Q\{w/z\}$, where in each case w does not already occur in Q.

(iv) P and Q are α-equivalent, $P =_\alpha Q$, if Q can be obtained from P via a finite number of α-conversion steps. □

α-conversion is the meta-operation of renaming consistently the occurrences of a name bound within a process. Implicit α-conversion of bound names that clash with names

$P \equiv Q$, if $P =_\alpha Q$	(SC-ALPHA)
$P \mid (Q \mid R) \equiv (P \mid Q) \mid R$	(SC-COMP-ASSOC)
$P \mid Q \equiv Q \mid P$	(SC-COMP-COMM)
$P \mid \mathbf{0} \equiv P$	(SC-COMP-INACT)
$(\boldsymbol{\nu}\, z, w)\, P \equiv (\boldsymbol{\nu}\, w, z)\, P$	(SC-RES-ASSOC)
$(\boldsymbol{\nu}\, z)\, (P \mid Q) \equiv P \mid (\boldsymbol{\nu}\, z)\, Q$, if $z \notin \mathsf{fn}(P)$	(SC-RES-COMP)
$(\boldsymbol{\nu}\, z)\, \mathbf{0} \equiv \mathbf{0}$	(SC-RES-INACT)
$!P \equiv P \mid !P$	(SC-REP)

Table 2.2: The axioms of structural congruence.

received via input actions avoids the unintended capture of the latter by the binding occurrences of the former. Intuitively it can be seen that α-conversion preserves each difference between names and thus α-equivalence describes literal process equality up to this syntactic constraint.

2.2 Reduction

The reduction relation on processes, $\longrightarrow$, describes how a system represented as π-term can evolve independently of its environment as a result of progressive intraaction. The structural congruence relation enables this process by bringing together potential intraactors.

Definition 2.2.1 (equivalence, context and congruence)

(i) An *equivalence relation* $\mathcal{R}$ on processes is a binary relation where

$P \mathcal{R} P$	(REFL)
if $P \mathcal{R} Q$ then $Q \mathcal{R} P$	(SYMM)
if $P \mathcal{R} Q$ and $Q \mathcal{R} R$ then $P \mathcal{R} R$	(TRANS)

(ii) A *context* is obtained when an occurrence of $\mathbf{0}$ in a term given by the process grammar in Definition 2.1.1 is replaced with a hole $[\cdot]$. Given a process P and a context $\mathcal{C}$, $\mathcal{C}[P]$ is the literal replacement of the $[\cdot]$ in $\mathcal{C}$ by P.

(iii) A *congruence* is an equivalence relation on processes that satisfies the condition

if $P \mathcal{R} Q$ then $\mathcal{C}[P] \mathcal{R} \mathcal{C}[Q]$	(CONG)

□

Definition 2.2.2 (Structural Congruence) *Structural congruence*, $\equiv$, is the smallest congruence on processes that satisfies the axioms in Table 2.2. □

Structural congruence allows the syntactic structure of a process to be manipulated whilst preserving the essential behaviour of the process, or in other words, structural

$$\text{(R-PAR)}\quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{(R-RES)}\quad \frac{P \longrightarrow P'}{(\boldsymbol{\nu}\, z)\, P \longrightarrow (\boldsymbol{\nu}\, z)\, P'}$$

$$\text{(R-COMM)}\quad \frac{}{\overline{x}y \mid xz.P \longrightarrow P\{y/z\}} \qquad \text{(R-STRUCT)}\quad \frac{P \equiv Q \longrightarrow Q' \equiv P'}{P \longrightarrow P'}$$

Table 2.3: The rules of π-reduction.

congruence identifies those processes with the same behavioural aspects of interest. This is achieved through the careful specification of axioms that express the intrinsic meanings of the π-operators and allow only the desired manipulations.

Example 2.2.3 Rule (SC-REP) captures the idea of replication as an infinite composition.

$$\begin{aligned} !\,P &\equiv P \mid !\,P && \text{(SC-REP)} \\ &\equiv P \mid P \mid !\,P && \text{(SC-REP)} \\ &\equiv \ldots \end{aligned}$$

□

Example 2.2.4 We prove the intuitive result that restricting a name not free in a process has no effect. Take P where $z \notin \mathsf{fn}(P)$.

$$\begin{aligned} (\boldsymbol{\nu}\, z)\, P &\equiv (\boldsymbol{\nu}\, z)\, (P \mid \mathbf{0}) && \text{(SC-COMP-INACT)} \\ &\equiv P \mid (\boldsymbol{\nu}\, z)\, \mathbf{0} && \text{(SC-RES-COMP)} \\ &\equiv P \mid \mathbf{0} && \text{(SC-RES-INACT)} \\ &\equiv P && \text{(SC-COMP-INACT)} \end{aligned}$$

□

Definition 2.2.5 (reduction) The *reduction relation*, $\longrightarrow$, is the smallest relation between processes satisfying the rules in Table 2.3. □

Reduction defines how a system can perform internal action through the interaction of its constituent components. The reduction relation can be manifested as a term-rewriting system: (R-STRUCT) allows (sub)terms to be manipulated into a form which can be reduced, and ensures by construction that the relation is closed under structural congruence by its application after a reduction. Intuitively, it can be seen that the reduction relation captures the intended semantics of the π-operators as described in Section 2.1. The absence of $P \longrightarrow P'$ implies $!\,P \longrightarrow !\,P'$ is consistent with the notion of replication as infinite composition as such a rule would express infinite progress in a single step.

Example 2.2.6 Let $P \stackrel{\text{def}}{=} \overline{x}a \mid xz.\overline{z}y \mid \overline{x}b$. We have both

$$P \longrightarrow P' \stackrel{\text{def}}{=} \overline{a}y \mid \overline{x}b \qquad \text{(R-PAR, R-COMM)}$$

and

$$P \longrightarrow P'' \stackrel{\text{def}}{=} \overline{b}y \mid \overline{x}a \qquad \text{(R-STRUCT, R-PAR, R-COMM)}$$

which shows that reduction is non-deterministic, as both reductions are valid, and also non-confluent, as neither P' nor P'' can be further reduced. □

Example 2.2.7 We illustrate name hiding and *scope extrusion*, and also the role of (R-STRUCT) in manipulating processes into forms which can be reduced. The first case again demonstrates that restriction of a name not free in a process has no effect (see Example 2.2.4).

$$(\boldsymbol{\nu}\, x)\, u.\mathbf{0} \mid \overline{u}y \longrightarrow \mathbf{0} \qquad \text{(R-STRUCT, R-COMM)}$$

The following shows how name restriction is enforced by the side-condition of (SC-RES-COMP).

$$(\boldsymbol{\nu}\, x)\, x.\mathbf{0} \mid \overline{x}y \not\longrightarrow$$

Reduction is not possible because the component on the right cannot be brought within the restriction. Using α-conversion to by-pass this constraint is, of course, futile as the subjects of the actions will no longer correspond when (R-COMM) needs to be applied.

However, scope extrusion allows a restricted process to send a private channel name to another process,

$$\begin{aligned}(\boldsymbol{\nu}\, x)\, (\overline{v}x \mid x.\mathbf{0}) \mid vz.\overline{z}y &\longrightarrow (\boldsymbol{\nu}\, x)\, (x.\mathbf{0} \mid \overline{x}y) && \text{(R-STRUCT, SC-RES-PAR, R-RES, R-COMM)}\\ &\longrightarrow \mathbf{0} && \text{(R-RES, R-COMM)}\end{aligned}$$

and the subsequent reduction within the restriction can be seen as communication between the processes over this private channel. In the final case, we again use scope extrusion to bring the component on the right within the restriction.

$$(\boldsymbol{\nu}\, x)\, (\overline{v}x \mid x.\mathbf{0}) \mid vz.(\overline{z}y \mid \overline{x}y) \longrightarrow (\boldsymbol{\nu}\, w)\, (w.\mathbf{0} \mid \overline{w}y \mid \overline{x}y) \quad \text{(R-STRUCT, SC-RES-PAR, R-RES, R-COMM)}$$

In order to do this, the component on the left was first α-converted by (R-STRUCT) to satisfy the side-condition of (SC-RES-COMP). This shows that the sent name is genuinely private and is not known to any other process. Thus, the combination $(\boldsymbol{\nu}\, z)\, \overline{x}z$ expresses the capability to simultaneously create and send a fresh name, and name clashes can always be avoided as there are an infinite number of fresh names to chose from. □

2.3 Transition

Reduction as a description of system behaviour is limited in that it cannot say how a system can interact with its environment, asking only enough of process syntax to satisfy intraaction and no more. Transition overcomes these limitations by utilising more of

$$(\text{ALPHA})\quad \frac{P =_\alpha Q \xrightarrow{l} Q' =_\alpha P'}{P \xrightarrow{l} P'} \qquad (\text{IN}_s)\quad \frac{}{xy.P \xrightarrow{xy} P\{y/x\}} \qquad (\text{OUT})\quad \overline{x}y \xrightarrow{\overline{x}y} \mathbf{0}$$

$$(\text{PAR-L})\quad \frac{P \xrightarrow{l} P'}{P \mid Q \xrightarrow{l} P' \mid Q}\ \mathsf{bn}(l) \cap \mathsf{fn}(Q) = \emptyset \qquad (\text{REP})\quad \frac{P \mid !\,P \xrightarrow{l} P'}{!\,P \xrightarrow{l} P'}$$

$$(\text{RES})\quad \frac{P \xrightarrow{l} P'}{(\boldsymbol{\nu}\, z)\,P \xrightarrow{l} (\boldsymbol{\nu}\, z)\,P'}\ z \notin \mathsf{fn}(l) \cup \mathsf{bn}(l) \qquad (\text{OPEN})\quad \frac{P \xrightarrow{\overline{x}z} P'}{(\boldsymbol{\nu}\, z)\,P \xrightarrow{\overline{x}(z)} P'}\ x \neq z$$

$$(\text{COMM-L})\quad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{xy} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad (\text{CLOSE-L})\quad \frac{P \xrightarrow{\overline{x}(z)} P' \quad Q \xrightarrow{xz} Q'}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\, z)\,(P' \mid Q')}\ z \notin \mathsf{fn}(Q)$$

Table 2.4: The rules of synchronous early transition.

the information that can derived from process syntax, namely the specific actions that a process can perform on its environment. The transition relations on processes are labelled for each transitional action, where $P \xrightarrow{l} Q$ expresses that P can become Q by performing the action denoted by l. As will be discussed below, this report assumes the synchronous early transition relations except where stated otherwise.

Definition 2.3.1 (labels and actions) The *labels*, ranged over by $l, l', ...$, are given by

$$Lab \quad ::= \quad xy \quad | \quad \overline{x}y \quad | \quad \overline{x}(z) \quad | \quad \tau$$

and denote *actions*. τ denotes the *invisible* action; the others are *visible*. The first (left-most) name of a label specifying a visible action is the *subject*, and the second the *object*. The free names of a label are the names that occur in the label, with the exceptions of z in $\overline{x}(z)$, which is bound, and τ, which has no subject or object and contains no occurrences of any names. The application of a substitution to a label concerns only the free names of the label. □

The first label denotes the action of receiving the name y via the name x, the second is sending y via x, and the third is sending a fresh name via x. τ represents the internal process action.

Definition 2.3.2 (Transition) The synchronous early transition relations for the asynchronous π-calculus, $\{\xrightarrow{l} \mid l \in Lab\}$, are the smallest relations defined by the rules in Table 2.4 along with (PAR-R), (COMM-R) and (CLOSE-R), the symmetric forms of (PAR-L), (COMM-L) and (CLOSE-L), in which the roles of the left and right components are exchanged. □

Transition takes the position of an observer who can communicate with a system as well as being able to detect changes in system state. Transition, unlike reduction, reflects the observation of explicit actions, specifically the messages that a process can send and receive. Both reduction and transition have intuitive justification for their respective formal expression of the meanings of the π-operators and concurrent computation. However,

transition goes beyond reduction as a description of process behaviour by exposing the capability for communication as well as internal action. Indeed, internal transition and reduction are the same up to structural congruence [8].

The terminology *early* transition refers to abstracted message receipt, that is, the instantiation of bound names in a process by the object of the message, occurring when the input action of the receiver is inferred. Alternatively, the *late transition relations* perform this instantiation when communication is inferred by replacing the (synchronous) early input action

$$xy.P \xrightarrow{xy} P\{y/x\}$$

with bound-input actions of the form

$$x(z).P \xrightarrow{x(z)} P$$

where z serves as a placeholder for a received name to be instantiated in

$$(\text{L-COMM-L}) \quad \frac{P \xrightarrow{\overline{x}y} P' \quad Q \xrightarrow{x(z)} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'\{y/z\}}$$

The choice of communication model as determined by a particular set of transition relations impacts upon how process behaviour can be compared. However, the three main behavioural equivalences employed in the full π-calculus coincide for the asynchronous π-calculus. It should also be noted that both of the above transition relations allow any name to be received via an input action, a feature highlighted by the use of bound placeholders in late input. A fourth behavioural equivalence based on the receipt of a single fresh name, a condition too weak to be useful in the full π-calculus, also coincides with the three previously mentioned equivalences in the asynchronous π-calculus. Section 2.4 discusses behavioural equivalence in more detail.

Example 2.3.3 The rule (ALPHA) avoids clashes between bound and received names; in reduction, this feature was encompassed by (R-STRUCT) as structural congruence contains α-equivalence.

Let $S \stackrel{\text{def}}{=} P \mid Q$, where $P \stackrel{\text{def}}{=} \overline{x}z$ and $Q \stackrel{\text{def}}{=} \mid (\boldsymbol{\nu}\, z)\, xw.\overline{w}z$. S has an internal transition from rules (COMM-L), (OUT), (ALPHA), (RES) and (IN_s).

$$\dfrac{\dfrac{}{\overline{x}z \xrightarrow{\overline{x}z} \mathbf{0}} \quad \dfrac{\dfrac{\dfrac{}{xw.\overline{w}y \xrightarrow{xz} \overline{z}y}}{(\boldsymbol{\nu}\, z)\, xw.\overline{w}z =_\alpha (\boldsymbol{\nu}\, y)\, xw.\overline{w}y \xrightarrow{xz} (\boldsymbol{\nu}\, y)\, \overline{z}y}}{(\boldsymbol{\nu}\, z)\, xw.\overline{w}z \xrightarrow{xz} (\boldsymbol{\nu}\, y)\, \overline{z}y}}{\overline{x}z \mid (\boldsymbol{\nu}\, z)\, xw.\overline{w}z \xrightarrow{\tau} \mathbf{0} \mid (\boldsymbol{\nu}\, y)\, \overline{z}y}$$

As mentioned in Section 2.1, α-conversion is often used implicitly. □

Example 2.3.4 We again consider scope extrusion which is performed using the two explicit rules for opening and closing the scope of a restriction, (OPEN) and (CLOSE). Note that the side condition of (RES) ensures that only (OPEN) can be used for scope opening, as well as preventing communication over the restricted name. The latter property is also

enforced by the side condition of (OPEN).

Let $S \stackrel{\text{def}}{=} (P \mid R) \mid Q$, where $P \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\, \overline{u}x$ and $Q \stackrel{\text{def}}{=} \overline{u}z.\overline{z}$. Then, by (CLOSE-L), (PAR-L), (OPEN), (OUT) and (IN),

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\overline{u}x \xrightarrow{\overline{u}x} \mathbf{0}}}{(\boldsymbol{\nu}\, x)\, \overline{u}x \xrightarrow{\overline{u}(x)} \mathbf{0}}}{(\boldsymbol{\nu}\, x)\, \overline{u}x \mid R \xrightarrow{\overline{u}(x)} \mathbf{0} \mid R} \qquad \dfrac{}{uz.\overline{z}y \xrightarrow{ux} \overline{x}y}}{((\boldsymbol{\nu}\, x)\, \overline{u}x \mid R) \mid \overline{u}z.\overline{z}y \xrightarrow{\tau} (\boldsymbol{\nu}\, x)\, (\mathbf{0} \mid R \mid \overline{x}y)}$$

provided that $x \notin \mathsf{fn}(R)$ as required to fulfil the side condition of (PAR-L), ensuring that the sent name is fresh. Thus, the side conditions of (CLOSE-L), (PAR-L) and their symmetric forms serve the same purpose as (SC-RES-COMP) in reduction (see Example 2.2.7). However, (ALPHA) can again be used to avoid name clashes by allowing suitably fresh names to be chosen. □

In the full π-calculus, observation of process behaviour is the same as interacting with a process by performing complementary actions to the capabilities of the process. This is possible because both input and output actions can have continuations: prefixing provides a mechanism for synchronisation, and thus the occurrence of either type of action can be directly observed through dependent components. This is the meaning of the terminology *synchronous* transition.

In the asynchronous π-calculus, output continuations are precluded to affect asynchronous communication. However, input prefixing allows output actions to be observed as in the full π-calculus, and input actions can be observed indirectly. For example, the component

$$(\boldsymbol{\nu}\, z)\, (\overline{x}z \mid zw.P)$$

can detect that the name z, private to the component, has been received if the receiver replies by sending a message on z. Thus, the asynchronous π-calculus as presented above models asynchronous communication through the modification of process syntax, whilst adopting the existing transition relations based on synchronous (early) transition. This report is mainly concerned with process equivalences based on this particular representation of asynchronicity. Alternatively, asynchronous communication could be modelled using *asynchronous transition relations*, as discussed in Section 2.6.

2.4 Behavioural Equivalence

The reduction and transition relations express the behaviour of systems represented as π-calculus terms. These formal descriptions of process behaviour permit various methods for comparing processes, in particular, establishing whether two processes have equivalent behaviour. These methods are based on the general technique of bisimulation, a process governed by two sets of rules: what processes are allowed to do, and what is required of a process with respect to the capabilities of another to qualify for equivalence. The former can be defined in terms of either reduction, or more usefully (as will be seen), transition. The latter is given by the rules of the particular bisimulation being employed. Bisimilarity

relations differ in terms of their discriminatory strength, robustness across differing variants of the π-calculus, and the tractability of their corresponding proof techniques. The combination of these factors determines how useful a bisimilarity equivalence is.

Definition 2.4.1 (bisimulations and bisimilarities) A *bisimulation* is an equivalence relation on a subset of processes closed under its defining conditions. A *bisimilarity* is the union of all corresponding bisimulations that satisfy the conditions. Thus, a bisimilarity is the largest such bisimulation, and a bisimulation can be defined implicitly within the definition of the bisimilarity. □

This notion of bisimulation is sufficient for this report, and assumes that bisimulations will be defined to respect the properties of equivalence. However, symmetry is not a fundamental requirement for bisimulation. The fact that a union of bisimulations will be a bisimulation (the union of a finite number of closed sets is closed) gives transitivity and justifies the above definition of a bisimilarity. Formally, we should show that a bisimilarity is an equivalence from the corresponding bisimulation definition, but we shall employ Definition 2.4.1 in this report as an informal "shortcut". As an example, consider reduction bisimilarity, which is defined firstly in terms of reduction bisimulation.

Definition 2.4.2 (reduction bisimilarity I) A relation $\mathcal{S}$ is a *reduction bisimulation* if whenever $(P, Q) \in \mathcal{S}$,

(1) $P \xrightarrow{\tau} P'$ implies $Q \xrightarrow{\tau} Q'$ for some Q' with $(P', Q') \in \mathcal{S}$
(2) $Q \xrightarrow{\tau} Q'$ implies $P \xrightarrow{\tau} P'$ for some P' with $(P', Q') \in \mathcal{S}$

Reduction bisimilarity is the union of all reduction bisimulations; P and Q are *reduction bisimilar* if $(P, Q) \in \mathcal{S}$ for some reduction bisimulation $\mathcal{S}$. □

The definition of reduction bisimilarity can be re-expressed under Definition 2.4.1 to implicitly include the definition of reduction bisimulation as follows. The existential quantification in the former definition is absorbed into the $\xrightarrow{\tau}\mathcal{B}_\tau$ operator composition.

Definition 2.4.3 (reduction bisimilarity II) *Reduction bisimilarity* is the bisimilarity relation on processes, $\mathcal{B}_\tau$, such that whenever $P\,\mathcal{B}_\tau\,Q$, $P \xrightarrow{\tau} P'$ implies $Q \xrightarrow{\tau}\mathcal{B}_\tau\, P'$. □

A reduction bisimulation is essentially a comparison of possible τ-transition derivation sequences, or in other words, the capability for internal action, between two processes. For example, any two processes that have no τ-transitions, such as $\mathbf{0}$ and $xz.\mathbf{0}$ are reduction bisimilar. Consequently, reduction bisimilarity does not provide an appropriate description of equivalent process behaviour. However, this basic instance of reduction-based equivalence can be improved upon by forming a congruence from reduction bisimilarity using Definition 2.2.1 (iii).

Definition 2.4.4 (reduction congruence) Processes P and Q are *reduction congruent* if $P\,\mathcal{B}_\tau\,Q$ implies $\mathcal{C}[P]\,\mathcal{B}_\tau\,\mathcal{C}[Q]$ for every context $\mathcal{C}$. Thus, *reduction congruence* is the largest congruence included in $\mathcal{B}_\tau$. □

The definition states that two processes are reduction congruent if they reduction bisimilar in all contexts. Thus, reduction congruence can distinguish between $\overline{x}a$ and $\overline{x}b$ using $\mathcal{C} \stackrel{\text{def}}{=} [\cdot] \mid xw.(\overline{w}c \mid az.\mathbf{0})$, and between $\overline{u}y$ and $\overline{v}y$ using $\mathcal{C}' \stackrel{\text{def}}{=} [\cdot] \mid uz.\mathbf{0}$. However, reduction congruence is still limited as a process equivalence by its absolute dependence on internal transition. For example, the process $P \stackrel{\text{def}}{=\!!} (\overline{x}y \mid xz.\mathbf{0})$ cannot be distinguished from $P \mid Q$ for any Q as $P \stackrel{\tau}{\longrightarrow} \equiv P$. More is required than the simple comparison of independent process evolution to form a satisfactory process equivalence.

Definition 2.4.5 (strong barbs) Let μ range over the set of names and co-names. The strong observability predicate, or *barb*, $\downarrow_\mu$, is defined by

(1) $P \downarrow_x \iff P \equiv (\boldsymbol{\nu}\, \tilde{z})\,(xy.Q \mid R)$ for some Q, R and where $x \notin \{\tilde{z}\}$
(2) $P \downarrow_{\overline{x}} \iff P \equiv (\boldsymbol{\nu}\, \tilde{z})\,(\overline{x}y \mid Q)$ for some Q and where $x \notin \{\tilde{z}\}$. □

In other words, $P \downarrow_\mu$ expresses that P is immediately capable of performing an output action with subject μ. The definition of barbs formalises observability, and is in accordance with the notion of observation and communication as the same phenomenon in the π-calculus. More specifically, the observations of interest are the capabilities of a process for performing input and output actions; the perspective of an observer is from within the system as a special component added for this purpose. Reduction bisimilarity can now be augmented to incorporate observability.

Firstly, note that in the asynchronous π-calculus, the exclusion of output continuations means that it would not be inappropriate to restrict the observables to be the co-names. This strategy would yield asynchronous behavioural equivalences that compare only the output barbs, $\downarrow_{\overline{x}}$, for a pair of processes. Such equivalences would relate $P_1 \stackrel{\text{def}}{=} xz.\overline{x}z$ to $Q \stackrel{\text{def}}{=} \mathbf{0}$, and also $P_2 \stackrel{\text{def}}{=\!!} xz.\overline{x}z$ to Q, expressing that a communication buffer cannot be observed under asynchronous communication. However, we will continue with both input and output barbs; this is justified by the fact that strong barbed congruence and barbed congruence, which will be defined shortly, imply asynchronous strong barbed congruence and asynchronous barbed congruence respectively [8].

Definition 2.4.6 (strong barbed bisimilarity) *Strong barbed bisimilarity* is the bisimilarity relation on processes, $\dot{\sim}$, such that whenever $P \dot{\sim} Q$,

(1) $P \downarrow_\mu$ implies $Q \downarrow_\mu$
(2) $P \stackrel{\tau}{\longrightarrow} P'$ implies $Q \stackrel{\tau}{\longrightarrow} \dot{\sim} P'$. □

A strong barbed bisimulation is defined as a reduction bisimulation that respects observability. Thus, strongly barbed bisimilar processes are reduction bisimilar processes with matching barbs at each τ-transition step in a bisimulation containing the processes. This is due to Definition 2.4.1, which enables bisimilarity to be shown between two processes by exhibiting such a bisimulation, and because strong output barbs indicate only the immediate actions that a process may perform. This highlights the convenience of using τ-transition rather than reduction in the definition of (strong barbed) bisimilarity, as it generally allows the subsequent proofs to be more concise.

However, the independence of barbs from the objects of observable actions renders strong barbed bisimulation unsatisfactory as a process equivalence: any two processes that

can evolve over a common progression of internal action with corresponding capabilities for output over the same channels are deemed behaviourally equivalent, regardless of the content of sent messages. As a result, any two single output terms with the same subject, such as $\overline{x}a$ and $\overline{x}b$, are strongly barbed bisimilar, as are $\overline{x}a$ and $(\boldsymbol{\nu}\, z)\,\overline{x}z$. There is also no consideration of how the capabilities expressed by barbs may actually be exercised; strong barbed bisimilarity is inadequate when multiple derivations exist for interleaving output and reduction actions. For example, $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\overline{u}y \mid \overline{x} \mid x.\mathbf{0}) \mathrel{\dot{\sim}} S \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(!\,\overline{u}y \mid \overline{x} \mid x.\mathbf{0})$. These problems are addressed by strong barbed congruence.

Definition 2.4.7 (strong barbed congruence) Processes P and Q are *strong barbed congruent*, $P \mathrel{\dot{\simeq}} Q$, if $\mathcal{C}[P] \mathrel{\dot{\simeq}} \mathcal{C}[Q]$ for every context $\mathcal{C}$. Thus, $\mathrel{\dot{\simeq}}$ is the largest congruence in $\mathrel{\dot{\sim}}$. □

The extra discriminatory power of strong barbed congruence with respect to strong barbed bisimilarity is analogous to that of reduction congruence with respect to reduction bisimilarity. In particular, contexts can now be used to distinguish between processes, such as those mentioned above, based on the names they can send. For instance, $\mathcal{C} \stackrel{\text{def}}{=} [\cdot] \mid xz.(\overline{z} \mid a.\mathbf{0})$ is enough to show both $\overline{x}a \not\mathrel{\dot{\simeq}} \overline{x}b$ and $\overline{x}a \not\mathrel{\dot{\simeq}} (\boldsymbol{\nu}\, z)\,\overline{x}z$, and $\mathcal{C}' \stackrel{\text{def}}{=} [\cdot] \mid u.\mathbf{0}$ distinguishes R and S as defined above.

Strong barbed congruence is a satisfactory process equivalence [8]. To apply the π-calculus, however, it is desirable to abstract away from internal action, which strong barbed congruence considers on equal terms to visible actions.

Definition 2.4.8 (weak transition)

(i) $\Longrightarrow$ is the reflexive and transitive closure of $\stackrel{\tau}{\longrightarrow}$.

(ii) $\stackrel{l}{\Longrightarrow}$ is $\Longrightarrow\stackrel{l}{\longrightarrow}\Longrightarrow$, where $l \in Lab$.

$\stackrel{\hat{l}}{\Longrightarrow}$ stands for $\Longrightarrow$ if $l = \tau$, and $\stackrel{l}{\Longrightarrow}$ otherwise. □

For strong barbed bisimilarity, the comparison of process behaviour is performed over single transition steps. (Weak) barbed bisimilarity, defined below, employs weak transition which collapses any number, possibly zero, of invisible actions onto a single visible action. Internal action must similarly be abstracted away from the notion of observability in order to define behavioural equivalence under this abstraction.

Definition 2.4.9 (weak barbs) $\Downarrow_\mu$ is $\Longrightarrow\downarrow_\mu$. □

Weak versions of the strong bisimilarities can now be defined by adjusting the rules that determine what is required of a process to successfully match the actions of another. Due to the nature of weak transition, it can intuitively be seen that showing weak bisimilarity will require more work than its strong counterpart as there will, in general, be more computation paths to be explored. However, the symmetric and recursive nature of bisimulation allow the same relation to be obtained when the abstracted versions of transition and observability are restricted to one side of a bisimulation [8]. In other words, matching $\longrightarrow$ with $\Longrightarrow$ generates the same relation as matching $\Longrightarrow$ with $\Longrightarrow$, but will require less work to show.

Definition 2.4.10 (barbed bisimilarity) *Barbed bisimilarity* is the bisimilarity relation on processes, $\dot{\approx}$, such that whenever $P \dot{\approx} Q$,

(1) $P\downarrow_\mu$ implies $Q\Downarrow_\mu$
(2) $P \xrightarrow{\tau} P'$ implies $Q \Longrightarrow \dot{\approx} P'$ □

Barbed bisimilarity does allow some abstraction from internal process activity, but, like strong barbed bisimilarity, suffers from the inability to distinguish between the contents of sent messages. For example, $(\boldsymbol{\nu}\, x)\,(\overline{x}u \mid xz.\overline{z}y) \;\dot{\approx}\; \overline{u}v$. Thus, a stronger relation is once again formed by taking the largest congruence from within barbed bisimilarity; this relation, barbed congruence, is often applied as the main behavioural equivalence on π-calculus processes.

Definition 2.4.11 (barbed congruence) Processes P and Q are *barbed congruent*, $P \dot{\cong} Q$, if $\mathcal{C}[P] \dot{\approx} \mathcal{C}[Q]$ for every context $\mathcal{C}$. □

Unfortunately, it is often difficult to work directly with weak (and strong) barbed congruence from their definitions. The requirement of equivalence under all contexts makes barbed bisimulation proof in the general case a demanding procedure. However, certain cases are more amenable. For example, to prove $(\boldsymbol{\nu}\, x)\,(x.P) \dot{\cong} \mathbf{0}$ it can be observed that neither process has any barbs or reductions. Then by induction over the asynchronous π-operators, and from the base cases of $\mathbf{0}$ and $\overline{u}y$ for all u and y, $\mathcal{C}[(\boldsymbol{\nu}\, x)\,(x.P)] \dot{\cong} \mathcal{C}[\mathbf{0}]$ can be proved for all $\mathcal{C}$.

Proposition 2.4.12 $(\equiv \subseteq \dot{\cong})$
Proof We can show that $\equiv$ implies $\dot{\approx}$, and then the desired result follows from the fact that $\equiv$ is a congruence, and that $\dot{\cong}$ is the largest congruence in $\dot{\approx}$. Alternatively, we can derive the result from the $\equiv$-based definition of weak output barbs, the (R-STRUCT) rule in the definition of $\longrightarrow$, and the fact that $\equiv$ is a congruence. □

Proposition 2.4.12, which shows $\equiv$ is more restrictive than, and respects $\dot{\cong}$, allows equivalences to be proved from the axioms of $\equiv$ using the congruence and transitivity properties of $\dot{\cong}$ [5]. For example,

$$\overline{x} \equiv \overline{x} \mid \mathbf{0} \dot{\cong} \overline{x} \mid (\boldsymbol{\nu}\, z)\,\mathbf{0} \dot{\cong} (\boldsymbol{\nu}\, z)\,(\overline{x} \mid \mathbf{0}) \dot{\cong} \ldots$$

Showing that two processes are not barbed congruent is typically an easier task; as demonstrated previously for reduction and (strong) barbed congruence, it is enough to identify a single context that can distinguish the processes. Still, it is useful to have other indirect but more tractable techniques for showing barbed congruence. Firstly, we consider (weak) bisimilarity, which implies, and is strictly contained by, barbed bisimilarity [8]. Bisimilarity and can be proved without needing to quantify over all contexts; its nature is a departure from the reduction-based equivalences seen so far.

Definition 2.4.13 (bisimilarity) *Bisimilarity* is the bisimilarity relation on processes, $\approx$, such that whenever $P \approx Q$, $P \xrightarrow{l} P'$ implies $Q \xRightarrow{\hat{l}} \approx P'$. □

Bisimilarity is a labelled equivalence: it uses the full expressiveness of the transition relations to describe process equivalence through direct comparison of actions. Bisimilarity requires less work to show than barbed congruence, which effectively observes the behaviour of processes when embedded into all possible system contexts. However, the (IN_s) and (OPEN) transition rules implicitly capture some of the power of explicit context comparison by allowing any name to be received by input, and any fresh name to be sent by bound-output actions. Omitted from this report is the definition of the relation $\sim$, the strong version of $\approx$, which is strictly contained within $\approx$ (and thus could be used to show $\approx$).

Bisimilarity has an implicit caveat that a bound-output action of one process need only be matched by the other if the bound-output object is not free in either process. This is a consequence of asserting that bound-output objects must be fresh, as enforced by the side-conditions of (PAR-L) and (PAR-R) (see Section 2.3). This restriction is a minor issue as the bound-output object can be almost any name, and by convention, freshness is extended to encompass all processes and entities in consideration.

For the full π-calculus, bisimilarity is not preserved by arbitrary substitution but is a non-input congruence, that is, a congruence quantified over all contexts for which the hole does not occur under an input prefix [8]. This is because new possibilities for intraaction may be created when certain free names are received. For example, taking the full π-calculus processes $P \stackrel{\text{def}}{=} \overline{z} \mid w$ and $Q \stackrel{\text{def}}{=} \overline{z}.w + w.\overline{z}$, then we have $P \approx Q$, but $xz.P \not\approx xz.Q$. However, the substitution-closure of bisimilarity, full bisimilarity, is the largest congruence included in bisimilarity, and implies $\dot{\cong}$ [8].

Proposition 2.4.14 ($\equiv \subseteq \approx$)

Proof By the fact that if $P \equiv Q$ and $P \stackrel{l}{\longrightarrow} P'$, then for some Q' we have $Q \stackrel{l}{\longrightarrow} Q'$ and $P' \equiv Q'$ [8], it follows from the definitions of $\Longrightarrow$ and $\approx$ that $\equiv$ is a bisimulation, and thus the desired result is obtained. □

This result can be used to conveniently shorten bisimulation proofs when applicable (see Example 2.4.17), and corresponds to the fact that replacing the transition rule (ALPHA) with

$$\text{(T-STRUCT)} \quad \frac{P \equiv Q \stackrel{l}{\longrightarrow} Q' \equiv P'}{P \stackrel{l}{\longrightarrow} P'}$$

would yield the same relation, namely $\approx$. This modification would make (REP) redundant, but bisimilarity proofs would require working up to structural congruence for each transition. As previously mentioned, defining the reduction-based bisimilarities in terms of reduction rather than τ-transition would have similar consequences.

There are actually several variants of bisimilarity, each with its own particular use. The late transition relations, discussed in Section 2.3, lead naturally to late bisimilarity, a very similar relation to bisimilarity but with stronger requirements for matching input actions. Open bisimilarity, unlike early and late bisimilarity, is by definition a congruence, and is more demanding than the other variants but has the advantages that no explicit or implicit quantification over input names or substitutions is involved. In fact, each of these variants of bisimilarity can be expressed using either of the early and the late transition relations, but for late bisimilarity the latter are more convenient [8].

For the asynchronous π-calculus, however, bisimilarity and the variants mentioned above all coincide, and the common relation is a congruence preserved by all contexts of

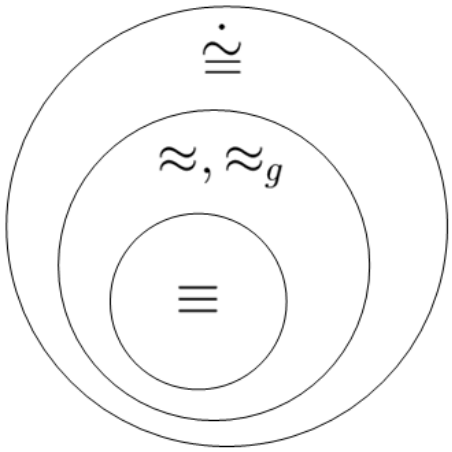

Figure 2.1: The relationship between $\dot{\cong}$, $\approx$, $\approx_g$ and $\equiv$ on the asynchronous π-calculus.

the full π-calculus that implies barbed congruence [8]. Furthermore, an additional variant, ground bisimilarity, also coincides with the common congruence. This is a useful result as ground bisimilarity requires less work to show than bisimilarity for the following reason. (IN_s) expresses that any name can be received by an input action, and thus to show bisimilarity between P and Q, $P \xrightarrow{xy} P'$ must be matched with a $Q \xrightarrow{xy} Q'$ for each name y. For ground bisimilarity, it is enough to consider a single fresh name as the object of an input or bound-output action.

As a consequence, ground bisimilarity is less discriminating than bisimilarity, and is unsatisfactory as a process equivalence for the full π-calculus. The main reason for this, as with barbed bisimilarity and bisimilarity, is that all names are open to instantiation to any name. The strength of ground bisimilarity in the asynchronous π-calculus stems from the fact that an output cannot preclude any other action nor enable further action due to the absence of summation and output continuations.

Definition 2.4.15 (ground bisimilarity) *Ground bisimilarity* is the bisimilarity relation, $\approx_g$, such that whenever $P \approx_g Q$, there is $z \notin \mathsf{fn}(P) \cup \mathsf{fn}(Q)$ such that if $P \xrightarrow{l} P'$ where $l \in \{xz, \overline{x}y, \overline{x}(z), \tau\}$, then $Q \stackrel{\hat{l}}{\Longrightarrow} \approx_g P'$. □

Lemma 2.4.16 (Properties of $\approx_g$ for the asynchronous π-calculus)

(i) $\approx_g$ is a congruence on the asynchronous π-calculus.

(ii) $\approx_g \subseteq \dot{\cong}$.

(iii) $\approx_g = \approx$.

These results are cited from [8]. □

Thus, to show barbed congruence between asynchronous π-processes, it is enough to exhibit a ground bisimulation containing them. The coincidence of bisimilarity and ground bisimilarity on the asynchronous π-calculus leads to the symbols $\approx$ and $\approx_g$ being interchanged freely within this report when referring to the common relation. Figure 2.1 illustrates the relationship between the most important, with regards to this report, of the

behavioural equivalences (all of which are congruences) defined in this chapter, namely $\dot{\cong}$, $\approx$, $\approx_g$ and $\equiv$.

Example 2.4.17 Let $L \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\overline{x}y \mid xz.\overline{z})$ and $R \stackrel{\text{def}}{=} \overline{y}$.

1. We first prove $L \approx R$ by constructing a bisimulation, $\mathcal{S}$, containing (L, R).
   (1) $(L, R) \in \mathcal{S}$
   - $L \xrightarrow{\tau} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \overline{y})$ matched by $R \Longrightarrow R$
   - $R \xrightarrow{\overline{y}} R' \stackrel{\text{def}}{=} \mathbf{0}$ matched by $L \stackrel{\overline{y}}{\Longrightarrow} L'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \mathbf{0})$

   (2) $(L', R) \in \mathcal{S}$
   - $L' \xrightarrow{\overline{y}} L''$ matched by $R \xrightarrow{\overline{y}} R'$

   (3) $(R', L'') \in \mathcal{S}$ and neither has any transitions.

   $\mathcal{S} = \{(L, R), ((\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \overline{y}), R), (\mathbf{0}, (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \mathbf{0}))\}$ is a bisimulation, and therefore $L \approx R$.

2. We now illustrate the application of Definition 2.4.1.
   (1) Check $L \approx R$.
   - $L \xrightarrow{\tau} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \overline{y})$ matched by $R \Longrightarrow R$
   - $R \xrightarrow{\overline{y}} R' \stackrel{\text{def}}{=} \mathbf{0}$ matched by $L \stackrel{\overline{y}}{\Longrightarrow} L'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \mathbf{0})$

   (2) Check $L' \approx R$.
   - $L' \xrightarrow{\overline{y}} L''$ matched by $R \xrightarrow{\overline{y}} R'$

   (3) $R' \approx L''$ as neither has any transitions.

   Thus, we have the desired result.

3. We can now show how Proposition 2.4.14 can be used to shorten proofs.
   (1) Check $L \approx R$.
   - $L \xrightarrow{\tau} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \overline{y})$ matched by $R \Longrightarrow R$
   - $R \xrightarrow{\overline{y}} R' \stackrel{\text{def}}{=} \mathbf{0}$ matched by $L \stackrel{\overline{y}}{\Longrightarrow} L'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x)\,(\mathbf{0} \mid \mathbf{0})$

   (2) $L' \approx R$ as $L' \equiv R$.
   (3) $R' \approx L''$ as $R' \equiv L''$.

   Furthermore, to show $P \approx Q$, it would suffice to show $P \equiv P' \approx Q' \equiv Q$ due to the transitivity of $\approx$. □

2.5 Expressiveness

This section discusses some issues relating to the expressiveness of the monadic asynchronous π-calculus as presented in this chapter. It is of both practical and theoretical interest

to determine what a particular system is capable of expressing, and in particular how different combinations of operators affect expressiveness. The analysis of expressiveness is the primary motivation for the **cc**, and its main application.

2.5.1 Polyadicity

In the polyadic π-calculus, tuples of arbitrary length may be instantaneously communicated. Simple arity-matching can be used to control communication, but more generally, some form of typing system is required to determine whether a polyadic process is well formed [8].

However, polyadic π-terms can be expressed in the monadic asynchronous π-calculus; the translation is a homomorphism, except for the input prefix and output action which are extended to follow a monadic protocol for (synchronously) receiving and sending multiple names respectively.

$$\begin{aligned} [\![x(z_1,...,z_n).P]\!] &\stackrel{\text{def}}{=} xs.(\boldsymbol{\nu}\, r)\,(\overline{s}r \mid rz_1.(... \mid (\overline{s}r \mid rz_n.P)...)) \\ [\![\overline{x}\langle y_1,...,y_n\rangle]\!] &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, s)\,(\overline{x}s \mid sr.(\overline{r}y_1 \mid sr.(... \mid sr.(\overline{r}y_n)...))) \end{aligned}$$

Firstly, the sender and receiver exchange fresh names w and v to create two private channels between them. The sender then uses the receiver's private channel to transmit each element of the tuple, $y_1,...,y_n$, to be acknowledged in turn by the receiver over the sender's channel. Input prefixing ensures that the sender only sends each element after the previous has been acknowledged, and that the receiver only acknowledges elements that it has received. Thus, the order of elements within the tuple is preserved.

This mapping is sound, but not complete. For example, taking $P \stackrel{\text{def}}{=} xz.xz.\mathbf{0}$ and $Q \stackrel{\text{def}}{=} xz.\mathbf{0} \,|\, xz.\mathbf{0}$, we have $P \approx Q$, but $[\![P]\!] \not\approx [\![Q]\!]$. In $[\![P]\!]$, the two instances of the protocol are serialised, but in $[\![Q]\!]$, they may be run concurrently.

2.5.2 Synchronised Communication

The preceding example illustrates how a communication feature, in this case polyadicity, that is not directly supported can be emulated through a suitable encoding. The multiple name passing protocol can be extended to perform general synchronised communication by emulating output continuations. Firstly, we use the following mapping to encode output continuations in the polyadic asynchronous π-calculus.

$$\begin{aligned} [\![x\tilde{z}.P]\!] &\stackrel{\text{def}}{=} x(\tilde{z},s).([\![P]\!] \mid \overline{s}) \\ [\![\overline{x}\tilde{y}.P]\!] &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, s)\,(\overline{x}\langle\tilde{y},s\rangle \mid s.[\![P]\!]) \end{aligned}$$

This encoding describes a standard protocol for performing synchronised interaction using a pair of asynchronous communications: the sender sends a fresh name with the message, and waits for an acknowledgement signal from the receiver via the fresh name before proceeding. This protocol can then be combined with the protocol in the preceding section to give a full encoding of synchronous, multiple name passing in the monadic asynchronous π-calculus. Again, the resulting mapping is sound, but not complete.

2.6 Asynchronous Bisimilarity

The main behavioural equivalences discussed in Section 2.4 are based on the synchronous early transition relations defined in Section 2.3. As discussed previously, alternative transition relations can be used to establish behavioural equivalences with differing characteristics. Following is a definition of asynchronous (early) transition relations.

Definition 2.6.1 (asynchronous early transition [9]) The *asynchronous early transition relations* for the asynchronous π-calculus are the smallest relations defined by (T-STRUCT) in Section 2.4 and (OUT), (PAR-L), (PAR-R), (REP), (RES), (OPEN) given in Table 2.4 with $\xrightarrow{l}$ replaced with $\xrightarrow{l}_a$, along with the following (IN_a) and (TAU) rules.

$$(\text{IN}_a)\quad \mathbf{0} \xrightarrow{xy}_a \overline{x}y \qquad (\text{TAU})\quad \frac{P \xrightarrow{\tau} P'}{P \xrightarrow{\tau}_a P'}$$

Asynchronous bisimilarity, $\approx_a$, is defined by replacing $\longrightarrow$ and $\Longrightarrow$ in Definition 2.4.1 with $\longrightarrow_a$ and $\Longrightarrow_a$ respectively, where $\Longrightarrow_a$ is defined as expected. □

Inaction, which has no transitions under the synchronous transition relations, is now capable of performing asynchronous input (and so is no longer appropriately named). Whereas output asynchronicity is emulated through syntax restriction, input asynchronicity is expressed through the semantics of the new input action: the act of receiving a message has been decoupled from the use of the message. The latter is captured by the primitive input prefix abstraction, and thus this construct cannot be dropped from process syntax in the manner of output prefixing.

The asynchronous transition relations can actually be viewed as having a "late" quality as the instantiation of received names within a system is delayed from the actual arrival of a message in the system. Consequently, a system can no longer directly synchronise with its environment, and the consumption of a received message cannot be observed.

Proving bisimilarity under asynchronous transition requires finding an appropriate closure that respects the capability of a system to receive any message at any point of execution. Asynchronous bisimilarity naturally satisfies the asynchrony law, $\mathbf{I}\langle x\rangle \approx_a \mathbf{0}$. This result captures the essence of asynchronous communication, showing that it is not possible to observe the presence of a communication buffer. $\mathbf{I}\langle x\rangle \not\approx \mathbf{0}$, as $\mathbf{0}$ is the inaction, but the law would be satisfied by asynchronous barbed congruence (under synchronous transition) as communicating with the buffer simply returns the sent message; an external observer cannot detect that anything has happened.

Chapter 3

Concurrent Combinators

The **cc** was introduced in Section 1.2 and is now examined in greater detail. Firstly, the **cc** itself is presented formally. It is then shown that the **cc** can be precisely embedded into the asynchronous π-calculus as the $\mathbf{cc}_\pi$. Finally, it is shown that the asynchronous π-calculus, as presented in Chapter 2, and the $\mathbf{cc}_\pi$ have the same expressive power up to (weak) bisimilarity: every asynchronous π-calculus process has a bisimilar $\mathbf{cc}_\pi$ representation.

Throughout this chapter, relations will often be decorated by their respective calculus to avoid confusion, for example to distinguish $\longrightarrow_\pi$ from $\longrightarrow_{\mathbf{cc}}$.

3.1 The cc [2]

Definition 3.1.1 (cc-atoms) The set of atoms in the **cc** is defined to be

$$\mathbb{A}_{\mathbf{cc}} \stackrel{\text{def}}{=} \{\mathcal{M}, \mathcal{D}, \mathcal{K}, \mathcal{FW}, \mathcal{B}_l, \mathcal{B}_r, \mathcal{S}\}$$

and is ranged over by $\mathcal{C}, \mathcal{C}', \ldots$. The *arity* of an atom is given by the function $ar : \mathbb{A}_{\mathbf{cc}} \to \{1, 2, 3\}$ where

$$ar(\mathcal{C}) = \begin{cases} 3, & \text{if } \mathcal{C} \in \{\mathcal{D}, \mathcal{S}\} \\ 2, & \text{if } \mathcal{C} \in \{\mathcal{M}, \mathcal{FW}, \mathcal{B}_l, \mathcal{B}_r\} \\ 1, & \text{if } \mathcal{C} \in \{\mathcal{K}\} \ . \end{cases}$$

□

Definition 3.1.2 (atomic agents) An *atomic agent* has the form $\mathcal{C}(\tilde{v}^n)$ where $\mathcal{C} \in \mathbb{A}_{\mathbf{cc}}$, $n = ar(\mathcal{C})$, and $\tilde{v}^n$ is a sequence of identifier parameters of length n which range over names $a, b, c, \ldots$. Each parameter of an atomic agent has an associated *polarity* ranging over $\{+, -, \pm\}$. The set $\mathbf{A}_{\mathbf{cc}}$ of **cc** atomic agents and the polarities of their parameters are

$$\mathbf{A}_{\mathbf{cc}} \stackrel{\text{def}}{=} \{\mathcal{M}(x^+y^\pm),\ \mathcal{FW}(x^-y^+),\ \mathcal{D}(x^-y^+z^+),\ \mathcal{K}(x^-),\ \mathcal{B}_l(x^-y^+),\ \mathcal{B}_r(x^-y^-),\ \mathcal{S}(x^-y^-z^+)\} \ .$$

The *subject* of an atomic agent is the identifier specified in the first (leftmost) parameter position; the other identifiers, if any, are called *objects*. The subject polarity of an atomic agent is referred to as the polarity of the atomic agent. □

$$\mathcal{D}(x^- u^+ v^+) \mid \mathcal{M}(x^+ y) \longrightarrow \mathcal{M}(u^+ y) \mid \mathcal{M}(u^+ y) \qquad \mathcal{K}(x^-) \mid \mathcal{M}(x^+ y) \longrightarrow \mathbf{0}$$
$$\mathcal{FW}(x^- u^+) \mid \mathcal{M}(x^+ y) \longrightarrow \mathcal{M}(u^+ y) \qquad \mathcal{B}_l(x^- u^+) \mid \mathcal{M}(x^+ y) \longrightarrow \mathcal{FW}(y^- u^+)$$
$$\mathcal{B}_r(x^- u^-) \mid \mathcal{M}(x^+ y) \longrightarrow \mathcal{FW}(u^- y^+) \qquad \mathcal{S}(x^- u^- v^+) \mid \mathcal{M}(x^+ y) \longrightarrow \mathcal{FW}(u^- v^+)$$

Table 3.1: Reduction rules for **cc** terms.

As described in Section 1.2, the parameters of an atomic agent can be seen as bindings between the interaction ports of an atom to outside names. The number of ports of an atom is given by its arity, and the interactive behaviour of an atom is coupled to its subject polarity. The neutral object polarity, $\pm$, of $\mathcal{M}(x^+ y^\pm)$ will generally be omitted.

Definition 3.1.3 (cc-terms) The set of terms, also called *agents*, that can be expressed in the **cc**, $P_{\mathbf{cc}}$, and ranged over by $P, Q, R, ...$ is defined by the following grammar

$$P_{\mathbf{cc}} \quad ::= \quad \mathcal{C}(\tilde{v}^n) \quad | \quad P \mid Q \quad | \quad (\boldsymbol{\nu}\, z)\, P \quad | \quad !\, P \quad | \quad \mathbf{0}$$

where $\mathcal{C}(\tilde{v}^n) \in \mathbf{A}_{\mathbf{cc}}$. In each of $(\boldsymbol{\nu}\, z)\, P$, occurrences of the name z are *bound* within the scope of P. An occurrence of a name is *free* if it is not bound. Again, $\mathsf{bn}(P)$ and $\mathsf{fn}(P)$ are used to denote the bound and free names of P respectively. Substitution and α-equivalence for **cc**-terms are defined as for the π-calculus in Definition 2.1.2, appropriately replacing the references to processes with **cc**-terms, and replacing (iii) with

(iii) *α-conversion* of a **cc**-term is the replacement of a subterm $(\boldsymbol{\nu}\, z)\, P$ by $(\boldsymbol{\nu}\, w)\, P\{^w/_z\}$ where w does not already occur in P. □

The **cc**-operators are analogous to their counterparts in the π-calculus, and an interpretation of parallel composition, name hiding and replication as applied to the **cc** was given in Section 1.2. Unlike the π-calculus, the absence of any prefixing constructs means that restriction is the sole source of name binding.

Definition 3.1.4 (cc structural congruence and reduction) The *structural congruence* relation $\equiv_{cc}$ is defined over **cc**-terms by the same rules as in the π-calculus (Table 2.2). The *reduction* relation on cc-terms, $\longrightarrow_{\mathbf{cc}}$, is the smallest relation generated by the rules in Table 3.1 along with (R-PAR), (R-RES) and (STR) in Table 2.3. □

Thus, two atomic agents can be reduced if they have opposing subject polarities, and there is one and only one rule for each such case. Combinator polarities are preserved by reduction, and the free identifiers of the resulting term are a subset of the free identifiers of the reagents. An understanding of **cc**-term behaviour as specified by these fixed dyadic interaction rules was described in Section 1.2.

Example 3.1.5 Synchronous multiple name passing (see Section 2.5.1) is demonstrated using **cc** atomic agents [2]. Assume the following agents.

$$\begin{aligned}
\mathcal{SW}(xy) &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\, (\mathcal{B}_r(xz) \mid \mathcal{M}(zy)) \\
\mathcal{SW}'(xyuv) &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, w, z)\, (\mathcal{D}(xwz) \mid \mathcal{SW}(w)y \mid \mathcal{S}(zuv)) \\
\mathcal{R}(u_1 u_2 v_1 v_2 y) &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, x, w, z)\, (\mathcal{D}(u_1 u_2 w) \mid \mathcal{D}(v_1 v_2 z) \mid \mathcal{S}(zwx) \mid \mathcal{SW}(xy))
\end{aligned}$$

$\mathcal{SW}(xy)$ is the *Switcher*, which has the behaviour

$$\mathcal{SW}(xy) \mid \mathcal{M}(xu) \longrightarrow^* \mathcal{M}(uy) \ ,$$

and $\mathcal{SW}'(xyuv)$ is an augmented Switcher that creates an additional $\mathcal{FW}(uv)$ after switching out y. The agent $\mathcal{R}(u_1u_2v_1v_2y)$ firstly relays the names received over u_1 and v_1 to u_2 and v_2 respectively, and then sends y to the name received over u_1. We now define

$$\begin{aligned} src(v_1,...,v_n) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_1, ..., c_n)\, (\mathcal{M}(ac) &\mid \mathcal{FW}(cc_1) \\ &\mid \mathcal{SW}'(c_1v_1cc_2) \mid ... \mid \mathcal{SW}'(c_{n-1}v_{n-1}cc_n) \\ &\mid \mathcal{SW}(c_nv_n)) \end{aligned}$$

and

$$\begin{aligned} rcv(u_1,...,u_n) \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, p_0, p_1, ..., p_n, q_1, ..., q_n) & \\ (\mathcal{D}(ap_0p_1) &\mid \mathcal{SW}(p_0q_1) \\ &\mid \mathcal{R}(p_1p_2q_1u_1q_2) \mid ... \mid \mathcal{R}(p_{n-1}p_nq_{n-1}u_{n-1}q_n) \\ &\mid \mathcal{S}(p_nq_nu_n)) \ , \end{aligned}$$

and trace the reduction of $src(v_1,...,v_n) \mid rcv(u_1,...,u_n)$.

$$\begin{aligned} & src(v_1,...,v_n) \mid rcv(u_1,...,u_n) \\ & \quad \longrightarrow \quad (\boldsymbol{\nu}\, ...\,)\, (\mathcal{M}(p_0c) \mid \mathcal{M}(p_1c) \mid \mathcal{SW}(p_0q_1) \mid ...\,) \\ & \quad \longrightarrow \quad (\boldsymbol{\nu}\, ...\,)\, (\mathcal{M}(p_1c) \mid \mathcal{M}(cq_1) \mid \mathcal{FW}(cc_1) \mid ...\,) \\ & \quad \longrightarrow \quad (\boldsymbol{\nu}\, ...\,)\, (\mathcal{M}(p_1c) \mid \mathcal{M}(c_1q_1) \mid \mathcal{SW}'(c_1v_1cc_2) \mid ...\,) \\ & \quad \longrightarrow \quad (\boldsymbol{\nu}\, ...\,)\, (\mathcal{M}(p_1c) \mid \mathcal{M}(q_1v_1) \mid \mathcal{R}(p_1p_2q_1u_1q_2) \mid \mathcal{FW}(cc_2) \mid ...\,) \\ & \quad \longrightarrow \quad \mathcal{M}(u_1v_1) \mid (\boldsymbol{\nu}\, ...\,)\, (\mathcal{M}(p_2c) \mid \mathcal{M}(cq_2) \mid \mathcal{FW}(cc_2) \mid ...\,) \\ & \quad ... \\ & \quad \longrightarrow \quad \mathcal{M}(u_1v_1) \mid \mathcal{M}(u_2v_2) \mid ... \end{aligned}$$

As the reduction progresses, *src* will continue to send the remaining names $v_3, ..., v_n$ to *rcv* in the correct order. Interference is prevented due to c being created as a private channel. □

3.2 Embedding the cc within the π-calculus [2]

We now embed the **cc** within the π-calculus.

Definition 3.2.1 ($\mathbf{cc}_\pi$-atoms [9]) The $\mathbf{cc}_\pi$-atoms comprise the set $\mathbf{A}_\pi$ of π-processes named and annotated as follows.

$$\mathbf{m}(x^+y^\pm) \stackrel{\text{def}}{=} \overline{x}y \quad \mathbf{d}(x^-u^+v^+) \stackrel{\text{def}}{=} xz.(\mathbf{m}\langle uz\rangle \mid \mathbf{m}\langle vz\rangle) \quad \mathbf{k}(x^-) \stackrel{\text{def}}{=} xz.\mathbf{0} \quad \mathbf{fw}(x^-u^+) \stackrel{\text{def}}{=} xz.\mathbf{m}\langle uz\rangle$$

$$\mathbf{b}_l(x^-u^+) \stackrel{\text{def}}{=} xz.\mathbf{fw}\langle xu\rangle \quad \mathbf{b}_r(x^-u^-) \stackrel{\text{def}}{=} xz.\mathbf{fw}\langle uz\rangle \quad \mathbf{s}(x^-u^-v^+) \stackrel{\text{def}}{=} xz.\mathbf{fw}\langle uv\rangle$$

An *atomic agent* is an instance of one of these atoms. □

$$[\![P \mid Q]\!]_\pi \stackrel{\text{def}}{=} [\![P]\!]_\pi \mid [\![Q]\!]_\pi \qquad [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\, [\![P]\!]_\pi \qquad [\![!\, P]\!]_\pi \stackrel{\text{def}}{=} !\, [\![P]\!]_\pi$$

$$[\![\mathcal{C}(\tilde{v})]\!]_\pi \stackrel{\text{def}}{=} \mathbf{c}\langle\tilde{v}\rangle \qquad [\![\mathbf{0}]\!]_\pi \stackrel{\text{def}}{=} \mathbf{0}$$

Table 3.2: Translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$.

The parameter annotations $\{-, +\}$, as for **cc**-atoms, explicitly identify whether the named process will perform an input or output action on the annotated name. Again, the $\pm$ annotation signifies that a parameter will only be used passively by the immediate process, that is, the process can perform a send action which will transmit the annotated name. Being members of P_π, the $\mathbf{cc}_\pi$-atoms are of course encompassed by the $\longrightarrow_\pi$ and $\approx$ relations. The $\mathbf{cc}_\pi$-atoms are defined in accordance with the **cc** so that reduction can only occur between $\mathbf{m}\langle xy\rangle$ and one of the other atoms; this interaction is atomic in the sense that reduction will always result in one or more new atoms, with the exception of $\mathbf{m}\langle xy\rangle \mid \mathbf{k}\langle x\rangle \longrightarrow \mathbf{0}$.

Definition 3.2.2 (the $\mathbf{cc}_\pi$) The $\mathbf{cc}_\pi$ is the subcalculus of the π-calculus whose processes ranged over by $P, Q, R, ...$ are given by

$$P_{\mathbf{cc}_\pi} \quad ::= \quad \mathbf{c}\langle\tilde{v}\rangle \quad | \quad P \mid Q \quad | \quad (\boldsymbol{\nu}\, z)\, P \quad | \quad !\, P \quad | \quad \mathbf{0}$$

where $\mathbf{c}\langle\tilde{v}\rangle \in \mathbf{A}_\pi$ and $\tilde{v}$ is a sequence of names of length equal to the notational arity of atom **c**. □

Mapping of **cc**-terms to $\mathbf{cc}_\pi$-terms is straightforward.

Definition 3.2.3 (translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$) The mapping $[\![\cdot]\!]_\pi : P_{\mathbf{cc}} \to P_{\mathbf{cc}_\pi}$ is given by the rules in Table 3.2. The rule $[\![\mathcal{C}(\tilde{v})]\!]_\pi \stackrel{\text{def}}{=} \mathbf{c}\langle\tilde{v}\rangle$ describes the literal correspondence between **cc** atomic agents and $\mathbf{cc}_\pi$-atoms of the same name. □

Proposition 3.2.4 (syntactic and operational correspondence) The translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$ by the function $[\![\cdot]\!]_\pi$ exhibits the following properties for all $P, Q \in P_{\mathbf{cc}}$.

(i) $\mathsf{fn}(P) = \mathsf{fn}([\![P]\!]_\pi)$.
(ii) For any substitution σ, $[\![P\sigma]\!]_\pi \equiv [\![P]\!]_\pi\sigma$.
(iii) $P \equiv_{\mathbf{cc}} Q \Rightarrow [\![P]\!]_\pi \equiv_\pi [\![Q]\!]_\pi$.
(iv) $P \longrightarrow_{\mathbf{cc}} Q \Rightarrow [\![P]\!]_\pi \longrightarrow_\pi [\![Q]\!]_\pi$.

Proof The general approach of the required proofs firstly involve case by case verification of the proposition for each **cc** atomic agent or atomic reduction step. The inductive hypothesis is then applied over the structure of **cc** terms or relevant relation rules. Further details are given in Appendix A.1. □

These results show that $[\![\cdot]\!]_\pi$ exactly preserves the behaviour of **cc**-terms when mapped to their corresponding $\mathbf{cc}_\pi$ representations. A useful relation for $\mathbf{cc}_\pi$-terms is β-equality which requires the notion of pointedness.

Definition 3.2.5 (active names and pointedness)

(i) For a process P, the set of *output active names* is given by $\mathsf{an}_{\uparrow}(P) \stackrel{\text{def}}{=} \{x \mid P \downarrow_{\overline{x}}\}$, and the set of *input active names* is given by $\mathsf{an}_{\downarrow}(P) \stackrel{\text{def}}{=} \{x \mid P \downarrow_{x}\}$.

(ii) For $P \in P_{\mathbf{cc}_\pi}$, P is $x^{\uparrow}$*-pointed*, $P_{\langle x^{\uparrow}\rangle}$, iff

(1) $\mathsf{an}_{\uparrow}(P) = \{x\}$ and $\mathsf{an}_{\downarrow}(P) = \emptyset$

(2) $\forall \mathbf{c}\langle x^{-}\tilde{v}\rangle : (P \mid \mathbf{c}\langle x^{-}\tilde{v}\rangle \longrightarrow Q_1) \wedge (P \mid \mathbf{c}\langle x^{-}\tilde{v}\rangle \longrightarrow Q_2) \Rightarrow Q_1 \equiv Q_2$

(3) $P \not\longrightarrow$.

$P_{\langle x^{\downarrow}\rangle}$ is defined by replacing $\uparrow$ and $+$ with $\downarrow$ and $-$ respectively. □

Pointedness indicates that a $\mathbf{cc}_\pi$-term is immediately capable of interacting over the specified name, and over that name only. For example, $\mathbf{m}\langle xy\rangle$ has $\mathsf{an}_{\uparrow}(P) = \{x\}$, and all P such that $P_{\langle x^{\uparrow}\rangle}$ must contain exactly one such subterm in which the occurrence of x is free.

Definition 3.2.6 (β-equality [9]) *β-reduction*, $\longrightarrow_\beta$, is defined by the rule

$$(\text{COMM}_\beta) \quad (\boldsymbol{\nu}\, x)\,(P_{\langle x^{\uparrow}\rangle} \mid Q_{\langle x^{\downarrow}\rangle}) \longrightarrow_\beta (\boldsymbol{\nu}\, x)\, R$$

if $P_{\langle x^{\uparrow}\rangle} \mid Q_{\langle x^{\downarrow}\rangle} \longrightarrow_\beta R$, together with (R-PAR), (R-RES) and (STR) in Table 2.3. *β-equality*, $=_\beta$, is the symmetric and transitive closure of $\longrightarrow_\beta^{*} \cup \equiv$. □

Proposition 3.2.7 ($=_\beta \subseteq \approx$ [9])

Proof Take $\mathcal{R} \stackrel{\text{def}}{=} \{(P, Q) \mid P \longrightarrow_\beta Q\} \cup \equiv$ and show it is a bisimulation [9]. □

$\longrightarrow_\beta$ is the subset of $\longrightarrow$ where the interacting components exhibit complementary pointedness of the same name; due to the restriction of this name, β-reduction describes both the single capability for action of each component and the sole possible outcome of the subsequent interaction. It can be intuitively seen that $=_\beta$ implies $\approx$ by considering the possible transitions that have to be matched by each process in a bisimulation containing them, and from the fact that $\equiv$ respects pointedness. β-reduction is useful when working to show bisimilarity between $\mathbf{cc}_\pi$ terms because atomic agents, when under the necessary restriction, always fulfil such one-to-one pointedness.

Example 3.2.8 Consider again the Switcher, $\mathcal{SW}(xy)$, defined in Example 3.1.5. Intuitively, one would define a Switcher for the π-calculus by

$$\mathbf{sw}(xy) \stackrel{\text{def}}{=} xz.\overline{z}y\ .$$

We show that $\mathbf{sw}\langle xy\rangle \approx [\![\mathcal{SW}(xy)]\!]_\pi$ as an illustration of the **cc** to π-calculus mapping and its behaviour preserving properties. β-reduction is demonstrated as part of the proof.

$$\begin{aligned}
[\![\mathcal{SW}(xy)]\!]_\pi &\stackrel{\text{def}}{=} [\![(\boldsymbol{\nu}\, z)\,(\mathcal{B}_r(xz) \mid \mathcal{M}(zy))]\!]_\pi && \text{(Example 3.1.5)}\\
&= (\boldsymbol{\nu}\, z)\,[\![(\mathcal{B}_r(xz) \mid \mathcal{M}(zy))]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\,([\![\mathcal{B}_r(xz)]\!]_\pi \mid [\![\mathcal{M}(zy)]\!]_\pi) && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\,(\mathbf{b}_r\langle xz\rangle \mid \mathbf{m}\langle zy\rangle) && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\,(xv.zw.\overline{v}w \mid \overline{z}y) && \text{(Definition 3.2.1)}
\end{aligned}$$

Let $L \stackrel{\text{def}}{=} xz.\overline{z}y$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\,(xv.zw.\overline{v}w \mid \overline{z}y)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xu} L' \stackrel{\text{def}}{=} \overline{u}y$ matched by $R \xrightarrow{xu} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\,(zw.\overline{u}w \mid \overline{z}y)$

(2) Check $L' \approx R'$

$$\begin{aligned} R' &\longrightarrow_\beta (\boldsymbol{\nu}\, z)\,\overline{u}y && \text{(Definition 3.2.6)} \\ &\approx L' && \text{(Proposition 2.4.14, Proposition 3.2.7)} \end{aligned}$$

Also, it is easy to see that

$$\begin{aligned} \mathcal{SW}(xy) \mid \mathcal{M}(xa) &\longrightarrow_{\mathbf{cc}} (\boldsymbol{\nu}\, z)\,(\mathcal{FW}(za) \mid \mathcal{M}(zy)) \\ &\longrightarrow_{\mathbf{cc}} (\boldsymbol{\nu}\, z)\,(\mathcal{M}(ay)) \end{aligned}$$

corresponds with

$$\begin{aligned} [\![\mathcal{SW}(xy) \mid \mathcal{M}(xa)]\!]_\pi &\stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\,(xv.zw.\overline{v}w \mid \overline{z}y) \mid \overline{x}a \\ &\longrightarrow_\pi (\boldsymbol{\nu}\, z)\,(zw.\overline{a}w \mid \overline{z}y) \\ &\longrightarrow_\pi (\boldsymbol{\nu}\, z)\,(\overline{a}y) \end{aligned}$$

□

3.3 Encoding the π-Calculus in the $\mathbf{cc}_\pi$[9]

The crucial difference in syntactic structure between the monadic asynchronous π-calculus (without summation and matching) and the $\mathbf{cc}_\pi$ is the absence of arbitrary input prefixing from the latter. However, it shall be shown that input prefix behaviour can be represented using only the seven basic atoms of the $\mathbf{cc}_\pi$.

Firstly, substitution decomposition as a method of manipulating name dependencies within a term is explained. An input prefix meta-construct is then introduced for $\mathbf{cc}_\pi$-terms, along with a mapping of terms containing this abstraction to concrete members of $P_{\mathbf{cc}_\pi}$. This mapping is constructed to express each possible relationship between an input prefix and the prefixed term as some finite composition of $\mathbf{cc}_\pi$-atoms. Finally, a full encoding of P_π in the $P_{\mathbf{cc}_\pi}$ is defined, and it is shown that π-processes and their corresponding $\mathbf{cc}_\pi$ translations are (weakly) bisimilar and thus barbed congruent.

Proposition 3.3.1 (substitution decomposition [9])

(i) $\mathbf{c}\langle x^- \tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{c}\langle c\tilde{v}\rangle)$ and $\mathbf{c}\langle \tilde{u}x^+\tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle cx\rangle \mid \mathbf{c}\langle \tilde{u}c^+\tilde{v}\rangle)$ with c fresh.

(ii) $\forall \mathbf{c}\langle \tilde{v}\rangle \in P_{\mathbf{cc}_\pi} : P \approx (\boldsymbol{\nu}\, \tilde{c})\,(\mathbf{c}_1\langle \tilde{v}_1\rangle\sigma_1 \mid \ldots \mid \mathbf{c}_n\langle \tilde{v}_n\rangle\sigma_n\rangle)$ for some $\mathbf{c}_i\langle \tilde{v}_i\rangle \in \mathbf{A}_\pi$ and an injective renaming σ_i, and where each $v \in \tilde{v}_i$ is pairwise distinct.

Proof The former can be shown by mechanical verification for each applicable **cc**-atom. The latter is proved by the former. Further details are given in Appendix A.2. □

Substitution decomposition allows names to be "pushed out" of $\mathbf{cc}_\pi$-atoms by introducing a Forwarder as a level of indirection over the original term. This allows atomic agents with repeated parameters, which cannot be directly generated through substitution, to be decomposed into terms without repeated parameters. Part (ii) uses this result

to state that any $\mathbf{cc}_\pi$ atomic agents can be expressed as a parallel composition of the basic atoms belonging to $\mathbf{A}_\pi$. This holds because the only cases where this statement is not trivially true arise for atomic agents with repeated argument names, and can be dealt with using (i): it is always possible to use fresh names for the required restrictions which can then be moved to the front of the term by structural congruence. Furthermore, a stronger result can be shown to hold as Killer and Forwarder can be defined using the Duplicator, reducing the basic atoms to a minimal set of five. However, this report does not explore this result in further detail. Substitution decomposition will be encountered below when dealing with the binding aspects of the input prefix.

Example 3.3.2 Consider $\mathbf{s}\langle xxx\rangle$. By Proposition 3.3.1 we have

$$\begin{aligned}\mathbf{s}\langle xxx\rangle &\approx (\boldsymbol{\nu}\, c_1)\,(\mathbf{fw}\langle c_1 x\rangle \mid \mathbf{s}\langle c_1 x x\rangle)\\ &\approx (\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{fw}\langle x c_1\rangle \mid \mathbf{s}\langle c_1 x c_2\rangle \mid \mathbf{fw}\langle c_2 x\rangle) \qquad \text{(Proposition 2.4.16)}\end{aligned}$$

which is straightforward to verify. Let $L \stackrel{\text{def}}{=} xw.xz.\overline{x}z$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(xz.\overline{c_1}z|c_1 w.xz.\overline{c_2}z|c_2 z.\overline{x}z)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} xz.\overline{x}z$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(\overline{c_1}y \mid c_1 w.xz.\overline{c_2}z \mid c_2 z.\overline{x}z)$

(2) Check $L' \approx R'$. However,

$$R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(xz.\overline{c_2}z \mid c_2 z.\overline{x}z)\ ,$$

so check $L' \approx R''$.

- $L' \xrightarrow{xy} L'' \stackrel{\text{def}}{=} \overline{x}y$ matched by $R'' \xrightarrow{xy} R''' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(\overline{c_2}y \mid c_2 z.\overline{x}z)$

(3) $L'' \approx R'''$ as $R''' \longrightarrow_\beta R^{iv} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,\overline{x}y$ and $L'' \equiv R^{iv}$. □

Definition 3.3.3 (input meta-prefix mapping) Application of the input prefix meta-construct to $P \in P_{\mathbf{cc}_\pi}$ is represented by $a^*x.P$. The $\mathbf{cc}_\pi$ term denoted by $a^*x.P$ is formed by applying the rules in Table 3.3 where c, c_1 and c_2 are fresh and pairwise distinct. A single translation step resulting from the application of the nth rule is denoted by $\stackrel{(n)}{\longmapsto}$. □

The meta-prefix mapping rules are a deconstruction of the semantics of the input prefix, the idea being to emulate the essential behaviour of input-prefixed terms through the available constructs. The rules have been categorised according to the different behavioural aspects of the input prefix that they capture. The Composition rules deal with the structural implications of input-prefixing the other syntactical constructs. When the prefixed body contains no placeholders for received names, input-prefixing serves purely as a synchronisation mechanism; this behaviour is represented by the Synchronisation rules.

The Binding rules emulate the name binding abstraction of the input prefix by forwarding input names to the relevant locations. The general mechanism is closely related to substitution decomposition (Proposition 3.3.1), as expressed by the rules of Binding II and III. Binding II pushes bound output parameters out of a term into additional forwarders, and Binding III inserts forwarders in front of bound input parameters; rule (XI)

Composition				
(I)	$x^*z.(P \mid Q)$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid c_1^*z.P \mid c_2^*z.Q)$	
(II)	$x^*z.(\boldsymbol{\nu}\, c')\,P$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,x^*z.P\{^{c}/_{c'}\}$	
(III)	$x^*z.\mathbf{0}$	$\stackrel{\text{def}}{=}$	$\mathbf{k}\langle x\rangle$	
(IV)	$x^*z.!\,P$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid !\,c^*z.(P \mid \mathbf{m}\langle cz\rangle))$	
Synchronisation				
(V)	$x^*z.\mathbf{c}\langle v^+\tilde{w}\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,(\mathbf{s}\langle xcv\rangle \mid \mathbf{c}\langle c^+\tilde{w}\rangle)$	$z \notin \{v\tilde{w}\}$
(VI)	$x^*z.\mathbf{c}\langle v^-\tilde{w}\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,(\mathbf{s}\langle xvc\rangle \mid \mathbf{c}\langle c^-\tilde{w}\rangle)$	$z \notin \{v\tilde{w}\}$
Binding I				
(VII)	$x^*z.\mathbf{m}\langle vz\rangle$	$\stackrel{\text{def}}{=}$	$\mathbf{fw}\langle xv\rangle$	$z \neq v$
(VIII)	$x^*z.\mathbf{fw}\langle zv\rangle$	$\stackrel{\text{def}}{=}$	$\mathbf{b}_l\langle xv\rangle$	$z \neq v$
(IX)	$x^*z.\mathbf{fw}\langle vz\rangle$	$\stackrel{\text{def}}{=}$	$\mathbf{b}_r\langle xv\rangle$	$z \neq v$
Binding II				
(X)	$x^*z.\mathbf{c}\langle \tilde{w}_1 z^+ \tilde{w}_2\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,x^*z.(\mathbf{fw}\langle cz\rangle \mid \mathbf{c}\langle \tilde{w}_1 c^+ \tilde{w}_2\rangle)$	$z \notin \{\tilde{w}_1\}$
Binding III				
(XI)	$x^*z.\mathbf{c}\langle z^-\tilde{w}\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c)\,x^*z.(\mathbf{fw}\langle zc\rangle \mid \mathbf{c}\langle c^-\tilde{w}\rangle)$	
(XII)	$x^*z.\mathbf{b}_r\langle vz^-\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c_1, c_2, c_3)\,x^*z.(\mathbf{d}\langle vc_1c_2\rangle \mid \mathbf{s}\langle c_1zc_3\rangle \mid \mathbf{b}_r\langle c_2c_3\rangle)$	$z \neq v$
(XIII)	$x^*z.\mathbf{s}\langle uz^-v\rangle$	$\stackrel{\text{def}}{=}$	$(\boldsymbol{\nu}\, c_1, c_2)\,x^*z.(\mathbf{s}\langle uc_1c_2\rangle \mid \mathbf{m}\langle c_1z\rangle \mid \mathbf{b}_l\langle c_2v\rangle)$	$z \neq u$

Table 3.3: Input meta-prefix mapping for $\mathbf{cc}_\pi$ terms.

is applied when the bound input name is the subject of the term, and rules (XII) and (XIII) catch the remaining cases. As intended by substitution decomposition, these rules also handle the terms that contain repeated parameter bindings, which is enforced by the side conditions of rules (IV) to (IX). The rules of Binding I represent the binding base cases and are simply instances of structural congruence, as is rule (III).

The structure of the meta-prefix mapping rules corresponds to the syntax of $\mathbf{cc}_\pi$, and their semantics provides insight into the origins of the **cc** atoms and the particular behaviour that each encapsulates. For example, the Duplicator and Killer arise from rules (I) and (III) respectively, and the synchroniser, of course, has its roots in the Synchronisation rules.

Proposition 3.3.4 (properties of input meta-prefix mapping) The input meta-prefix mapping for $\mathbf{cc}_\pi$-terms exhibits the following properties for all $P, Q \in P_{\mathbf{cc}_\pi}$.

(i) $xz.P \approx x^*z.P$.
(ii) $P \approx Q \Rightarrow x^*z.P \approx x^*z.Q$.
(iii) $x^*z.P \mid \mathbf{m}\langle xv\rangle \longrightarrow\!\!\approx P\{^{v}/_{z}\}$.

Proof For (i), we proceed inductively over the rules in Table 3.3, and (ii) follows directly. Rules (I) and (IV) use Proposition 3.2.7 after the induction step, and the remaining cases simply require mechanical verification. The approach to (iii) has the same structure as for (i). Further details are given in Appendix A.3. □

These results verify that meta-prefix mapping has the desired behaviour, and that it both observes and preserves bisimilarity, and thus barbed congruence. The decomposition of the π-calculus input prefix into $\mathbf{cc}_\pi$-atoms enables the complete encoding of P_π in the $\mathbf{cc}_\pi$.

Example 3.3.5 Let $P \stackrel{\text{def}}{=} x^*z.(\mathbf{0} \mid \mathbf{0})$.

$$\begin{aligned} P &\stackrel{(\text{I})}{\longmapsto} (\boldsymbol{\nu}\, c_1c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid c_1^*z.\mathbf{0} \mid c_2^*z.\mathbf{0}) \\ &\stackrel{(\text{III})}{\longmapsto} (\boldsymbol{\nu}\, c_1c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{k}\langle c_1\rangle \mid c_2^*z.\mathbf{0}) \\ &\stackrel{(\text{III})}{\longmapsto} (\boldsymbol{\nu}\, c_1c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{k}\langle c_1\rangle \mid \mathbf{k}\langle c_2\rangle) \end{aligned}$$

As expected, the $\mathbf{cc}_\pi$ translation of P is a process which simply kills the input. Rule (I) is first applied to distribute the meta prefix over the composition, and the resulting meta-terms are instances of the base case represented by rule (III). □

Example 3.3.6 Let $Q \stackrel{\text{def}}{=} a^*x.b^*y.\mathbf{m}\langle xv\rangle$. Intuitively, it can be seen that the first meta-prefix of this term binds its input to the message subject, but the second has a synchronisation effect only. Consequently, the translation of Q demonstrates the function of rule (X) for binding output names, and of rules (V) and (VI) for pure synchronisation. From the application of each rule, we can observe how the different atoms come into being and what roles they play.

$$\begin{aligned} Q &\stackrel{(\text{V})}{\longmapsto} a^*x.(\boldsymbol{\nu}\, c_1)\,(\mathbf{s}\langle bc_1x\rangle \mid \mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{II})}{\longmapsto} (\boldsymbol{\nu}\, c_1)\,(a^*x.(\mathbf{s}\langle bc_1x\rangle \mid \mathbf{m}\langle c_1v\rangle)) \\ &\stackrel{(\text{I})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots3})\,(\mathbf{d}\langle ac_2c_3\rangle \mid c_2^*x.\mathbf{s}\langle bc_1x\rangle \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{X})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots3})\,(\mathbf{d}\langle ac_2c_3\rangle \mid (\boldsymbol{\nu}\, c_4)\,(c_2^*x.(\mathbf{fw}\langle c_4x\rangle \mid \mathbf{s}\langle bc_1c_4\rangle)) \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\equiv (\boldsymbol{\nu}\, c_{1\ldots4})\,(\mathbf{d}\langle ac_2c_3\rangle \mid c_2^*x.(\mathbf{fw}\langle c_4x\rangle \mid \mathbf{s}\langle bc_1c_4\rangle) \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{I})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots4})\,(\mathbf{d}\langle ac_2c_3\rangle \mid (\boldsymbol{\nu}\, c_5c_6)\,(\mathbf{d}\langle c_2c_5c_6\rangle \mid c_5^*x.\mathbf{fw}\langle c_4x\rangle \mid c_6^*x.\mathbf{s}\langle bc_1c_4\rangle) \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\equiv (\boldsymbol{\nu}\, c_{1\ldots6})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid c_5^*x.\mathbf{fw}\langle c_4x\rangle \mid c_6^*x.\mathbf{s}\langle bc_1c_4\rangle \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{IX})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots6})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid \mathbf{b}_r\langle c_5c_4\rangle \mid c_6^*x.\mathbf{s}\langle bc_1c_4\rangle \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{VI})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots6})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid \mathbf{b}_r\langle c_5c_4\rangle \mid (\boldsymbol{\nu}\, c_7)\,(\mathbf{s}\langle c_6bc_7\rangle \mid \mathbf{s}\langle c_7c_1c_4\rangle) \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\equiv (\boldsymbol{\nu}\, c_{1\ldots7})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid \mathbf{b}_r\langle c_5c_4\rangle \mid \mathbf{s}\langle c_6bc_7\rangle \mid \mathbf{s}\langle c_7c_1c_4\rangle \mid c_3^*x.\mathbf{m}\langle c_1v\rangle) \\ &\stackrel{(\text{V})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots7})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid \mathbf{b}_r\langle c_5c_4\rangle \mid \mathbf{s}\langle c_6bc_7\rangle \mid \mathbf{s}\langle c_7c_1c_4\rangle \mid \\ &\qquad\qquad (\boldsymbol{\nu}\, c_8)\,(\mathbf{s}\langle c_3c_8c_1\rangle \mid \mathbf{m}\langle c_8v\rangle)) \\ &\stackrel{(\text{V})}{\longmapsto} (\boldsymbol{\nu}\, c_{1\ldots8})\,(\mathbf{d}\langle ac_2c_3\rangle \mid \mathbf{d}\langle c_2c_5c_6\rangle \mid \mathbf{b}_r\langle c_5c_4\rangle \mid \mathbf{s}\langle c_6bc_7\rangle \mid \mathbf{s}\langle c_7c_1c_4\rangle \mid \mathbf{s}\langle c_3c_8c_1\rangle \mid \mathbf{m}\langle c_8v\rangle) \end{aligned}$$

By informal inspection, we can see that the $\mathbf{cc}_\pi$ translation has preserved the essential

$$[\![P \mid Q]\!]_{cc_\pi} \stackrel{\text{def}}{=} [\![P]\!]_{cc_\pi} \mid [\![Q]\!]_{cc_\pi} \qquad [\![(\boldsymbol{\nu}\, z)\, P]\!]_{cc_\pi} \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, z)\, [\![P]\!]_{cc_\pi} \qquad [\![!\, P]\!]_{cc_\pi} \stackrel{\text{def}}{=} !\, [\![P]\!]_{cc_\pi}$$

$$[\![\overline{x}z]\!]_{cc_\pi} \stackrel{\text{def}}{=} \mathbf{m}\langle xz\rangle \qquad [\![xz.P]\!]_{cc_\pi} \stackrel{\text{def}}{=} x^*z.[\![P]\!]_{cc_\pi} \qquad [\![\mathbf{0}]\!]_{cc_\pi} \stackrel{\text{def}}{=} \mathbf{0}$$

Table 3.4: Translation from P_π to $P_{\mathbf{cc}_\pi}$.

behaviour of Q. Initially, $\mathbf{d}\langle ac_2c_3\rangle$ is the only active agent, ensuring that synchronisation over a occurs before any other action as well as propagating the name received during synchronisation. This activates $\mathbf{s}\langle c_6bc_7\rangle$, permitting the synchronisation over b, corresponding to the second prefix, to occur whilst freeing the stored name v through $\mathbf{s}\langle c_3c_8c_1\rangle$. The $\mathbf{b}_r\langle c_5c_4\rangle$ agent, to which the name from the initial reduction is forwarded, will then send v to this name. □

Definition 3.3.7 (translation from P_π to $P_{\mathbf{cc}_\pi}$) The mapping $[\![\cdot]\!]_{cc_\pi} : P_\pi \to P_{\mathbf{cc}_\pi}$ is given by the rules in Table 3.4. □

The bulk of the translation work naturally lies in the mapping of input prefixes; the remaining syntactic operators are essentially unaffected by the translation and are directly carried over into the $\mathbf{cc}_\pi$. More precisely, the encoding is homomorphic, and also message preserving, $[\![\overline{x}z]\!]_{cc_\pi} \approx \overline{x}z$ [9]. The latter property indicates that the encoding preserves the essential meaning and behaviour of the translated terms. Indeed, the Message and $\mathbf{0}$ process constitute the base cases of this inductive translation; the role of prefix-mapping as the source of the other atoms provides insight on the relationship between the mechanics of the π-calculus and the function of each **cc**-atom.

Lemma 3.3.8 (properties of $[\![\cdot]\!]_{cc_\pi}$) The translation from P_π to $P_{\mathbf{cc}_\pi}$ by the function $[\![\cdot]\!]_{cc_\pi}$ exhibits the following properties for all $P, Q \in P_\pi$.

(i) $P \approx [\![P]\!]_{cc_\pi}$
(ii) $P \equiv Q \Rightarrow [\![P]\!]_{cc_\pi} \approx [\![Q]\!]_{cc_\pi}$.
(iii) $P \longrightarrow Q \Rightarrow [\![P]\!]_{cc_\pi} \longrightarrow\!\approx [\![Q]\!]_{cc_\pi}$.

Proof The proof of (i) is a straightforward verification proceeding inductively from the Message and $\mathbf{0}$ base cases over each of the mapping rules using the congruence property of $\approx$. (ii) follows directly from (i). (iii) has a similar structure to Proof A.1.4. Further details are given in Appendix A.4. □

Lemma 3.3.8 (i) shows that we can indeed express any term in the asynchronous π-calculus as a term in the $\mathbf{cc}_\pi$ up to bisimilarity, and thus barbed congruence. Parts (ii) and (iii) illustrate some of the subsequent properties of the encoding, verifying that translated terms behave as expected.

Example 3.3.9 In Example 3.2.8, we showed that $\mathbf{sw}(xy) \approx [\![\mathcal{SW}(xy)]\!]_\pi$ as a demonstration of the mapping from **cc**-terms to $\mathbf{cc}_\pi$-terms (Definition 3.2.3 and Proposition 3.2.4).

We now illustrate the encoding of π-terms in the $\mathbf{cc}_\pi$ (Definition 3.3.7 and Lemma 3.3.8) by verifying that $\mathbf{sw}(xy) \approx [\![\mathbf{sw}(xy)]\!]_{cc_\pi}$.

$$\begin{aligned}
[\![\mathbf{sw}(xy)]\!]_{cc_\pi} &\stackrel{\text{def}}{=} [\![xz.\overline{z}y]\!]_{cc_\pi} && \text{(Example 3.2.8)}\\
&= x^*z.[\![\overline{z}y]\!]_{cc_\pi} && \text{(Definition 3.3.7)}\\
&= x^*z.\mathbf{m}\langle zy\rangle && \text{(Definition 3.3.7)}
\end{aligned}$$

The translation is completed by expanding the input meta-prefix according to Definition 3.3.3.

$$\begin{aligned}
x^*z.\mathbf{m}\langle zy\rangle &\stackrel{(\text{X})}{\longmapsto} (\boldsymbol{\nu}\, c)\, x^*z.(\mathbf{fw}\langle cz\rangle \mid \mathbf{m}\langle cy\rangle)\\
&\stackrel{(\text{I})}{\longmapsto} (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{d}\langle xc_1c_2\rangle \mid c_1^*z.\mathbf{fw}\langle cz\rangle \mid c_2^*z.\mathbf{m}\langle cy\rangle)\\
&\stackrel{(\text{IX})}{\longmapsto} (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid c_2^*z.\mathbf{m}\langle cy\rangle)\\
&\stackrel{(\text{V})}{\longmapsto} (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid (\boldsymbol{\nu}\, c_3)\, (\mathbf{s}\langle c_2c_3c\rangle \mid \mathbf{m}\langle c_3y\rangle))\\
&\equiv (\boldsymbol{\nu}\, c, c_{1\ldots3})\, (\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid \mathbf{s}\langle c_2c_3c\rangle \mid \mathbf{m}\langle c_3y\rangle)
\end{aligned}$$

Let $L \stackrel{\text{def}}{=} xz.\overline{z}y$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_{1\ldots3})\, (xw.(\overline{c_1}w \mid \overline{c_2}w) \mid c_1w.cz.\overline{w}z \mid c_2w.c_3z.\overline{c}z \mid \overline{c_3}y)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xu} L' \stackrel{\text{def}}{=} \overline{u}y$ matched by
  $R \xrightarrow{xu} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_{1\ldots3})\, (\overline{c_1}u \mid \overline{c_2}u \mid c_1w.cz.\overline{w}z \mid c_2w.c_3z.\overline{c}z \mid \overline{c_3}y)$

(2) $L' \approx R'$ as $R' \longrightarrow_\beta^* R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_{1\ldots3})\, \overline{u}y$ and $L' \equiv R''$. □

Chapter 4

Related Work

This chapter presents some related work.

4.1 Replication in Concurrent Combinators [3]

The **cc**, presented in [2], demonstrates that the fundamental mechanics of process communication, as abstracted by process calculi such as the asynchronous π-calculus, can be expressed using only CC atoms and constructs of parallel composition and name hiding. At the heart of this work lies the **cc** encoding of the π-calculus input prefix which decomposes the implicit binding and synchronisation operations of the prefix into the basic units of process behaviour embodied by the atoms. The structure of this encoding both illustrates the origin and purpose of each atom, and provides a means for verifying the expressiveness of the **cc**.

The present paper generalises and extends the prior work to create a concurrent combinator system, $\mathbf{cc}_\varphi$, capable of representing replication. As discussed previously, the notion of replication is necessary in the name passing abstraction of concurrent computation because the mechanics of communication would otherwise strictly consume a process: no representation of unbounded behaviour would be possible. The main result of this paper shows that replication behaviour can be encoded using a finite set of $\mathbf{cc}_\varphi$ atoms without an explicit replication operator.

4.1.1 Generalised Concurrent Combinators

A generalised concurrent combinator system is formally defined as a triple $\langle P_{\mathbb{A}}, \equiv, \longrightarrow \rangle$ where $P_{\mathbb{A}}$ is the set of terms derived from $\mathbb{A}$, $\equiv$ is the structural equality over terms, and $\longrightarrow$ is the reduction relation over terms. The first element consists of a finite set of atoms, and a grammar for constructing terms defined as for the **cc** but without the replication operator. Structural equivalence is similarly modified to exclude replication. The reduction relation is specified through a rule schema that describes dyadic interaction between atoms, and includes (PAR), (R-RES) and (STRUCT). Pointedness, equality over agents, and β-equivalence as defined for the **cc** can be directly applied to the general case.

A specific combinator system, $\mathbb{S}$, is an instance of this general framework, and is *behaviourally closed* under prefix and replication if input prefix, as understood in the **cc**, and *simple replication* behaviour can be expressed, even though $\mathbb{S}$ lacks the corresponding

syntactic constructs. Simple replication behaviour is described by

$$\mathbf{!}^*_x P \mid \mathbf{?}M(x) \longrightarrow \mathbf{!}^*_x P \mid P$$

and implies eager replication as found in the π-calculus. These two properties are formally defined as the prefix and replication closures [3], which capture the essence of the respective abstractions. The original **cc** (without replication) [2] is clearly an instance closed under prefix, but not simple replication.

4.1.2 Encoding Replication

The theorem that an instance of the generalised concurrent combinator system which is behaviourally closed under both prefix and simple replication exists is proved by presenting $\mathbf{cc}_\varphi$ as such an instance. $\mathbf{cc}_\varphi$ contains the original **cc** atoms, plus *persistent* versions of each atom, excluding the Message, named the *atomic replicators*. Structural equivalence is as before, but the reduction relation comprises additional rules of the form

$$(\textsc{rep}) \quad !\mathcal{C}(x\tilde{v}) \mid \mathcal{M}(xy) \longrightarrow !\mathcal{C}(x\tilde{v}) \mid P \ ,$$

where $!\mathcal{C}(x\tilde{v})$ denotes the peristent version of $\mathcal{C}(x\tilde{v})$ (! is not a syntactic operator), and $\mathcal{C}(x\tilde{v}) \mid \mathcal{M}(xy) \longrightarrow P$. Persistence of atomic replicators across reduction reflects the essence of replication. The encoding of input prefixes is similarly extended with rules for the atomic replicators. This provides behavioural closure under prefix, but not replication because terms such as $\mathbf{!}^*_u(\boldsymbol{\nu}\, z)\, \mathcal{M}(xz)$ cannot be represented due to an inherent restriction on name distinction [3].

This problem is addressed by introducing the *name generator* atom

$$(\textsc{gen}) \quad \#\mathcal{N}(x) \mid \mathcal{M}(xy) \longrightarrow \#\mathcal{N}(x) \mid (\boldsymbol{\nu}\, z)\, \mathcal{M}(yz)$$

which can forever send new names to a specified location, and *template* atoms of the form

$$(\textsc{tem}) \quad \#\mathcal{C}(x) \mid \mathcal{M}(xy) \longrightarrow \#\mathcal{C}(x) \mid rcv(\tilde{v}).\, \mathcal{C}(\tilde{v})$$

for each of the original and persistent **cc** atoms, where *rcv* is as defined in Example 3.1.5. The templates allow agents with arbitrary parameter values to be created during computation. Additional rules for these new atoms are required for prefix mapping.

The general idea behind $\mathbf{cc}_\varphi$ replicators is to create a template for each atomic agent of the term being replicated and to instantiate these templates with existing names and/or new names using name generators. The actual encoding is quite involved and consists of three parts: an interface to the outside which accepts term replication requests, a name emitter that provides the required names, and a constructor which uses these names to instantiate the requested term.

4.2 Solos in Concert [4]

The fusion calculus is calculus of concurrency related to the π-calculus. The subcalculus formed by removing continuations and summation, the fusion calculus of solos, is as expressive as the parent calculus. This is shown using similar techniques to those discussed in Chapter 3 for the π-calculus and $\mathbf{cc}_\pi$. In particular, a similarly structured encoding from the parent calculus to the subcalculus, which tackles the prefix construct, is used as the basis for subsequent structural induction proofs.

4.2.1 The Fusion Calculus [7]

The *fusion calculus* is an extension through simplification of the π-calculus whose defining features are the symmetry of input and output, and a single operator for name binding. This is in contrast to the asynchronous π-calculus which is developed from the full π-calculus by increasing the asymmetry between input and output, and like its parent has two binding operators.

Input and output in the fusion calculus can perhaps be better described as action and co-action as they are essentially dual versions of the same operation. Whereas the π-calculus employs name passing as the basic communication primitive, the fusion calculus has fusion which simply equates the objects supplied by the two communicating processes. For example,

$$\overline{x}y.P \mid xz.Q \mid R \xrightarrow{\{y=z\}} P \mid Q \mid R\,,$$

where $\xrightarrow{\{y=z\}}$ expresses that the fusion has a global effect that applies equally to both the sender and receiver, and also any other process running in parallel. Communication in the fusion calculus can thus be seen as the update of an implicit shared state.

The semantics of the π-calculus input prefix combines the action of receiving a message with the binding of its scope. In the fusion calculus, binding has been decoupled from input and embodied within a single binding operator, scope, that captures the fundamental behaviour behind both input binding and the restriction operator of the π-calculus. As a result, binding due to scope is neither mandatory, as with input binding, nor impossible, as with restriction, and can be thought of as a means to control the extent and direction of a fusion. For instance, the transition given in the previous example identifies y and z, but leaves the implications of this fusion to future derivations of $P \mid Q \mid R$. Also,

$$(z)(\overline{x}vy.P \mid (w)(xwz.Q \mid R) \mid S \xrightarrow{\mathbf{1}} (P \mid (Q \mid R)\{v/w\} \mid S)\{y/z\}$$

illustrates that input binding is no longer directly tied to the actual input operator, but may be arbitrarily scoped. $\mathbf{1}$ denotes the identity relation, and $\xrightarrow{\mathbf{1}}$ expresses that the transition does not involve any fusions with global scope.

The simpler nature of the fusion calculus, in comparison to the π-calculus, is due to the decoupling of binding from input actions and having a single binding operator. This is evidenced by the symmetry of input and output, the smaller set of transition rules, the fact that the fusion calculus has only one sensible bisimulation congruence [7], and its general expressiveness. The relationship between the π-calculus and the fusion calculus can be understood by thinking of $xz.P_\pi$ as $(z)xz.P_f$, $(\boldsymbol{\nu}\, z)\, Q_\pi$ as $(z)Q_f$ if $Q \neq xz.R$, and $\mathbf{1}$ as τ. For example,

$$\overline{x}y.P \mid (z)xz.Q \xrightarrow{\mathbf{1}} P \mid Q\{y/z\}$$

illustrates how π-calculus interaction can be emulated in the fusion calculus by scoping input actions. By extending these mappings over the structure of π-syntax, it can be seen that the π-calculus is contained within the fusion calculus.

4.2.2 The Fusion Calculus of Solos [4]

The asynchronous π-calculus is formed by dropping output continuations from process syntax. To avoid breaking input and output symmetry, an asynchronous fusion calculus

can be formed by removing continuations completely. The remaining input and output particles are named *solos*, and the resulting calculus, the *fusion calculus of solos*, which shall be referred to in this report as just the solos calculus. As with the asynchronous π-calculus, the solos calculus omits summation (see Section 2.1).

In fact, the solos calculus is capable of encoding prefixes (and guarded summation), and thus is as expressive as the original calculus. Two different encodings are presented, one using the match operator which results in correspondence up to barbed bisimilarity, and the other using only solos, scope and parallel composition which results in correspondence up to weak barbed bisimilarity. From these results, it can be shown that each encoding preserves its relative level of correspondence between processes.

As with the encoding of P_π in the $P_{\mathbf{cc}_\pi}$, the encoding of the fusion calculus in the solos calculus using match is a homomorphism except in the case of prefix. Once again, the majority of the translation work involves re-expressing process behaviour due to prefixing as some finite composition of components. However, encoding prefix in the solos calculus is simpler than in the π-calculus due to the relative simplicity of the prefix construct in the former and the expressiveness of fusion interaction. Being free of any binding semantics, prefix in the fusion calculus serves only as a mechanism for synchronisation, and thus can be replaced by a suitable protocol.

The expressiveness of fusion allows transitions of the form

$$\overline{x}vy.P \mid (z)xzz \xrightarrow{\{v=y\}} P$$

where the *catalyst* agent $(z)xzz$ fuses the two names sent to it, even if bound in P. The solos calculus protocol used to emulate prefix synchronisation exploits this effect in combination with match operator guards to detect when interaction has occurred.

$$\begin{aligned}
[\![x\tilde{v}.P]\!] &\stackrel{\text{def}}{=} (wz)(x\tilde{v}zww \mid [w=z][\![P]\!]) \\
[\![\overline{x}\tilde{v}.P]\!] &\stackrel{\text{def}}{=} (wz)(\overline{x}\tilde{v}wwz \mid [w=z][\![P]\!])
\end{aligned}$$

Thus,

$$\begin{aligned}
[\![(v)(\overline{x}u.P \mid xv.Q)]\!] &\stackrel{\text{def}}{=} (v)((wz)(\overline{x}\tilde{v}wwz \mid [w=z][\![P]\!]) \mid (wz)(x\tilde{v}zww \mid [w=z][\![Q]\!])) \\
&\equiv (vw_1w_2z_1z_2)(\overline{x}\tilde{v}w_1w_1z_1 \mid [w_1=z_1][\![P]\!] \mid x\tilde{v}z_2w_2w_2 \mid [w_2=z_2][\![Q]\!])
\end{aligned}$$

Three protocol objects are appended to the original input and output actions, and the match guards prevent the continuations $[\![P]\!]$ and $[\![Q]\!]$ from proceeding. When interaction occurs, the subsequent fusion identifies w and z on each side of the interaction satisfying the conditions of both guards.

The second encoding does not use match but requires the notion of a "top-level" context. The synchronisation protocol used in this encoding works by recursively stripping prefixes from their continuations, and encoding every action with a fresh scoped subject thus preventing any progress. However, for each prefix, a key term is generated that will fuse the fresh name with the original subject upon interaction with a catalyst, whilst also creating a new catalyst to replace the consumed catalyst. Prefixes are appended with additional protocol objects that allow their continuations to proceed once interaction between the prefixes has occurred.

$$\begin{aligned}
[\![x\tilde{v}.P]\!]_s &\stackrel{\text{def}}{=} (s'w)(w\tilde{v}ss' \mid [\![P]\!]_{s'} \mid (t)(\overline{s}xwt \mid (z)tzzt)) \\
[\![\overline{x}\tilde{v}.P]\!]_s &\stackrel{\text{def}}{=} (s'w)(w\tilde{v}s's \mid [\![P]\!]_{s'} \mid (t)(\overline{s}xwt \mid (z)tzzt))
\end{aligned}$$

The resulting transitions can be viewed as a chain reaction, started by the initial catalyst added to the top-level encoding, which is scoped over the catalyst subject.

$$[\![P]\!] \stackrel{\text{def}}{=} (s)([\![P]\!]_s \mid (z)szzs)$$

Both of the presented encodings successfully decompose prefixes into communication and synchronisation components. As with the encoding of the π-calculus in the $\mathbf{cc}_\pi$ replication proves to be the tricky case: the first encoding behaves as desired if replication is added to the solos calculus, but the same does not hold for the second. It is also shown that guarded summation can be encoded with a weakening of correspondence, and that expressiveness of the polyadic solos calculus is retained by the dyadic solos calculus, but not the monadic solos calculus. This is in contrast to the work presented in Chapter 3 which is based on the monadic asynchronous π-calculus.

The mappings from the fusion calculus to the solos calculus (the first encoding) and π-calculus to the $\mathbf{cc}_\pi$ (as presented in Definition 3.3.7) are similar in that replication is included as a explicit syntactic construct. However, as discussed in the preceding section, [3] extends the **cc** with additional atoms that preclude the need for an explicit replication operation.

Chapter 5

Conclusions and Further Work

5.1 Conclusions

This report has studied the asynchronous π-calculus and the **cc** concurrent combinator system. In Chapter 2 covers some details of the former, in particular exploring relations that could be adopted for process equivalence. Chapter 1 introduced, and Chapter 3 presented a formal account of the **cc** and the relationship between the two systems. The main result, that the monadic asynchronous π-calculus without summation or matching can be encoded within the **cc**, was built up over a series of proofs in Chapter 3, the details of which are given in Appendix A. The case involving replication proved to be the trickiest, but its proof provides the **cc** with full computational power. The crux of the work is the decomposition of the π-calculus input prefix into a finite composition of **cc**atoms.

5.2 Further Work

The material in this report is mostly of an introductory or informal nature without great detail of techniques or proofs, apart some selected parts of Chapter 3. However, it provides a good basis for pursuing the study of process calculi and concurrency further. As briefly discussed in Section 4.1, it is possible to represent unbounded behaviour using some additional atoms that replace the replication operator [3]; it would be interesting to examine this work more thoroughly. It has also been mentioned that the **cc** can be used to characterise the expressiveness of other subcalculi of the π-calculus, as in [9]. This has not been covered in this report, but would be a natural step for further work.

Appendix A

Details of Proofs from Chapter 3

Some selected proofs from Chapter 3 are now presented in more detail. Application of the induction hypothesis is abbreviated to IH.

A.1 Proposition 3.2.4

Syntactic and operational correspondence

The translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$ by the function $[\![\cdot]\!]_\pi$ exhibits the following properties for all $P, Q \in P_{\mathbf{cc}}$.

(i) $\mathsf{fn}(P) = \mathsf{fn}([\![P]\!]_\pi)$.
(ii) For any substitution σ, $[\![P\sigma]\!]_\pi \equiv [\![P]\!]_\pi\sigma$.
(iii) $P \equiv_{\mathbf{cc}} Q \Rightarrow [\![P]\!]_\pi \equiv_\pi [\![Q]\!]_\pi$.
(iv) $P \longrightarrow_{\mathbf{cc}} Q \Rightarrow [\![P]\!]_\pi \longrightarrow_\pi [\![Q]\!]_\pi$.

Proof A.1.1 (i) We use induction on the structure of terms. The base cases consist of comparing the free names of the individual **cc** atomic agents and the **0** term against their corresponding $\mathbf{cc}_\pi$ mappings. Then for each **cc** structural construct, we apply the corresponding mapping rule and use the induction hypothesis to show that the free names of a arbitrary term are preserved.

Base cases (names are implicitly universally quantified in each case).

1. $\mathsf{fn}(\mathbf{0}) =_\alpha \mathsf{fn}([\![\mathbf{0}]\!]_\pi)$

$$\mathsf{fn}([\![\mathbf{0}]\!]_\pi) = \mathsf{fn}(\mathbf{0}) \qquad \text{(Definition 3.2.3)}$$

2. $\mathsf{fn}(\mathcal{M}(xy)) = \mathsf{fn}([\![\mathcal{M}(xy)]\!]_\pi)$

$$\begin{aligned} \mathsf{fn}([\![\mathcal{M}(xy)]\!]_\pi) &= \mathsf{fn}(\mathbf{m}\langle xy\rangle) && \text{(Definition 3.2.3)} \\ &= \mathsf{fn}(\overline{x}y) && \text{(Definition 3.2.1)} \\ &= \{x, y\} && \text{(Definiton 2.1.1)} \\ &= \mathsf{fn}(\mathcal{M}(xy)) && \text{(Definition 3.1.3)} \end{aligned}$$

3. $\mathsf{fn}(\mathcal{D}(xuv)) = \mathsf{fn}([\![\mathcal{D}(xuv)]\!]_\pi)$

$$\begin{aligned}
\mathsf{fn}([\![\mathcal{D}(xuv)]\!]_\pi) &= \mathsf{fn}(\mathbf{d}\langle xuv\rangle) && \text{(Definition 3.2.3)}\\
&= \mathsf{fn}(xz.(\overline{u}z \mid \overline{v}z)) && \text{(Definition 3.2.1)}\\
&= \{x, u, v\} && \text{(Definiton 2.1.1)}\\
&= \mathsf{fn}(\mathcal{D}(xuv)) && \text{(Definition 3.1.3)}
\end{aligned}$$

4. $\mathsf{fn}(\mathcal{K}(x)) = \mathsf{fn}([\![\mathcal{K}(x)]\!]_\pi)$

$$\begin{aligned}
\mathsf{fn}([\![\mathcal{K}(x)]\!]_\pi) &= \mathsf{fn}(\mathbf{k}\langle x\rangle) && \text{(Definition 3.2.3)}\\
&= \mathsf{fn}(xz.\mathbf{0}) && \text{(Definition 3.2.1)}\\
&= \{x\} && \text{(Definiton 2.1.1)}\\
&= \mathsf{fn}(\mathcal{K}(x)) && \text{(Definition 3.1.3)}
\end{aligned}$$

And similarly for the remaining base cases.

Inductive cases ($P, Q \in P_{\mathbf{cc}}$ for all cases).

1. $\mathsf{fn}(P \mid Q) = \mathsf{fn}([\![P \mid Q]\!]_\pi)$

$$\begin{aligned}
\mathsf{fn}(P \mid Q) &= \mathsf{fn}(P) \cup \mathsf{fn}(Q) && \text{(Definiton 3.1.3)}\\
&= \mathsf{fn}([\![P]\!]_\pi) \cup \mathsf{fn}([\![Q]\!]_\pi) && \text{(IH)}\\
&= \mathsf{fn}([\![P]\!]_\pi \mid [\![Q]\!]_\pi) && \text{(Definiton 2.1.1)}\\
&= \mathsf{fn}([\![P \mid Q]\!]_\pi) && \text{(Definition 3.3.7)}
\end{aligned}$$

2. $\mathsf{fn}((\boldsymbol{\nu}\, z)\, P) = \mathsf{fn}([\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi)$

$$\begin{aligned}
\mathsf{fn}((\boldsymbol{\nu}\, z)\, P) &= \mathsf{fn}(P) \setminus \{z\} && \text{(Definiton 3.1.3)}\\
&= \mathsf{fn}([\![P]\!]_\pi) \setminus \{z\} && \text{(IH)}\\
&= \mathsf{fn}((\boldsymbol{\nu}\, z)\, [\![P]\!]_\pi) && \text{(Definiton 2.1.1)}\\
&= \mathsf{fn}([\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi) && \text{(Definition 3.3.7)}
\end{aligned}$$

3. $\mathsf{fn}((\boldsymbol{\nu}\, z)\, P) = \mathsf{fn}([\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi)$

$$\begin{aligned}
\mathsf{fn}(!\, P) &= \mathsf{fn}(P) && \text{(Definiton 3.1.3)}\\
&= \mathsf{fn}([\![P]\!]_\pi) && \text{(IH)}\\
&= \mathsf{fn}(![\![P]\!]_\pi) && \text{(Definiton 2.1.1)}\\
&= \mathsf{fn}([\![!P]\!]_\pi) && \text{(Definition 3.3.7)}
\end{aligned}$$

This concludes this proof. □

Proof A.1.2 (ii) We again use induction on the structure of terms as for part (i). The base cases involve comparing the two resulting π-calculus terms from the application of a general substitution σ prior and following the application of the mapping function to each atomic agent and the $\mathbf{0}$ term.

Base cases (names are implicitly universally quantified in each case).

1. $[\![\mathbf{0}\sigma]\!]_\pi =_\alpha [\![\mathbf{0}]\!]_\pi\sigma$

$$\begin{aligned}
[\![\mathbf{0}\sigma]\!]_\pi &= [\![\mathbf{0}]\!]_\pi && \text{(Definition 2.1.2)}\\
&= \mathbf{0} && \text{(Definition 3.2.3)}\\
&= \mathbf{0}\sigma && \text{(Definition 2.1.2)}\\
&= [\![\mathbf{0}]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

2. $[\![\mathcal{M}(xy)\sigma]\!]_\pi =_\alpha [\![\mathcal{M}(xy)]\!]_\pi\sigma$ where $\sigma = \{x', y', \dots/x, y, \dots\}$

$$\begin{aligned}
[\![\mathcal{M}(xy)\sigma]\!]_\pi &= [\![\mathcal{M}(x'y')]\!]_\pi && \text{(Definition 2.1.2)}\\
&= \mathbf{m}\langle x'y'\rangle && \text{(Definition 3.2.3)}\\
&= \mathbf{m}\langle xy\rangle\sigma && \text{(Definition 2.1.2)}\\
&= [\![\mathcal{M}(xy)]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

3. $[\![\mathcal{D}(xuv)\sigma]\!]_\pi =_\alpha [\![\mathcal{D}(xyv)]\!]_\pi\sigma$ where $\sigma = \{x', u', v', \dots/x, u, v, \dots\}$

$$\begin{aligned}
[\![\mathcal{D}(xuv)\sigma]\!]_\pi &= [\![\mathcal{D}(x'u'v')]\!]_\pi && \text{(Definition 2.1.2)}\\
&= \mathbf{d}\langle x'y'v'\rangle && \text{(Definition 3.2.3)}\\
&= \mathbf{d}\langle xuv\rangle\sigma && \text{(Definition 2.1.2)}\\
&= [\![\mathcal{D}(xuv)]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

4. $[\![\mathcal{K}(x)\sigma]\!]_\pi =_\alpha [\![\mathcal{K}(x)]\!]_\pi\sigma$ where $\sigma = \{x', \dots/x, \dots\}$

$$\begin{aligned}
[\![\mathcal{K}(x)\sigma]\!]_\pi &= [\![\mathcal{K}(x')]\!]_\pi && \text{(Definition 2.1.2)}\\
&= \mathbf{k}\langle x'\rangle && \text{(Definition 3.2.3)}\\
&= \mathbf{k}\langle x\rangle\sigma && \text{(Definition 2.1.2)}\\
&= [\![\mathcal{K}(x)]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

And similarly for the remaining base cases.

Inductive cases ($P, Q \in P_{\mathbf{cc}}$ for all cases).

1. $[\![(P \mid Q)\sigma]\!]_\pi =_\alpha [\![P \mid Q]\!]_\pi\sigma$

$$\begin{aligned}
[\![(P \mid Q)\sigma]\!]_\pi &= [\![P\sigma \mid Q\sigma]\!]_\pi && \text{(Definition 3.1.3)}\\
&= [\![P\sigma]\!]_\pi \mid [\![Q\sigma]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P]\!]_\pi\sigma \mid [\![Q]\!]_\pi\sigma && \text{(IH)}\\
&= ([\![P]\!]_\pi \mid [\![Q]\!]_\pi)\sigma && \text{(Definition 2.1.2)}\\
&= ([\![P \mid Q]\!]_\pi)\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

2. $[\![((\boldsymbol{\nu}\, z)\, P)\sigma]\!]_\pi =_\alpha [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi\sigma$

$$\begin{aligned}
[\![((\boldsymbol{\nu}\, z)\, P)\sigma]\!]_\pi &= [\![(\boldsymbol{\nu}\, z)\, (P\sigma)]\!]_\pi && \text{(Definition 3.1.3)}\\
&= (\boldsymbol{\nu}\, z)\, [\![P\sigma]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\, ([\![P]\!]_\pi\sigma) && \text{(IH)}\\
&= ((\boldsymbol{\nu}\, z)\, [\![P]\!]_\pi)\sigma && \text{(Definition 2.1.2)}\\
&= [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

3. $[\![(!\,P)\sigma]\!]_\pi =_\alpha [\![!\,P]\!]_\pi\sigma$

$$\begin{aligned}
[\![(!\,P)\sigma]\!]_\pi &= [\![!\,(P\sigma)]\!]_\pi && \text{(Definition 3.1.3)}\\
&= !\,[\![P\sigma]\!]_\pi && \text{(Definition 3.2.3)}\\
&= !([\![P]\!]_\pi\sigma) && \text{(IH)}\\
&= (!\,[\![P]\!]_\pi)\sigma && \text{(Definition 2.1.2)}\\
&= [\![!\,P]\!]_\pi\sigma && \text{(Definition 3.2.3)}
\end{aligned}$$

This concludes this proof. □

Proof A.1.3 (iii) We give an inductive proof over the axioms of structural congruence. This involves verifying the proposition for each case; we can conclude that if the desired result holds for each axiom then it will hold for the entire relation. We take $P, Q \in P_{\mathbf{cc}}$.

1. $P =_\alpha Q \Rightarrow [\![P]\!]_\pi =_\alpha [\![Q]\!]_\pi$

$$\begin{aligned}
\mathsf{fn}(P) &= \mathsf{fn}(Q) && \text{(Definition 3.1.3)}\\
\Rightarrow \mathsf{fn}([\![P]\!]_\pi) &= \mathsf{fn}([\![Q]\!]_\pi) && \text{(Proposition 3.2.4 (i))}
\end{aligned}$$

2. $P \mid Q \equiv_{\mathbf{cc}} Q \mid P \Rightarrow [\![P \mid Q]\!]_\pi \equiv_\pi [\![Q \mid P]\!]_\pi$

$$\begin{aligned}
[\![P \mid Q]\!]_\pi &= [\![P]\!]_\pi \mid [\![Q]\!]_\pi && \text{(Definition 3.2.3)}\\
&\equiv [\![Q]\!]_\pi \mid [\![P]\!]_\pi && \text{(Definition 2.2.2)}\\
&= [\![Q \mid P]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

3. $P \mid (Q \mid R) \equiv_{\mathbf{cc}} (P \mid Q) \mid R \Rightarrow [\![P \mid (Q \mid R)]\!]_\pi \equiv_\pi [\![(P \mid Q) \mid R]\!]_\pi$

$$\begin{aligned}
[\![P \mid (Q \mid R)]\!]_\pi &= [\![P]\!]_\pi \mid [\![Q \mid R]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P]\!]_\pi \mid [\![Q]\!]_\pi \mid [\![R]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P \mid Q]\!]_\pi \mid [\![R]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![(P \mid Q) \mid R]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

4. $P \mid \mathbf{0} \equiv_{\mathbf{cc}} P \Rightarrow [\![P \mid \mathbf{0}]\!]_\pi \equiv_\pi [\![P]\!]_\pi$

$$\begin{aligned}
[\![P \mid \mathbf{0}]\!]_\pi &= [\![P]\!]_\pi \mid [\![\mathbf{0}]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P]\!]_\pi \mid \mathbf{0} && \text{(Definition 3.2.3)}\\
&\equiv [\![P]\!]_\pi && \text{(Definition 2.2.2)}
\end{aligned}$$

5. $(\boldsymbol{\nu}\, z)\,(\boldsymbol{\nu}\, z)\,P \equiv_{\mathbf{cc}} (\boldsymbol{\nu}\, z)\,P \Rightarrow [\![(\boldsymbol{\nu}\, z)\,(\boldsymbol{\nu}\, z)\,P]\!]_\pi \equiv [\![(\boldsymbol{\nu}\, z)\,P]\!]_\pi$

$$\begin{aligned}
[\![(\boldsymbol{\nu}\, z)\,(\boldsymbol{\nu}\, z)\,P]\!]_\pi &= (\boldsymbol{\nu}\, z)\,[\![(\boldsymbol{\nu}\, z)\,P]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\,(\boldsymbol{\nu}\, z)\,[\![P]\!]_\pi && \text{(Definition 3.2.3)}\\
&\equiv (\boldsymbol{\nu}\, z)\,[\![P]\!]_\pi && \text{(Definition 2.2.2)}\\
&= [\![(\boldsymbol{\nu}\, z)\,P]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

6. $(\boldsymbol{\nu}\, z, w)\, P \equiv_{\mathbf{cc}} (\boldsymbol{\nu}\, w, z)\, P \Rightarrow [\![(\boldsymbol{\nu}\, z, w)\, P]\!]_\pi \equiv [\![(\boldsymbol{\nu}\, w, z)\, P]\!]_\pi$

$$\begin{aligned}
[\![(\boldsymbol{\nu}\, z, w)\, P]\!]_\pi &= (\boldsymbol{\nu}\, z)\, [\![(\boldsymbol{\nu}\, w)\, P]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\, (\boldsymbol{\nu}\, w)\, [\![P]\!]_\pi && \text{(Definition 3.2.3)}\\
&\equiv (\boldsymbol{\nu}\, w)\, (\boldsymbol{\nu}\, z)\, [\![P]\!]_\pi && \text{(Definition 2.2.2)}\\
&= (\boldsymbol{\nu}\, w)\, [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![(\boldsymbol{\nu}\, w, z)\, P]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

7. $(\boldsymbol{\nu}\, z)\, (P \mid Q) \equiv_{\mathbf{cc}} P \mid (\boldsymbol{\nu}\, z)\, Q \Rightarrow [\![(\boldsymbol{\nu}\, z)\, (P \mid Q)]\!]_\pi \equiv [\![P \mid (\boldsymbol{\nu}\, z)\, Q]\!]_\pi$, where $z \notin \mathsf{fn}(P)$ on both sides.

$$\begin{aligned}
[\![(\boldsymbol{\nu}\, z)\, (P \mid Q)]\!]_\pi &= (\boldsymbol{\nu}\, z)\, [\![P \mid Q]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\, ([\![P]\!]_\pi \mid [\![Q]\!]_\pi) && \text{(Definition 3.2.3)}\\
&\equiv [\![P]\!]_\pi \mid (\boldsymbol{\nu}\, z)\, [\![Q]\!]_\pi && \text{(Definition 2.2.2)}\\
&= [\![P]\!]_\pi \mid [\![(\boldsymbol{\nu}\, z)\, Q]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P \mid (\boldsymbol{\nu}\, z)\, Q]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

8. $(\boldsymbol{\nu}\, z)\, \mathbf{0} \equiv_{\mathbf{cc}} \mathbf{0} \Rightarrow [\![(\boldsymbol{\nu}\, z)\, \mathbf{0}]\!]_\pi \equiv [\![\mathbf{0}]\!]_\pi$

$$\begin{aligned}
[\![(\boldsymbol{\nu}\, z)\, \mathbf{0}]\!]_\pi &= (\boldsymbol{\nu}\, z)\, [\![\mathbf{0}]\!]_\pi && \text{(Definition 3.2.3)}\\
&= (\boldsymbol{\nu}\, z)\, \mathbf{0} && \text{(Definition 3.2.3)}\\
&\equiv \mathbf{0} && \text{(Definition 2.2.2)}\\
&= [\![\mathbf{0}]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

9. $!\, P \equiv_{\mathbf{cc}} P \mid !\, P \Rightarrow [\![P]\!]_\pi \equiv_\pi [\![P \mid !\, P]\!]_\pi$

$$\begin{aligned}
[\![P]\!]_\pi &\equiv [\![P]\!]_\pi \mid !\, [\![P]\!]_\pi && \text{(Definition 2.2.2)}\\
&= [\![P]\!]_\pi \mid [\![!\, P]\!]_\pi && \text{(Definition 3.2.3)}\\
&= [\![P \mid !\, P]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

This concludes this proof. □

Proof A.1.4 (iv) We give an inductive proof over the rules of reduction. The base cases consist of verifying that reduction between **cc** atomic agents is preserved by π-reduction. This then justifies the use of the inductive hypothesis in verifying that the proposition holds over the compositional reduction rules.

Base cases (names are implicitly universally quantified in each case).

1. $\mathcal{D}(xuv) \mid \mathcal{M}(xy) \longrightarrow_{\mathbf{cc}} \mathcal{M}(uy) \mid \mathcal{M}(vy) \Rightarrow [\![\mathcal{D}(xuv) \mid \mathcal{M}(xy)]\!]_\pi \longrightarrow_\pi [\![\mathcal{M}(uy) \mid \mathcal{M}(vy)]\!]_\pi$

$$\begin{aligned}
[\![\mathcal{D}(xuv) \mid \mathcal{M}(xy)]\!]_\pi &= [\![\mathcal{D}(xuv)]\!]_\pi \mid [\![\mathcal{M}(xy)]\!]_\pi && \text{(Definition 3.2.3)}\\
&= \mathbf{d}\langle xuv\rangle \mid \mathbf{m}\langle xy\rangle && \text{(Definition 3.2.3)}\\
&= xz.(\overline{u}z \mid \overline{v}z) \mid \overline{x}y && \text{(Definition 3.2.1)}\\
&\longrightarrow \overline{u}y \mid \overline{v}y && \text{(Definition 2.2.5)}\\
&= \mathbf{m}\langle uy\rangle \mid \mathbf{m}\langle vy\rangle && \text{(Definition 3.2.1)}\\
&= [\![\mathcal{M}(uy) \mid \mathcal{M}(vy)]\!]_\pi && \text{(Proposition 3.2.3)}
\end{aligned}$$

2. $\mathcal{K}(x) \mid \mathcal{M}(xy) \longrightarrow_{\mathbf{cc}} \mathbf{0} \Rightarrow [\![\mathcal{K}(x) \mid \mathcal{M}(xy)]\!]_\pi \longrightarrow_\pi [\![\mathbf{0}]\!]_\pi$

$$\begin{aligned}
[\![\mathcal{K}(x) \mid \mathcal{M}(xy)]\!]_\pi &= [\![\mathcal{K}(x)]\!]_\pi \mid [\![\mathcal{M}(xy)]\!]_\pi && \text{(Definition 3.2.3)}\\
&= xz.\mathbf{0} \mid \overline{x}y && \text{(Definition 3.2.3, Definition 3.2.1)}\\
&\longrightarrow \mathbf{0} && \text{(Definition 2.2.5)}\\
&= [\![\mathbf{0}]\!]_\pi && \text{(Proposition 3.2.3)}
\end{aligned}$$

3. $\mathcal{FW}(xu) \mid \mathcal{M}(xy) \longrightarrow_{\mathbf{cc}} \mathcal{M}(uy) \Rightarrow [\![\mathcal{FW}(xu) \mid \mathcal{M}(xy)]\!]_\pi \longrightarrow_\pi [\![\mathcal{M}(uy)]\!]_\pi$

$$\begin{aligned}
[\![\mathcal{FW}(xu) \mid \mathcal{M}(xy)]\!]_\pi &= [\![\mathcal{FW}(xu)]\!]_\pi \mid [\![\mathcal{M}(xy)]\!]_\pi && \text{(Definition 3.2.3)}\\
&= xz.\overline{u}z \mid \overline{x}y && \text{(Definition 3.2.3, Definition 3.2.1)}\\
&\longrightarrow \overline{u}y && \text{(Definition 2.2.5)}\\
&= [\![\mathcal{M}(uy)]\!]_\pi && \text{(Definition 3.2.1, Proposition 3.2.3)}
\end{aligned}$$

4. $\mathcal{B}_l(xu) \mid \mathcal{M}(xy) \longrightarrow_{\mathbf{cc}} \mathcal{FW}(uy) \Rightarrow [\![\mathcal{B}_l(xu) \mid \mathcal{M}(xy)]\!]_\pi \longrightarrow_\pi [\![\mathcal{FW}(yu)]\!]_\pi$

$$\begin{aligned}
[\![\mathcal{B}_l(xu) \mid \mathcal{M}(xy)]\!]_\pi &= [\![\mathcal{B}_l(xu)]\!]_\pi \mid [\![\mathcal{M}(xy)]\!]_\pi && \text{(Definition 3.2.3)}\\
&= xw.wz.\overline{u}z \mid \overline{x}y && \text{(Definition 3.2.3, Definition 3.2.1)}\\
&\longrightarrow yz.\overline{u}z && \text{(Definition 2.2.5)}\\
&= [\![\mathcal{B}_l(yu)]\!]_\pi && \text{(Definition 3.2.1, Proposition 3.2.3)}
\end{aligned}$$

And similarly for the remaining two base cases.

Inductive cases ($P, Q \in P_{\mathbf{cc}}$ for all cases).

1. $P \longrightarrow_{\mathbf{cc}} P' \Rightarrow [\![P \mid Q]\!]_\pi \longrightarrow_\pi [\![P' \mid Q]\!]_\pi$

$$\begin{aligned}
P \longrightarrow_{\mathbf{cc}} P' &\Rightarrow [\![P]\!]_\pi \longrightarrow_\pi [\![P']\!]_\pi && \text{(IH)}\\
&\Rightarrow [\![P]\!]_\pi \mid [\![Q]\!]_\pi \longrightarrow_\pi [\![P']\!]_\pi \mid [\![Q]\!]_\pi && \text{(Definition 2.2.2)}\\
&\Rightarrow [\![P \mid Q]\!]_\pi \longrightarrow_\pi [\![P' \mid Q]\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

2. $P \longrightarrow_{\mathbf{cc}} P' \Rightarrow [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi \longrightarrow_\pi [\![(\boldsymbol{\nu}\, z)\, P']\!]_\pi$

$$\begin{aligned}
P \longrightarrow_{\mathbf{cc}} P' &\Rightarrow [\![P]\!]_\pi \longrightarrow_\pi [\![P']\!]_\pi && \text{(IH)}\\
&\Rightarrow (\boldsymbol{\nu}\, z)\, [\![P]\!]_\pi \longrightarrow_\pi (\boldsymbol{\nu}\, z)\, [\![P']\!]_\pi && \text{(Definition 2.2.2)}\\
&\Rightarrow [\![(\boldsymbol{\nu}\, z)\, P]\!]_\pi \longrightarrow_\pi [\![(\boldsymbol{\nu}\, z)\, P']\!]_\pi && \text{(Definition 3.2.3)}
\end{aligned}$$

3. $P \equiv_{\mathbf{cc}} Q \longrightarrow_{\mathbf{cc}} Q' \equiv_{\mathbf{cc}} P' \Rightarrow [\![P]\!]_\pi \longrightarrow_\pi [\![P']\!]_\pi$

From Proposition 3.2.4 (iv),

- $P \equiv_{\mathbf{cc}} Q \Rightarrow [\![P]\!]_\pi \equiv_\pi [\![Q]\!]_\pi$
- $Q' \equiv_{\mathbf{cc}} P' \Rightarrow [\![Q']\!]_\pi \equiv_\pi [\![P']\!]_\pi$,

and by the induction hypothesis,

$$Q \longrightarrow_{\mathbf{cc}} Q' \Rightarrow [\![Q]\!]_\pi \longrightarrow_\pi [\![Q']\!]_\pi \,.$$

Then, by applying (R-STRUCT),

$$\frac{[\![P]\!]_\pi \equiv_\pi [\![Q]\!]_\pi \longrightarrow_\pi [\![Q']\!]_\pi \equiv_\pi [\![P']\!]_\pi}{[\![P]\!]_\pi \longrightarrow_\pi [\![P']\!]_\pi}$$

This concludes this proof. □

A.2 Proposition 3.3.1

Substitution decomposition

The translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$ performed by the function $[\![\cdot]\!]_\pi$ exhibits the following properties for all $P, Q \in P_{\mathbf{cc}}$.

(i) $\mathbf{c}\langle x^- \tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{c}\langle c\tilde{v}\rangle)$ and $\mathbf{c}\langle \tilde{u}x^+\tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle cx\rangle \mid \mathbf{c}\langle \tilde{u}c^+\tilde{v}\rangle)$ with c fresh.

(ii) $\forall \mathbf{c}\langle\tilde{v}\rangle \in P_{\mathbf{cc}_\pi} : P \approx (\boldsymbol{\nu}\, \tilde{c})\,(\mathbf{c}_1\langle\tilde{v}_1\rangle\sigma_1 \mid \ldots \mid \mathbf{c}_n\langle\tilde{v}_n\rangle\sigma_n\rangle)$ for some $\mathbf{c}_i\langle\tilde{v}_i\rangle \in \mathbf{A}_\pi$ and an injective renaming σ_i, and where each $v \in \tilde{v}_i$ is pairwise distinct.

Proof A.2.1 **($\mathbf{c}\langle x^-\tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{c}\langle c\tilde{v}\rangle)$)** We perform a case by case verification for each $\mathbf{cc}_\pi$ atom. This simply involves showing that the corresponding atomic processes are indeed bisimilar to their substitution decompositions. Where used, the names c, c_1 and c_2 are assumed fresh in all cases.

1. $\mathbf{d}\langle xuv\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle | \mathbf{d}\langle cuv\rangle)$. Let $L \stackrel{\text{def}}{=} xz.(\overline{u}z|\overline{v}z)$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.\overline{c}z|cz.(\overline{u}z|\overline{v}z))$.
   (1) Check $L \approx R$.
   - $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} \overline{u}y \mid \overline{v}y$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{c}y \mid cz.(\overline{u}z \mid \overline{v}z))$

   (2) $L' \approx R'$ as $R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{u}y \mid \overline{u}y)$ and $L' \equiv R''$.

2. $\mathbf{k}\langle x\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{k}\langle c\rangle)$. Let $L \stackrel{\text{def}}{=} xz.\mathbf{0}$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.\overline{c}z \mid cz.\mathbf{0})$.
   (1) Check $L \approx R$.
   - $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} \mathbf{0}$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{c}y \mid cz.\mathbf{0})$

   (2) $L' \approx R'$ as $R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,\mathbf{0}$ and $L' \equiv R''$.

3. $\mathbf{fw}\langle xu\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{fw}\langle cu\rangle)$. Let $L \stackrel{\text{def}}{=} xz.\overline{u}z$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.\overline{c}z \mid cz.\overline{u}z)$.
   (1) Check $L \approx R$.
   - $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} \overline{u}y$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{c}y \mid cz.\overline{u}z)$

   (2) $L' \approx R'$ as $R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,\overline{u}y$ and $L' \equiv R''$.

4. $\mathbf{b}_l\langle xu\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle xc\rangle \mid \mathbf{b}_l\langle cu\rangle)$. Let $L \stackrel{\text{def}}{=} xw.wz.\overline{u}z$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.\overline{c}z \mid cw.wz.\overline{u}z)$.
   (1) Check $L \approx R$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} yz.\overline{u}z$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{c}y \mid cw.wz.\overline{u}z)$

(2) $L' \approx R'$ as $R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(yz.\overline{u}z)$ and $L' \equiv R''$.

And similarly for the remaining two cases. □

Proof A.2.2 $(\mathbf{c}\langle \tilde{u}x^+\tilde{v}\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle cx\rangle \mid \mathbf{c}\langle \tilde{u}c^+\tilde{v}\rangle))$ We take a similarly structured approach to the preceding proof. Again, the names c, c_1 and c_2 are assumed fresh in all cases.

1. $\mathbf{m}\langle xy\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle cx\rangle \mid \mathbf{m}\langle cy\rangle)$. Let $L \stackrel{\text{def}}{=} \overline{x}y$, $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(cz.\overline{x}z \mid \overline{c}y)$.
   (1) $L \approx R$ as $R \longrightarrow_\beta R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{x}y)$ and $L \equiv R'$.

2. $\mathbf{d}\langle xuv\rangle \approx (\boldsymbol{\nu}\, c)\,(\mathbf{fw}\langle cu\rangle|\mathbf{d}\langle xcv\rangle)$. Let $L \stackrel{\text{def}}{=} xz.(\overline{u}z|\overline{v}z)$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(cz.\overline{u}z|xz.(\overline{c}z|\overline{v}z))$.
   (1) Check $L \approx R$.
   - $L \xrightarrow{\overline{x}y} L' \stackrel{\text{def}}{=} \overline{u}y \mid \overline{v}y$ matched by $R \xrightarrow{\overline{x}y} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(cz.\overline{u}z \mid (\overline{c}y \mid \overline{v}y))$

   (2) Check $L' \approx R'$.
   - $L' \xrightarrow{\overline{u}y} L'' \stackrel{\text{def}}{=} \mathbf{0} \mid \overline{v}y$ matched by $R' \stackrel{\text{def}}{=} R'' \xRightarrow{\overline{u}y} (\boldsymbol{\nu}\, c)\,(\mathbf{0} \mid (\mathbf{0} \mid \overline{v}y))$
   - $L' \xrightarrow{\overline{v}y} L''' \stackrel{\text{def}}{=} \overline{u}y \mid \mathbf{0}$ matched by $R' \xrightarrow{\overline{v}y} R'''(\boldsymbol{\nu}\, c)\,(cz.\overline{u}z \mid (\overline{c}y \mid \mathbf{0}))$

   (3) $L'' \approx R''$ as $L'' \equiv R''$.
   (4) $L''' \approx R'''$ as $R''' \longrightarrow_\beta R^{iv}(\boldsymbol{\nu}\, c)\,\overline{u}z$ and $L''' \equiv R^{iv}$.

   The dual case where v is pushed out of the duplicator is proved in the same way.

And similarly for the remaining three cases, ($\mathbf{fw}\langle xy\rangle$, $\mathbf{b}_l\langle xy\rangle$ and $\mathbf{s}\langle xuv\rangle$). □

Proof A.2.3 (ii)

This is a direct consequence of the preceding proof. Firstly, note that every $\mathbf{c}\langle \tilde{v}_i\rangle \in P_{\mathbf{cc}_\pi}$ satisfies at least one of the forms, $\mathbf{c}\langle x^-\tilde{v}\rangle$ and $\mathbf{c}\langle \tilde{u}x^+\tilde{v}\rangle$. Then, by (i), we observe that each $\mathbf{c}\langle \tilde{v}_i\rangle$, including those with repeated parameters, can be expressed up to weak bisimilarity as a parallel composition of forwarders and $\mathbf{c}\langle \tilde{w}_i\rangle$ whose free names are $\mathsf{fn}(\mathbf{c}\langle \tilde{v}_i\rangle)$, and where the parameters of each constituent atomic agent are unique with respect to the agent. As any restricted names generated due to (i) are chosen to be fresh with respect to the entire term, these can be pushed out to the front of the resulting term by structual comgruence. We then have the desired result. □

A.3 Proposition 3.3.4

Properties of input meta-prefix mapping

The input meta-prefix mapping for $\mathbf{cc}_\pi$-terms exhibits the following properties for all $P, Q \in P_{\mathbf{cc}_\pi}$.

(i) $xz.P \approx x^*z.P$.

(ii) $P \approx Q \Rightarrow x^*z.P \approx x^*z.Q$.

(iii) $x^*z.P \mid \mathbf{m}\langle xv\rangle \longrightarrow\!\!\approx P\{^v/_z\}$.

Proof A.3.1 (i) We give an inductive proof over the meta-prefix mapping rules given in Table 3.3. This involves verifying the proposition for each case; we can conclude that if the desired property is preserved by the single application of each individual rule, then it is preserved by the mapping process as a whole.

(I) $xz.(P \mid Q) \approx x^*z.(P \mid Q)$

$$\begin{aligned} x^*z.(P \mid Q) &= (\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid c_1^*z.P \mid c_2^*z.Q) && \text{(Definition 3.3.3)}\\ &= (\boldsymbol{\nu}\, c_1, c_2)\,(xw.(\overline{c_1}w \mid \overline{c_2}w) \mid c_1^*z.P \mid c_2^*z.Q) && \text{(Definition 3.2.1)}\\ &\approx (\boldsymbol{\nu}\, c_1, c_2)\,(xw.(\overline{c_1}w \mid \overline{c_2}w) \mid c_1z.P \mid c_2z.Q) && \text{(IH, Lemma 2.4.16)} \end{aligned}$$

Let $L \stackrel{\text{def}}{=} xz.(P \mid Q)$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(xw.(\overline{c_1}w \mid \overline{c_2}w) \mid c_1z.P \mid c_2z.Q)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} P' \mid Q'$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(\overline{c_1}y \mid \overline{c_2}y \mid c_1z.P \mid c_2z.Q)$, where $P \stackrel{\text{def}}{=} P\{^y/_z\}$ and $Q \stackrel{\text{def}}{=} Q\{^y/_z\}$.

(2) $L' \approx R'$ as $R' \longrightarrow_\beta^* R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(P' \mid Q')$ and $L' \equiv R'$.

(II) $xz.(\boldsymbol{\nu}\, c')\,P \approx x^*z.(\boldsymbol{\nu}\, c')\,P$

$$\begin{aligned} x^*z.(\boldsymbol{\nu}\, c')\,P &= (\boldsymbol{\nu}\, c)\,x^*z.P' && (P' \stackrel{\text{def}}{=} P\{^c/_{c'}\}\text{, Definition 3.3.3})\\ &\approx (\boldsymbol{\nu}\, c)\,xz.P' && \text{(IH, Lemma 2.4.16)} \end{aligned}$$

(1) Check $L \stackrel{\text{def}}{=} xz.(\boldsymbol{\nu}\, c')\,P \approx R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,xz.P'$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c')\,P''$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,P'''$, where $P'' \stackrel{\text{def}}{=} P\{^y/_z\}$ and $P''' \stackrel{\text{def}}{=} P'\{^y/_z\}$

(2) $L' \approx R'$ as $L' \equiv R'$.

(III) $xz.\mathbf{0} \approx x^*z.\mathbf{0}$

$$\begin{aligned} x^*z.\mathbf{0} &= \mathbf{k}\langle x\rangle && \text{(Definition 3.3.3)}\\ &= xz.\mathbf{0} && \text{(Definition 3.2.1)} \end{aligned}$$

(IV) $xz.!\,P \approx x^*z.!\,P$

First show that $Q \stackrel{\text{def}}{=} c^*z.(P \mid \mathbf{m}\langle cz\rangle) \approx cz.(P \mid \mathbf{m}\langle cz\rangle)$.

$$\begin{aligned} Q &= (\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{d}\langle cc_1c_2\rangle \mid c_1^*z.P \mid c_2^*z.\mathbf{m}\langle cz\rangle) && \text{(Definition 3.3.3)}\\ &= (\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{d}\langle cc_1c_2\rangle \mid c_1^*z.P \mid \mathbf{fw}\langle c_2c\rangle) && \text{(Definition 3.3.3)}\\ &\approx (\boldsymbol{\nu}\, c_1, c_2)\,(\mathbf{d}\langle cc_1c_2\rangle \mid c_1z.P \mid \mathbf{fw}\langle c_2c\rangle) && \text{(IH, Lemma 2.4.16)} \end{aligned}$$

Let $P' \stackrel{\text{def}}{=} P\{^y/_z\}$. Note that $(!\,P)\{^y/_z\} =!\,P'$ by Definition 2.1.2.

(1) Check $L \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(cz.(\overline{c_1}z \mid \overline{c_2}z) \mid c_1 z.P \mid c_2 z.\bar{c}z) \approx R \stackrel{\text{def}}{=} cz.(P \mid \bar{c}z)$

- $L \xrightarrow{cy} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,((\overline{c_1}y \mid \overline{c_2}y) \mid c_1 z.P \mid c_2 z.\bar{c}z)$ matched by $R \xrightarrow{cy} R' \stackrel{\text{def}}{=} P' \mid \bar{c}y$

(2) $L' \approx R'$ as $R' \longrightarrow_\beta^* R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c_1, c_2)\,(P' \mid \bar{c}y)$ and $L' \equiv R''$.

An immediate consequence of this result is

$$(\boldsymbol{\nu}\, c)\,(!\, c^* z.(P \mid \mathbf{m}\langle cz\rangle) \mid \mathbf{m}\langle cy\rangle) \approx (\boldsymbol{\nu}\, c)\,(!\, cz.(P \mid \mathbf{m}\langle cz\rangle) \mid \mathbf{m}\langle cy\rangle) \quad \text{(Lemma 2.4.16)}$$

The next goal is to show $(\boldsymbol{\nu}\, c)\,(!\, cz.(P \mid \mathbf{m}\langle cz\rangle) \mid \mathbf{m}\langle cy\rangle) \approx !\, P'$, where $P' \stackrel{\text{def}}{=} P\{y/z\}$. Let $L \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(!\, cz.(P \mid \bar{c}z) \mid \bar{c}y)$ and $R \stackrel{\text{def}}{=} !\, P'$.

(1) Check $L \approx R$.

- $L \xrightarrow{\tau} L' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(P' \mid \bar{c}y \mid !\, cz.(P \mid \bar{c}z) \mid \mathbf{0})$ matched by $R \Longrightarrow R$
- $R \xrightarrow{l \neq \tau} R' \stackrel{\text{def}}{=} P'' \mid !\, P'$ matched by $L \stackrel{l}{\Longrightarrow} L'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(P'' \mid \bar{c}y \mid !\, cy.(P \mid \bar{c}z))$
- $R \xrightarrow{\tau} R'' \stackrel{\text{def}}{=} P''' \mid !\, P'$ matched by $L \Longrightarrow L''' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(P''' \mid \bar{c}y \mid !\, cy.(P \mid \bar{c}z))$

(2) Check $L' \approx R$. However,

$$\begin{aligned} L &\longrightarrow_\beta (\boldsymbol{\nu}\, c)\,(P' \mid \bar{c}y \mid !\, cz.(P \mid \bar{c}z)) && \text{(Definition 3.2.6)} \\ &\equiv L' && \text{(Proposition 2.2.2)} \end{aligned}$$

And so, $L \approx L'$ by Proposition 2.4.14 and Proposition 3.2.7.

(3) $R' \approx L''$ by the IH and congruence of $\approx$ as $R' \equiv P'' \mid R$ and $L'' \equiv P'' \mid L$.

(4) $R'' \approx L'''$ by the IH and congruence of $\approx$ as $R' \equiv P''' \mid R$ and $L''' \equiv P''' \mid L$.

We have now shown $(\boldsymbol{\nu}\, c)\,(!\, cz.(P \mid \bar{c}z) \mid \bar{c}y) \approx !\, P'$. Observe that,

$$\begin{aligned} x^* z.!\, P &= (\boldsymbol{\nu}\, c)\,(xw.\bar{c}w \mid !\, c^* z.(P \mid \bar{c}z)) && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\xrightarrow{xy} (\boldsymbol{\nu}\, c)\,(\bar{c}y \mid !\, cz.(P \mid \bar{c}z)) && \text{(Definition 2.3.2)} \end{aligned}$$

which, together with $xz.!\, P \xrightarrow{xy} !\, P'$, implies $xz.P \approx x^* z.!\, P$.

(v) $xz.\mathbf{c}\langle v^+ \tilde{w}\rangle \approx x^* z.\mathbf{c}\langle v^+ \tilde{w}\rangle$

$\mathbf{m}\langle vw\rangle$ is the only $\mathbf{cc}_\pi$-atom of the form $\mathbf{c}\langle v^+ \tilde{w}\rangle$.

$$x^* z.\mathbf{m}\langle vw\rangle = (\boldsymbol{\nu}\, c)\,(\mathbf{s}\langle xcv\rangle \mid \mathbf{m}\langle cw\rangle) \qquad \text{(Definition 3.3.3)}$$

Let $L \stackrel{\text{def}}{=} xz.\bar{v}w$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.ct.\bar{v}t \mid \bar{c}w)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} \bar{v}w$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(ct.\bar{v}t \mid \bar{c}w)$

(2) $L' \approx R'$ as $R' \longrightarrow_\beta R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,\bar{v}w$ and $L' \equiv R''$.

(VI) $xz.\mathbf{c}\langle v^{-}\tilde{w}\rangle \approx x^{*}z.\mathbf{c}\langle v^{-}\tilde{w}\rangle$

By mechanical verification each of the six cases. We demonstrate the case for the Duplicator, and the remaining five cases follow similarly.

1. $x^{*}z.\mathbf{d}\langle vw_1w_2\rangle = (\boldsymbol{\nu}\, c)\,(\mathbf{s}\langle xvc\rangle \mid \mathbf{d}\langle cw_1w_2\rangle)$ (Definition 3.3.3)
   Let $L \stackrel{\text{def}}{=} xz.vs.(\overline{w_1}s \mid \overline{w_2}s)$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(xz.vt.\overline{c}t \mid cz.(\overline{w_1}z \mid \overline{w_2}z))$.

   (1) Check $L \approx R$.

   - $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} vs.(\overline{w_1}s \mid \overline{w_2}s)$ matched by $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(vt.\overline{c}t \mid cz.(\overline{w_1}z \mid \overline{w_2}z))$

   (2) Check $L' \approx R'$.

   - $L' \xrightarrow{vy} L'' \stackrel{\text{def}}{=} \overline{w_1}y \mid \overline{w_2}y$ matched by $R' \xrightarrow{vy} R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{c}y \mid cz.(\overline{w_1}z \mid \overline{w_2}z))$

   (3) $L'' \approx R''$ as $R'' \longrightarrow_{\beta} R''' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c)\,(\overline{w_1}y \mid \overline{w_2}y)$ and $L'' \equiv R'''$.

(VII) $xz.\mathbf{m}\langle vz\rangle \approx x^{*}z.\mathbf{m}\langle vz\rangle$

$$\begin{aligned} x^{*}z.\mathbf{m}\langle vz\rangle &= \mathbf{fw}\langle xv\rangle && \text{(Definition 3.3.3)}\\ &= xz.\mathbf{m}\langle vz\rangle && \text{(Definition 3.2.1)}\end{aligned}$$

(VIII) $xz.\mathbf{fw}\langle zv\rangle \approx x^{*}z.\mathbf{fw}\langle zv\rangle$

$$\begin{aligned} x^{*}z.\mathbf{fw}\langle zv\rangle &= \mathbf{b}_l\langle xv\rangle && \text{(Definition 3.3.3)}\\ &= xz.\mathbf{fw}\langle zv\rangle && \text{(Definition 3.2.1)}\end{aligned}$$

(IX) $xz.\mathbf{fw}\langle vz\rangle \approx x^{*}z.\mathbf{fw}\langle vz\rangle$

$$\begin{aligned} x^{*}z.\mathbf{fw}\langle vz\rangle &= \mathbf{b}_r\langle xv\rangle && \text{(Definition 3.3.3)}\\ &= xz.\mathbf{fw}\langle vz\rangle && \text{(Definition 3.2.1)}\end{aligned}$$

(X) $xz.\mathbf{c}\langle \tilde{w_1}z^{+}\tilde{w_2}\rangle \approx x^{*}z.\mathbf{c}\langle \tilde{w_1}z^{+}\tilde{w_2}\rangle$

$$\begin{aligned} x^{*}z.\mathbf{c}\langle \tilde{w_1}z^{+}\tilde{w_2}\rangle &= (\boldsymbol{\nu}\, c)\, x^{*}z.(\mathbf{fw}\langle cz\rangle \mid \mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle) && \text{(Definition 3.3.3)}\\ &= (\boldsymbol{\nu}\, c, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid c_1^{*}z.\mathbf{fw}\langle cz\rangle \mid c_2^{*}z.\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle) && \text{(Definition 3.3.3)}\\ &= (\boldsymbol{\nu}\, c, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid c_2^{*}z.\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle) && \text{(Definition 3.3.3)}\\ &= (\boldsymbol{\nu}\, c, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid c_2z.\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle) && \text{(IH, Proposition 2.4.16)}\end{aligned}$$

We can then verify that $xz.\mathbf{c}\langle \tilde{w_1}z^{+}\tilde{w_2}\rangle \approx (\boldsymbol{\nu}\, c, c_1, c_2)\,(\mathbf{d}\langle xc_1c_2\rangle|\mathbf{b}_r\langle c_1c\rangle|c_2z.\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle)$ for all $\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle$. However, it is possible to reduce the amount of work required using β-equality.

Let $L \stackrel{\text{def}}{=} xz.\mathbf{c}\langle \tilde{w_1}z^{+}\tilde{w_2}\rangle$ and $R \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_1, c_2)\,(xz.(\overline{c_1}z|\overline{c_2}z)|c_1z_1.cz_2.\overline{z_1}z_2|c_2z.\mathbf{c}\langle \tilde{w_1}c^{+}\tilde{w_2}\rangle)$.

(1) Check $L \approx R$.

- $L \xrightarrow{xy} L' \stackrel{\text{def}}{=} \mathbf{c}\langle \tilde{w_1} z^+ \tilde{w_2} \rangle \{y/z\}$ matched by
  $R \xrightarrow{xy} R' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_1, c_2)\,((\overline{c_1} y \mid \overline{c_2} y) \mid c_1 z_1.cz_2.\overline{z_1} z_2 \mid c_2 z.\mathbf{c}\langle \tilde{w_1} c^+ \tilde{w_2} \rangle)$

(2) Check $L' \approx R'$. However,

$$R' \longrightarrow_\beta^* R'' \stackrel{\text{def}}{=} (\boldsymbol{\nu}\, c, c_1, c_2)\,(cz.\overline{y} z \mid \mathbf{c}\langle \tilde{w_1} c^+ \tilde{w_2} \rangle)\ .$$

$L' \approx R''$ can then be verified for each case. Note that $L' \equiv \mathbf{c}\langle \tilde{w_1} y^+ \tilde{w_2} \rangle$ for all cases.

(XI) $xz.\mathbf{c}\langle z^- \tilde{w} \rangle \approx x^* z.\mathbf{c}\langle z^- \tilde{w} \rangle$

Similar to the preceding rule.

(XII) $xz.\mathbf{b}_r\langle vz^- \rangle \approx x^* z.\mathbf{b}_r\langle vz^- \rangle$

$$x^* z.\mathbf{b}_r\langle vz \rangle = (\boldsymbol{\nu}\, c_1, c_2, c_3)\, x^* z.(\mathbf{d}\langle vc_1c_2 \rangle \mid \mathbf{s}\langle c_1 z c_3 \rangle \mid \mathbf{b}_r\langle c_2 c_3 \rangle) \quad \text{(Definition 3.3.3)}$$

Firstly, Definition 3.3.3 is used to expand out the meta-prefix by applying (I) twice, (VI) twice, (XIII), ... Bisimilarity between the resulting term and $xz.\mathbf{b}_r\langle vz \rangle$ can then be verified (the proof is ommitted due to length).

(XIII) $xz.\mathbf{s}\langle uz^- v \rangle \approx x^* z.\mathbf{s}\langle uz^- v \rangle$

Similar to the preceding rule.

This concludes this proof. □

Proof A.3.2 (ii) This is an immediate consequence of the preceding proof. Assume $P \approx Q$.

$$\begin{aligned} P \approx Q &\Rightarrow xz.P \approx xz.Q && \text{(Lemma 2.4.16)} \\ &\Rightarrow x^* z.P \approx x^* z.Q && \text{(Proposition 3.3.4 (i), Definition 2.4.1)} \end{aligned}$$

This concludes this proof. □

Proof A.3.3 (iii) We again give an inductive proof over the meta-prefix mapping rules as for (i), applying the reduction to the expanded prefix-terms before verifying the bisimilarity result. Let $P' \stackrel{\text{def}}{=} P\{y/z\}$ and $Q' \stackrel{\text{def}}{=} Q\{y/z\}$.

(I) $x^* z.(P \mid Q) \mid \mathbf{m}\langle xy \rangle \longrightarrow\approx (P \mid Q)\{y/z\}$

$$\begin{aligned} x^* z.(P \mid Q) \mid \mathbf{m}\langle xy \rangle &= (\boldsymbol{\nu}\, c_1, c_2)\,(xw.(\overline{c_1} w \mid \overline{c_2} w) \mid c_1^* z.P \mid c_2^* z.Q) \mid \overline{x} y \\ &\qquad \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\longrightarrow (\boldsymbol{\nu}\, c_1, c_2)\,(\overline{c_1} y \mid \overline{c_2} y \mid c_1^* z.P \mid c_2^* z.Q) \qquad \text{(Definition 2.2.5)} \\ &\approx (\boldsymbol{\nu}\, c_1, c_2)\,(\overline{c_1} y \mid \overline{c_2} y \mid c_1 z.P \mid c_2 z.Q) \\ &\qquad \text{(Proposition 3.3.4 (i), Lemma 2.4.16)} \\ &\approx (P \mid Q)\{y/z\} \qquad \text{(Definition 2.2.2, Definition 2.1.2)} \end{aligned}$$

(II) $x^*z.(\boldsymbol{\nu}\, c')\, P \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx (\boldsymbol{\nu}\, c')\, P'$

$$\begin{aligned} x^*z.(\boldsymbol{\nu}\, c)\, P \mid \mathbf{m}\langle xy\rangle &= (\boldsymbol{\nu}\, c')\, x^*z.P\{c/c'\} \mid \overline{x}y && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\equiv (\boldsymbol{\nu}\, c)\, (x^*z.P\{c/c'\} \mid \overline{x}y) && \text{(Definition 2.2.2)} \\ &\longrightarrow\!\approx (\boldsymbol{\nu}\, c)\, (P\{c/c'\}\{y/z\}) && \text{(IH, Lemma 2.4.16)} \\ &\approx (\boldsymbol{\nu}\, c')\, (P') && \text{(Definition 2.2.2, Proposition 2.4.14)} \end{aligned}$$

(III) $x^*z.\mathbf{0} \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{0}\{y/z\}$

$$\begin{aligned} x^*z.\mathbf{0} \mid \mathbf{m}\langle xy\rangle &= xz.\mathbf{0} \mid \overline{x}y && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\longrightarrow \mathbf{0}\{y/z\} && \text{(Definition 2.2.5)} \end{aligned}$$

(IV) $x^*z.!\, P \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx !\, P\{y/z\}$

This result was proved as part of the corresponding case in the proof of (i).

(V) $x^*z.\mathbf{c}\langle v^+\tilde{w}\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{c}\langle v^+\tilde{w}\rangle\{y/z\}$

$$\begin{aligned} x^*z.\mathbf{c}\langle v^+\tilde{w}\rangle \mid \mathbf{m}\langle xy\rangle &= (\boldsymbol{\nu}\, c)\, (xz.cz.\overline{v}z \mid \mathbf{c}\langle c^+\tilde{w}\rangle) \mid \overline{x}y && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\longrightarrow (\boldsymbol{\nu}\, c)\, (cz.\overline{v}z \mid \mathbf{c}\langle c^+\tilde{w}\rangle) && \text{(Definition 2.2.5)} \end{aligned}$$

However, $\mathbf{m}\langle vw\rangle$ is the only $\mathbf{cc}_\pi$-atom of the form $\mathbf{c}\langle c^+\tilde{w}\rangle$. By Definition 3.2.6 and Definition 2.1.2,

$$(\boldsymbol{\nu}\, c)\, (cz.\overline{v}z \mid \mathbf{m}\langle cw\rangle) \approx \mathbf{m}\langle vw\rangle\{y/z\}\ .$$

(VI) $x^*z.\mathbf{c}\langle v^-\tilde{w}\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{c}\langle v^-\tilde{w}\rangle\{y/z\}$

$$\begin{aligned} x^*z.\mathbf{c}\langle v^-\tilde{w}\rangle \mid \mathbf{m}\langle xy\rangle &= (\boldsymbol{\nu}\, c)\, (xz.vz.\overline{c}z \mid \mathbf{c}\langle c^-\tilde{w}\rangle) \mid \overline{x}y && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\longrightarrow (\boldsymbol{\nu}\, c)\, (vz.\overline{c}z \mid \mathbf{c}\langle c^-\tilde{w}\rangle) && \text{(Definition 2.2.5)} \end{aligned}$$

We can then verify that $(\boldsymbol{\nu}\, c)\, (vz.\overline{c}z \mid \mathbf{c}\langle c^-\tilde{w}\rangle) \approx \mathbf{c}\langle v^-\tilde{w}\rangle\{y/z\}$ for each of the six cases. Note that $\mathbf{c}\langle v^-\tilde{w}\rangle\{y/z\} = \mathbf{c}\langle v^-\tilde{w}\rangle$ for all cases.

(VII) $x^*z.\mathbf{m}\langle vz\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{m}\langle vz\rangle\{y/z\}$

$$\begin{aligned} x^*z.\mathbf{m}\langle vz\rangle \mid \mathbf{m}\langle xy\rangle &= xz.\overline{v}z \mid \overline{x}y && \text{(Definition 3.3.3, Definition 3.2.1)} \\ &\longrightarrow \overline{v}z\{y/z\} && \text{(Definition 2.2.5)} \end{aligned}$$

(VIII) $x^*z.\mathbf{fw}\langle zv\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{fw}\langle zv\rangle\{y/z\}$

Similar to rule (VII).

(IX) $x^*z.\mathbf{fw}\langle vz\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{fw}\langle vz\rangle\{y/z\}$

Similar to rule (VII).

(X) $x^*z.\mathbf{c}\langle \tilde{w}_1 z^+ \tilde{w}_2\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\!\approx \mathbf{c}\langle \tilde{w}_1 z^+ \tilde{w}_2\rangle\{y/z\}$

$$\begin{aligned}
x^*z.\mathbf{c}\langle\tilde{w_1}z^+\tilde{w_2}\rangle \mid \mathbf{m}\langle xy\rangle &= (\boldsymbol{\nu}\, c)\, x^*z.(\mathbf{fw}\langle cz\rangle \mid \mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle) \mid \mathbf{m}\langle xy\rangle \\
&= (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{d}\langle xc_1c_2\rangle \mid c_1^*z.\mathbf{fw}\langle cz\rangle \mid c_2^*z.\mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle) \mid \mathbf{m}\langle xy\rangle \\
&= (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{d}\langle xc_1c_2\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid c_2^*z.\mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle) \mid \mathbf{m}\langle xy\rangle \\
&\longrightarrow (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{m}\langle c_1y\rangle \mid \mathbf{m}\langle c_2y\rangle \mid \mathbf{b}_r\langle c_1c\rangle \mid c_2^*z.\mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle) \\
&\longrightarrow_\beta (\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{m}\langle c_2y\rangle \mid \mathbf{fw}\langle cy\rangle \mid c_2^*z.\mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle)
\end{aligned}$$

We can then verify that $(\boldsymbol{\nu}\, c, c_1, c_2)\, (\mathbf{m}\langle c_2y\rangle|\mathbf{fw}\langle cy\rangle|c_2^*z.\mathbf{c}\langle\tilde{w_1}c^+\tilde{w_2}\rangle) \approx \mathbf{c}\langle\tilde{w_1}z^+\tilde{w_2}\rangle\{y/z\}$ for all $\mathbf{c}\langle\tilde{w_1}z^+\tilde{w_2}\rangle$. Note that the cases where $z \in \tilde{w_2}$ and $z \notin \tilde{w_2}$ must be considered separately.

(XI) $x^*z.\mathbf{c}\langle z^-\tilde{w}\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\approx \mathbf{c}\langle z^-\tilde{w}\rangle\{y/z\}$

Similar to the preceding rule.

(XII) $x^*z.\mathbf{b}_r\langle vz^-\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\approx \mathbf{b}_r\langle vz^-\rangle\{y/z\}$

Firstly, Definition 3.3.3 is used to expand out the meta-prefix by applying (XII), (I) twice, (VI) twice, (XIII), ..., and the resulting term can be reduced, consuming $\mathbf{m}\langle xy\rangle$. Bisimilarity between the reduced term and $x^*z.\mathbf{b}_r\langle vz^-\rangle\{y/z\}$ can then be verified.

(XIII) $x^*z.\mathbf{s}\langle uz^-v\rangle \mid \mathbf{m}\langle xy\rangle \longrightarrow\approx \mathbf{s}\langle uz^-v\rangle\{y/z\}$

Similar to the preceding rule.

This concludes this proof. □

A.4 Lemma 3.3.8

properties of $[\![\cdot]\!]_{cc_\pi}$

The translation from P_π to $P_{\mathbf{cc}_\pi}$ by the function $[\![\cdot]\!]_{cc_\pi}$ exhibits the following properties for all $P, Q \in P_\pi$.

(i) $P \approx [\![P]\!]_{cc_\pi}$
(ii) $P \equiv Q \Rightarrow [\![P]\!]_{cc_\pi} \approx [\![Q]\!]_{cc_\pi}$.
(iii) $P \longrightarrow Q \Rightarrow [\![P]\!]_{cc_\pi} \longrightarrow\approx [\![Q]\!]_{cc_\pi}$.

Proof A.4.1 (i) We use structural induction over the syntax of the π-calculus.

Base cases (names are implicitly universally quantified in each case).

1. $\overline{x}z \approx [\![\overline{x}z]\!]_{cc_\pi}$

$$\begin{aligned}
[\![\overline{x}y]\!]_{cc_\pi} &= \mathbf{m}\langle xy\rangle && \text{(Definition 3.3.7)} \\
&= \overline{x}y && \text{(Definition 3.2.1)}
\end{aligned}$$

2. $\mathbf{0} \approx [\![\mathbf{0}]\!]_{cc_\pi}$

$$[\![\mathbf{0}]\!]_{cc_\pi} = \mathbf{0} \qquad \text{(Definition 3.3.7)}$$

Inductive cases ($P, Q \in P_{\mathbf{cc}_\pi}$ in all cases).

1. $P \mid Q \approx [\![P \mid Q]\!]_{cc_\pi}$

$$\begin{aligned} P \mid Q &\approx [\![P]\!]_{cc_\pi} \mid [\![Q]\!]_{cc_\pi} && \text{(IH)} \\ &= [\![P \mid Q]\!]_{cc_\pi} && \text{(Definition 3.3.7)} \end{aligned}$$

2. $(\boldsymbol{\nu}\, z)\, P \approx [\![(\boldsymbol{\nu}\, z)\, P]\!]_{cc_\pi}$

$$\begin{aligned} (\boldsymbol{\nu}\, z)\, P &\approx (\boldsymbol{\nu}\, z)\, [\![P]\!]_{cc_\pi} && \text{(IH, Definition 2.4.1)} \\ &= [\![(\boldsymbol{\nu}\, z)\, P]\!]_{cc_\pi} && \text{(Definition 3.3.7)} \end{aligned}$$

3. $xz.P \approx [\![xz.P]\!]_{cc_\pi}$

$$\begin{aligned} xz.P &\approx xz.[\![P]\!]_{cc_\pi} && \text{(IH, Definition 2.4.1)} \\ &\approx x^*z.[\![P]\!]_{cc_\pi} && \text{(Proposition 3.3.4)} \\ &= [\![xz.P]\!]_{cc_\pi} && \text{(Definition 3.3.7)} \end{aligned}$$

4. $!\, P \approx [\![!\, P]\!]_{cc_\pi}$

$$\begin{aligned} !\, P &\approx !\, [\![P]\!]_{cc_\pi} && \text{(IH, Definition 2.4.1)} \\ &= [\![!\, P]\!]_{cc_\pi} && \text{(Definition 3.3.7)} \end{aligned}$$

This concludes this proof. □

Proof A.4.2 (ii) One possible method would be to proceed as in Proof A.1.3; in fact, the proofs would be indentical. However, we can show this result more directly using Lemma 3.3.8 in a similar style to Proof A.3.2.

$$\begin{aligned} P \equiv Q &\Rightarrow P \approx Q && \text{(Proposition 2.4.14)} \\ &\Rightarrow [\![P]\!]_{cc_\pi} \approx [\![Q]\!]_{cc_\pi} && \text{(Definition 2.4.1, Lemma 3.3.8)} \end{aligned}$$

This concludes this proof. □

List of Tables

2.1 Application of substitution. 9
2.2 The axioms of structural congruence. 10
2.3 The rules of π-reduction. 11
2.4 The rules of synchronous early transition. 13

3.1 Reduction rules for **cc** terms. 26
3.2 Translation from $P_{\mathbf{cc}}$ to $P_{\mathbf{cc}_\pi}$. 28
3.3 Input meta-prefix mapping for $\mathbf{cc}_\pi$ terms. 32
3.4 Translation from P_π to $P_{\mathbf{cc}_\pi}$. 34

List of Figures

1.1 Combinator interaction. 3
1.2 Combinator interaction with name hiding. 4
1.3 Interaction of **cc** combinators embedded into the π-calculus. 5
1.4 Encoding of the **cc** within the π-calculus and vice versa. 6

2.1 The relationship between $\dot{\cong}$, $\approx$, $\approx_g$ and $\equiv$ on the asynchronous π-calculus. . 21

Bibliography

[1] J. Roger Hindley and Jonathan P. Seldin. *Introduction to combinators and λ-calculus*. Cambridge University Press, New York, NY, USA, 1986.

[2] Kohei Honda and Nobuko Yoshida. Combinatory representation of mobile processes. In *Conference Record of POPL '94: 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 348–360, Portland, Oregon, 1994.

[3] Kohei Honda and Nobuko Yoshida. Replication in concurrent combinators. In *Theoretical Aspects of Computer Software*, pages 786–805, 1994.

[4] Cosimo Laneve and Björn Victor. Solos in concert. *Lecture Notes in Computer Science*, 1644:513–539, 1999.

[5] Sergio Maffeis. Models of concurrent computation part ii lecture notes, 2005.

[6] Robin Milner. *Communication and Concurrency*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1995.

[7] Joachim Parrow and Bjorn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Logic in Computer Science*, pages 176–185, 1998.

[8] Davide Sangiorgi and David Walker. *PI-Calculus: A Theory of Mobile Processes*. Cambridge University Press, New York, NY, USA, 2001.

[9] Nobuko Yoshida. Minimality and separation results on asynchronous mobile processes: Representability theorems by concurrent combinators (extended abstract). In *International Conference on Concurrency Theory*, pages 131–146, 1998.