An Introduction to the π-Calculus

Chapter to appear in *Handbook of Process Algebra*, ed. Bergstra, Ponse and Smolka, Elsevier

Joachim Parrow[*]
Dep. Teleinformatics,
Royal Institute of Technology, Stockholm

Abstract

The π-calculus is a process algebra where processes interact by sending communication links to each other. This paper is an overview of and introduction to its basic theory. We explore the syntax, semantics, equivalences and axiomatisations of the most common variants.

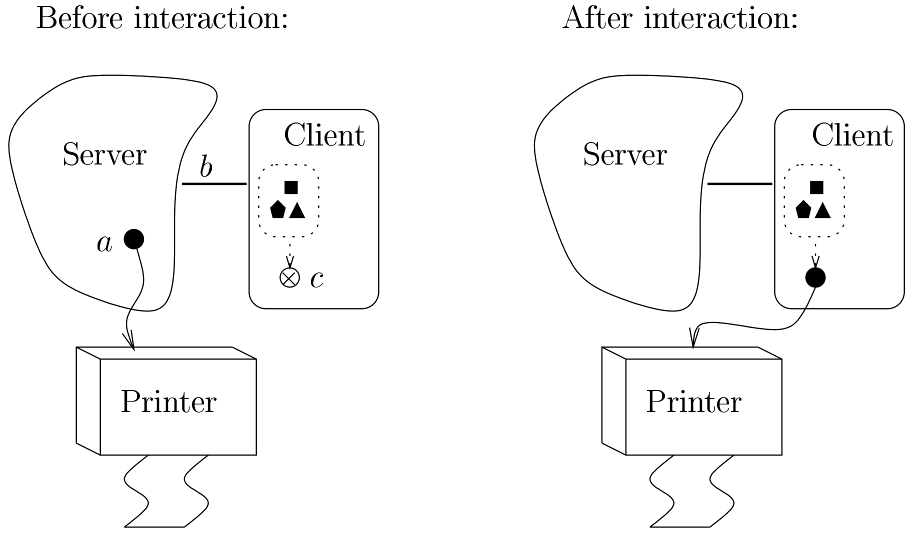
[*]email `joachim@it.kth.se`

Contents

1 Introduction

The π-calculus is a mathematical model of processes whose interconnections change as they interact. The basic computational step is the transfer of a communication link between two processes; the recipient can then use the link for further interaction with other parties. This makes the calculus suitable for modelling systems where the accessible resources vary over time. It also provides a significant expressive power since the notions of access and resource underlie much of the theory of concurrent computation, in the same way as the more abstract and mathematically tractable concept of a function underlies functional computation. This introduction to the π-calculus is intended for a theoretically inclined reader who knows a little about the general principles of process algebra and who wishes to learn the fundamentals of the calculus and its most common and stable variants.

Let us first consider an example. Suppose a server controls access to a printer and that a client wishes to use it. In the original state only the server itself has access to the printer, represented by a communication link a. After an interaction with the client along some other link b this access to the printer has been transferred:



In the π-calculus this is expressed as follows: the server that sends a along b is $\overline{b}a\,.\,S$; the client that receives some link along b and then uses it to send data along it is $b(c)\,.\,\overline{c}d\,.\,P$. The interaction depicted above is formulated

$$\overline{b}a\,.\,S \mid b(c)\,.\,\overline{c}d\,.\,P \quad \xrightarrow{\tau} \quad S \mid \overline{a}d\,.\,P$$

We see here that a plays two different roles. In the interaction between the server and the client it is an object transferred from one to the other. In a further interaction between the client and the printer it is the name of the communication link. The idea that the names of the links belong to the same category as the transferred objects is one of the cornerstones of the calculus, and is one way in which it is different from other process algebras. In the example a, b, c, d are all just *names* which intuitively represent access rights: a accesses the printer, b accesses the server, d accesses some data, and c is a placeholder for an access to arrive along a. If a is the only way to access the printer then we can say that the printer "moves" to the client, since after the interaction nothing else can access it. For this reason the π-calculus has been called a calculus of "mobile" processes. But the calculus is much more general than that. The printer may have many links that make it do different things, and the server can send these links to different clients to establish different access capabilities to a shared resource.

At first sight it appears as if the π-calculus is just a specialised form of a value-passing process algebra where the values are links. In such a comparison the calculus may be thought rather poor since there are no data types and no functions defined on the names; the transferable entities are simple atomic things without any internal structure. The reason that the π-calculus nevertheless is considered more expressive is that it admits migrating local scopes. This important point deserves an explanation here.

Most process algebras have a way to declare a communication link local to a set of processes. For example in CCS the fact that P and Q share a private port a is symbolised by $(P|Q)\backslash a$, where the operator $\backslash a$ is called *restriction* on a. The significance is that no other process can use the local link a, as if it were a name distinct from all other names in all processes.

In the π-calculus this restriction is written $(\boldsymbol{\nu} a)(P|Q)$. It is similar in that no other process can use a immediately as a link to P or Q. The difference is that the name a is also a transferable object and as such can be sent, by P or Q, to another process which then can use the restricted link. Returning to the example above suppose that a is a local link between the server and the printer. Represent the printer by R, then this is captured by $(\boldsymbol{\nu} a)(\overline{b}a\,.\,S \mid R)$. The server is still free to send a along b to the client. The result would be a private link shared between all three processes, but still distinct from any other name in any other process, and the transition is consequently written

$$(\boldsymbol{\nu} a)(\overline{b}a\,.\,S \mid R) \;\mid\; b(c)\,.\,\overline{c}d\,.\,P \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} a)(S \mid R \mid \overline{a}d\,.\,P)$$

So, although the transferable objects are simple atomic things they can also be declared local with a defined scope, and in this way the calculus transcends the ordinary value-passing process algebras. This is also the main source of difficulty in the development of the theory because the scope of an object, as represented by the operands of its restriction, must migrate with the object as it is transferred between processes.

The π-calculus is far from a single well defined body of work. The central idea, a process algebraic definition of link-passing, has been developed in several directions to accommodate specific applications or to determine the effects of various semantics. Proliferation is certainly a healthy sign for any scientific area although it poses problems for those who wish to get a quick overview. Presumably some readers new to the π-calculus will be satisfied with a compact presentation of a single version, while other may be interested in the spectrum of variations.

This paper aims to serve both these needs. In the following, the even-numbered sections develop a single strand of the calculus. Section 2 presents the syntax and give some small examples of how it is used. In Section 4 we proceed to the semantics in its most common form as a labelled transition system. In Section 6 we consider one of the main definitions of bisimulation and the congruence it induces, and in Section 8 we look at their axiomatisations through syntactic equalities of agents. These sections do not depend on the odd-numbered sections and can be considered as a basic course of the calculus. There will be full definitions and formulations of the central results, and sketches that explain the ideas and structure of the proofs.

Each odd-numbered section presents variations on the material in the preceding one. Thus, in Section 3 we explore different versions of the calculus, such as the effect of varying the operators, and the asynchronous, polyadic, and higher-order calculus. Section 5 treats alternative ways to define the semantics, with different versions of labelled and unlabelled transitions. Section 7 defines a few other common bisimulation equivalences (the π-calculus, like any process algebra, boasts a wide variety of equivalences but in this paper we concentrate on the aspects particular to π), and their axiomatisations are treated in Section 9. In these sections we do not always get a full formal account, but hopefully enough explanations that the reader will gain an understanding of the basic ideas. Finally, Section 10 contains references to other work. We give a brief account of how the calculus evolved and mention other overviews and introductory papers. We also indicate sources for the material treated in this paper.

It must be emphasised that there are some aspects of the π-calculus we do not treat at all, such as modal logics, analysis algorithms, implementations, and ways to use the calculus to model concurrent systems and languages. Also, the different variants can be combined in many ways, giving rise to a large variety of calculi. I hope that after this introduction a reader can explore the field with some confidence.

2 The π-Calculus

We begin with a sequence of definitions and conventions. The reader who makes it to Section 2.3 will be rewarded with small but informative examples.

2.1 Basic Definitions

We assume a potentially infinite set of *names* $\mathcal{N}$, ranged over by $a, b, \ldots, z$, which will function as all of communication ports, variables and data values, and a set of *(agent) identifiers* ranged over by A, each with a fixed nonnegative arity. The *agents*, ranged over by $P, Q, \ldots$ are defined Table 1. From that table we see that the agents can be of the following forms:

1. The empty agent $\mathbf{0}$, which cannot perform any actions.
2. An *Output Prefix* $\overline{a}x\,.\,P$. The intuition is that the name x is sent along the name a and thereafter the agent continues as P. So $\overline{a}$ can be thought of as an output port and x as a datum sent out from that port.
3. An *Input Prefix* $a(x)\,.\,P$, meaning that a name is received along a name a, and x is a placeholder for the received name. After the input the agent will continue as P but with the newly received name replacing x. So a can be thought of as an input port and x as a variable which will get its value from the input along a.
4. A *Silent Prefix* $\tau\,.\,P$, which represents an agent that can evolve to P without interaction with the environment. We use α, β to range over $a(x)$, $\overline{a}x$ and τ and call them *Prefixes*, and we say that $\alpha\,.\,P$ is a *Prefix form*, or sometimes just *Prefix* when this cannot cause confusion.
5. A *Sum* $P + Q$ representing an agent that can enact either P or Q.
6. A *Parallel Composition* $P \mid Q$, which represents the combined behaviour of P and Q executing in parallel. The components P and Q can act independently, and may also communicate if one performs an output and the other an input along the same port.
7. A *Match* $\texttt{if}\ x = y\ \texttt{then}\ P$. As expected this agent will behave as P if x and y are the same name, otherwise it does nothing.
8. A *Mismatch* $\texttt{if}\ x \neq y\ \texttt{then}\ P$. This agent will behave as P if x and y are *not* the same name, otherwise it does nothing.
9. A *Restriction* $(\boldsymbol{\nu} x)P$. This agent behaves as P but the name x is local, meaning it cannot immediately be used as a port for communication between P and its environment. However, it can be used for communication between components within P.

Prefixes	α	$::=$	$\overline{a}x$	Output
			$a(x)$	Input
			τ	Silent
Agents	P	$::=$	$\mathbf{0}$	Nil
			$\alpha\,.\,P$	Prefix
			$P + P$	Sum
			$P \mid P$	Parallel
			$\texttt{if}\ x = y\ \texttt{then}\ P$	Match
			$\texttt{if}\ x \neq y\ \texttt{then}\ P$	Mismatch
			$(\boldsymbol{\nu} x)P$	Restriction
			$A(y_1, \ldots, y_n)$	Identifier
Definitions			$A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$	(where $i \neq j \Rightarrow x_i \neq x_j$)

Table 1: The syntax of the π-calculus.

10. An *Identifier* $A(y_1, \ldots, y_n)$ where n is the arity of A. Every Identifier has a *Definition* $A(x_1, \ldots, x_n) \stackrel{\mathrm{def}}{=} P$ where the x_i must be pairwise distinct, and the intuition is that $A(y_1, \ldots, y_n)$ behaves as P with y_i replacing x_i for each i. So a Definition can be thought of as a process declaration, $x_1, \ldots, x_n$ as formal parameters, and the Identifier $A(y_1, \ldots, y_n)$ as an invocation with actual parameters $y_1, \ldots, y_n$.

The operators are familiar from other process algebras so we shall in the following concentrate on some important aspects particular to the π-calculus, trusting the reader to be confident with the more general principles.

The forms Nil, Sum and Parallel have exactly the same meaning and use as in other process algebras, and the Prefix forms are as in the algebras that admit value-passing. The `if` constructs Match and Mismatch may appear limited in comparison with value-passing algebras which usually admit arbitrary Boolean expressions (evaluating to either true or false). But on closer consideration it is apparent that combinations of Match and Mismatch are the only possible tests that can be performed in the π-calculus: the objects transmitted are just names and these have no structure and no operators are defined on them, so the only thing we can do is compare names for equality. We can combine such tests conjunctively by nesting them, for example

$$\texttt{if}\ x = y\ \texttt{then}\ \texttt{if}\ u \neq v\ \texttt{then}\ P$$

behaves as P if both $x = y$ and $u \neq v$ hold. We can combine them disjunctively by using Sum, for example

$$\texttt{if}\ x = y\ \texttt{then}\ P + \texttt{if}\ u \neq v\ \texttt{then}\ P$$

behaves as P if at least one of $x = y$ and $u \neq v$ hold. Sometimes we shall use a binary conditional

$$\texttt{if}\ x = y\ \texttt{then}\ P\ \texttt{else}\ Q$$

as an abbreviation for $\texttt{if}\ x = y\ \texttt{then}\ P + \texttt{if}\ x \neq y\ \texttt{then}\ Q$.

As in other algebras we say that P is *guarded* in Q if P is a proper subterm of a Prefix form in Q. Also, the input Prefix $a(x) \,.\, P$ is said to *bind* x in P, and occurrences of x in P are then called *bound*. In contrast the output Prefix $\overline{a}x \,.\, P$ does not bind x. These Prefixes are said to have *subject* a and *object* x, where the object is called *free* in the output Prefix and *bound* in the input Prefix. The silent Prefix τ has neither subject nor object.

The Restriction operator $(\boldsymbol{\nu} x)P$ also binds x in P. Its effect is as in other algebras (where it is written $\backslash x$ in CCS and δ_x in ACP) with one significant difference. In ordinary process algebras the things that are restricted are port names and these cannot be transmitted between agents. Therefore the restriction is static in the sense that the scope of a restricted name does not need to change when an agent executes. In the π-calculus there is no difference between "port names" and "values", and a name that represents a port can indeed be transmitted between agents. If that name is restricted the scope of the restriction must change, as we shall see, and indeed almost all of the increased complexity and expressiveness of the π-calculus over value-passing algebras come from the fact that restricted things move around. The reader may also think of $(\boldsymbol{\nu} x)P$ as "new x in P", by analogy with the object-oriented use of the word "new", since this construct can be thought of as declaring a new and hitherto unused name, represented by x for the benefit of P.

In summary, both input Prefix and Restriction bind names, and we can define the *bound names* $\texttt{bn}(P)$ as those with a bound occurrence in P and the *free names* $\texttt{fn}(P)$ as those with a not bound occurrence, and similarly $\texttt{bn}(\alpha)$ and $\texttt{fn}(\alpha)$ for a Prefix α. We sometimes write $\texttt{fn}(P, Q)$ to mean $\texttt{fn}(P) \cup \texttt{fn}(Q)$, and just α for $\texttt{fn}(\alpha) \cup \texttt{bn}(\alpha)$ when it is apparent that it represents a set of names, such as in "$x \in \alpha$". In a Definition $A(x_1, \ldots, x_n) \overset{\text{def}}{=} P$ we assume that $\texttt{fn}(P) \subseteq \{x_1, \ldots, x_n\}$. In some examples we shall elide the parameters of Identifiers and Definitions when they are unimportant or can be inferred from context.

A *substitution* is a function from names to names. We write $\{x/y\}$ for the substitution that maps y to x and is identity for all other names, and in general $\{x_1 \ldots x_n / y_1 \ldots y_n\}$, where the y_i are pairwise distinct, for a function that maps each y_i to x_i. We use σ to range over substitutions, and sometimes write $\tilde{x}$ for a sequence of names when the length is unimportant or can be inferred from context. The agent $P\sigma$ is P where all free names x are replaced by $\sigma(x)$, with alpha-conversion wherever needed to avoid captures. This means that bound names are renamed such that whenever x is replaced by $\sigma(x)$ then the so obtained occurrence of $\sigma(x)$ is free. For example,

$$(a(x) \,.\, (\boldsymbol{\nu} b)\overline{x}b \,.\, \overline{c}y \,.\, \mathbf{0})\{xb/yc\} \quad \text{is} \quad a(z) \,.\, (\boldsymbol{\nu} d)\overline{z}d \,.\, \overline{b}x \,.\, \mathbf{0}$$

A process algebra fan may have noticed that one common operator is not present in the π-calculus: that of relabelling (in CCS written $[a/b]$). The primary use of relabelling is to define instances of agents from other agents, for example, if B is a buffer with ports i and o then $B[i'/i, o'/o]$ is a buffer with ports i' and o'. In the π-calculus we will instead define instances through the parameters of the Identifiers, so for example a buffer with ports i and o is $B(i, o)$, and with ports i' and o' it is $B(i', o')$. For injective relabellings this is just another style of specification which allows us to economise on one operator. (A reader familiar with the CCS relabelling should be warned that it has the same effect as port substitution only if injective. In general they are different.)

Finally some notational conventions: A sum of several agents $P_1 + \cdots + P_n$ is written $\sum_{i=1}^{n} P_i$, or just $\sum_j P_j$ when n is unimportant or obvious, and we here allow the case $n = 0$ when the sum means $\mathbf{0}$. A sequence of distinct Restrictions $(\boldsymbol{\nu} x_1) \cdots (\boldsymbol{\nu} x_n)P$ is often abbreviated to $(\boldsymbol{\nu} x_1 \cdots x_n)P$. In a Prefix we sometimes elide the object if it is not important, so $a \,.\, P$ means $a(x) \,.\, P$ where x is a name that is never used, and similarly for output. And we sometimes elide a trailing $\mathbf{0}$, writing α for the agent $\alpha \,.\, \mathbf{0}$, where this cannot cause confusion. We give the unary operators precedence over the binary and $|$ precedence over $+$, so for example $(\boldsymbol{\nu} x)P \mid Q + R$ means $(((\boldsymbol{\nu} x)P) \mid Q) + R$.

2.2 Structural Congruence

The syntax of agents is in one sense too concrete. For example, the agents $a(x) \,.\, \overline{b}x$ and $a(y) \,.\, \overline{b}y$ are syntactically different, although they only differ in the choice of bound name and therefore intuitively represent the same behaviour: an agent that inputs something along a and then sends that along $\overline{b}$. As another example the agents $P|Q$ and $Q|P$ represent the same thing: a parallel composition of the agents P and Q. Our intuition about parallel composition is that it is inherently unordered, and we are forced to syntactically distinguish between $P|Q$ and $Q|P$ only because our language is linear.

We therefore introduce a *structural congruence* to identify the agents which intuitively represent the same thing. It should be emphasised that this has nothing to do with the traditional behavioural equivalences in process algebra which are defined in terms of the behaviour exhibited by an agent under some operational semantics. We have yet to define a semantics, and the structural congruence identifies only agents where it is immediately obvious from their *structure* that they are the same.

The structural congruence $\equiv$ is defined as the smallest congruence satisfying the following laws:

1. If P and Q are variants of alpha-conversion then $P \equiv Q$.
2. The Abelian monoid laws for Parallel: commutativity $P|Q \equiv Q|P$, associativity $(P|Q)|R \equiv P|(Q|R)$, and $\mathbf{0}$ as unit $P|\mathbf{0} \equiv P$; and the same laws for Sum.
3. The unfolding law $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \overset{\mathrm{def}}{=} P$.
4. The scope extension laws

$$\begin{array}{lcll}
(\boldsymbol{\nu} x)\mathbf{0} & \equiv & \mathbf{0} & \\
(\boldsymbol{\nu} x)(P \mid Q) & \equiv & P \mid (\boldsymbol{\nu} x)Q & \text{if } x \notin \mathtt{fn}(P) \\
(\boldsymbol{\nu} x)(P + Q) & \equiv & P + (\boldsymbol{\nu} x)Q & \text{if } x \notin \mathtt{fn}(P) \\
(\boldsymbol{\nu} x)\mathtt{if}\ u = v\ \mathtt{then}\ P & \equiv & \mathtt{if}\ u = v\ \mathtt{then}\ (\boldsymbol{\nu} x)P & \text{if } x \neq u \text{ and } x \neq v \\
(\boldsymbol{\nu} x)\mathtt{if}\ u \neq v\ \mathtt{then}\ P & \equiv & \mathtt{if}\ u \neq v\ \mathtt{then}\ (\boldsymbol{\nu} x)P & \text{if } x \neq u \text{ and } x \neq v \\
(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P & \equiv & (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P &
\end{array}$$

Table 2: The definition of structural congruence.

The reader will here correctly object that "represent the same thing" and "immediately obvious" are not formally defined concepts, and indeed several different versions of the structural congruence can be found in the literature; there is no canonical definition and each has different merits. In Section 5.1 we will meet some of them and explore their consequences. Until then we adopt a particular structural congruence. The definition is given in Table 2. We briefly comment on the clauses in the definition.

1. Alpha-conversion, i.e., choice of bound names, identifies agents like $a(x) \,.\, \overline{b}x$ and $a(y) \,.\, \overline{b}y$.
2. The Abelian monoid laws mean that Parallel and Sum are unordered. For example, when we think of a composition of three agents P,Q,R it does not matter if we write it as $(P|Q)|R$ or $(R|Q)|P$. The same holds for Sum. The fact that $\mathbf{0}$ is a unit means that $P|\mathbf{0} \equiv P$ and $P + \mathbf{0} \equiv P$, something which follows from the intuition that $\mathbf{0}$ is empty and therefore contributes nothing to a Parallel composition or Sum.
3. The unfolding just says that an Identifier is the same as its Definition, with the appropriate parameter instantiation.
4. The scope extension laws come from our intuition that $(\boldsymbol{\nu} x)P$ just says that x is a new unique name in P; it can be thought of as marking the occurrences

of x in P with a special colour saying that this is a local name. It then does not really matter where the symbols "$(\boldsymbol{\nu} x)$" are placed as long as they mark the same occurrences. For example, in $\mathbf{0}$ there are no occurrences so the Restriction can be removed at will. In Parallel composition, if all occurrences are in one of the components then it does not matter if the Restriction covers only that component or the whole composition.

Note that we do *not* have that $(\boldsymbol{\nu} x)(P \mid Q) \equiv (\boldsymbol{\nu} x)P \mid (\boldsymbol{\nu} x)Q$. The same occurrences are restricted in both agents, but in $(\boldsymbol{\nu} x)(P \mid Q)$ they are restricted by the *same* binder (or if you will, coloured by the same colour), meaning that P and Q can interact using x, in contrast to the situation in $(\boldsymbol{\nu} x)P \mid (\boldsymbol{\nu} x)Q$.

Through a combination of these laws we get that $(\boldsymbol{\nu} x)P \equiv P$ if $x \notin \mathtt{fn}(P)$:

$$P \equiv P \mid \mathbf{0} \equiv P \mid (\boldsymbol{\nu} x)\mathbf{0} \equiv (\boldsymbol{\nu} x)(P \mid \mathbf{0}) \equiv (\boldsymbol{\nu} x)P$$

So as a special case we get $(\boldsymbol{\nu} x)(\boldsymbol{\nu} x)P \equiv (\boldsymbol{\nu} x)P$ for all P.

Another key fact is that all unguarded Restrictions can be pulled out to the top level of an agent:

Proposition 1 *Let P be an agent where $(\boldsymbol{\nu} x)Q$ is an unguarded subterm. Then P is structurally congruent to an agent $(\boldsymbol{\nu} x')P'$ where P' is obtained from P by replacing $(\boldsymbol{\nu} x)Q$ with $Q\{x'/x\}$, for some name x' not occurring in P.*

The proof is by alpha-converting all bound names so that they become syntactically distinct, and then applying scope extension (from right to left) to move the Restriction to the outermost level. This corresponds to the intuition that instead of declaring something as local it can be given a syntactically distinct name: the effect is the same in that nothing else can access the name.

Our scope extension laws are in fact chosen precisely such that Proposition 1 holds. For example, we have not given any scope extension law for Prefixes and can therefore only pull out unguarded Restrictions. The reader may have expected a law like $(\boldsymbol{\nu} x)\alpha\,.\,P \equiv \alpha\,.\,(\boldsymbol{\nu} x)P$ for $x \notin \alpha$. Indeed such a law would be sound, in the sense that it conforms to intuition and does not disrupt any of the results in this paper, and it will hold for the behavioural equivalences explored later in sections 6 and 7. But it will not be necessary at this point, in particular it is not necessary to prove Proposition 1.

Structural congruence is much stronger, i.e., identifies fewer agents, than any of the behavioural equivalences. The structural congruence is used in the definition of the operational semantics, which in turn is used to define the behavioural equivalences. The main technical reasons for taking this route are that many of the following definitions and explanations become simpler and that we get a uniform treatment for those variants of the calculus that actually require a structural congruence. In Section 5.1 we comment on the possibility to define the calculus without a structural congruence.

2.3 Simple Examples

Although we shall not present the operational semantics just yet (a reader who wishes to look at it now will find it in Section 4) it might be illuminating to see some examples of the scope migration mentioned in Section 1, that Restrictions move with their objects. Formally, scope migration is a consequence of three straightforward postulates. The first is the usual law for inferring interactions between parallel components. This is present in most process algebras and implies that

$$a(x)\,.\,\overline{c}x \mid \overline{a}b \quad \xrightarrow{\tau} \quad \overline{c}b \mid \mathbf{0}$$

or in general

$$a(x)\,.\,P \mid \overline{a}b\,.\,Q \quad \xrightarrow{\tau} \quad P\{b/x\} \mid Q$$

The second postulate is that Restrictions do not affect silent transitions. $P \xrightarrow{\tau} Q$ represents an interaction between the components of P, and a Restriction $(\boldsymbol{\nu} x)P$ only restricts interactions between P and its environment. Therefore $P \xrightarrow{\tau} Q$ implies $(\boldsymbol{\nu} x)P \xrightarrow{\tau} (\boldsymbol{\nu} x)Q$. The third postulate is that structurally congruent agents should never be distinguished and thus any semantics must assign them the same behaviour. Now what are the implications for restricted objects? Suppose that b is a restricted name, i.e., that we are considering a composition

$$a(x)\,.\,\overline{c}x \mid (\boldsymbol{\nu} b)\overline{a}b$$

Will there be an interaction between the components and if so what should it be? Structural congruence gives the answer, because b is not free in the left hand component so the agent is by scope extension structurally congruent to

$$(\boldsymbol{\nu} b)(a(x)\,.\,\overline{c}x \mid \overline{a}b)$$

and this agent has a transition between the components: because of

$$a(x)\,.\,\overline{c}x \mid \overline{a}b \quad \xrightarrow{\tau} \quad \overline{c}b \mid \mathbf{0}$$

we get that

$$(\boldsymbol{\nu} b)(a(x)\,.\,\overline{c}x \mid \overline{a}b) \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} b)(\overline{c}b \mid \mathbf{0})$$

and the rightmost $\mathbf{0}$ can be omitted by the monoid laws. So by identifying structurally congruent agents we obtain that

$$a(x)\,.\,\overline{c}x \mid (\boldsymbol{\nu} b)\overline{a}b \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} b)\overline{c}b$$

or in general that, provided $b \notin \mathtt{fn}(P)$,

$$a(x)\,.\,P \mid (\boldsymbol{\nu} b)\overline{a}b\,.\,Q \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} b)(P\{b/x\} \mid Q)$$

In other words, the scope of $(\boldsymbol{\nu} b)$ “moves” with b from the right hand component to the left. This phenomenon is sometimes called scope extrusion. If $b \in \mathtt{fn}(P)$ a similar interaction is possible by first alpha-converting the bound b to some name $b' \notin \mathtt{fn}(P)$, and we would get

$$a(x) \,.\, P \mid (\boldsymbol{\nu} b)\overline{a}b \,.\, Q \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} b')(P\{b'/x\} \mid Q\{b'/b\})$$

So $P\{b'/x\}$ still contains b free and it is not the same as the received restricted name b'.

For another example consider:

$$((\boldsymbol{\nu} b)a(x) \,.\, P) \mid \overline{a}b \,.\, Q$$

Here the right hand component has a free b which should not be the same as the bound b to the left. Is there an interaction between the components? We cannot immediately extend the scope to the right hand component since it has b free. But we can first alpha-convert the bound b to some new name b' and then extend the scope to obtain

$$(\boldsymbol{\nu} b')(a(x) \,.\, P\{b'/b\} \mid \overline{a}b \,.\, Q)$$

and it is clear that we have a transition

$$(\boldsymbol{\nu} b')(a(x) \,.\, P\{b'/b\} \mid \overline{a}b \,.\, Q) \xrightarrow{\tau} (\boldsymbol{\nu} b')P\{b'/b\}\{b/x\} \mid Q$$

So the restricted name, now b', will still be local to the left hand component; the attempt to intrude the scope is thwarted by an alpha-conversion. In summary, through alpha-conversion and scope extension we can send restricted names as objects, and Restrictions will always move with the objects and never include free occurrences of that name.

This ability to send scopes along with restricted names is what makes the calculus convenient for modelling exchange of private resources. For example, suppose we have an agent R representing a resource, say a printer, and that it is controlled by a server S which distributes access rights to R. In the simplest case the access right is just to execute R. This can be modelled by introducing a new name e as a trigger, and guarding R by that name, as in

$$(\boldsymbol{\nu} e)(S \mid e \,.\, R)$$

Here R cannot execute until it receives a signal on e. The server can invoke it by performing an action $\overline{e}$, but moreover, the server can send e to a client wishing to use R. For example, suppose that a client Q needs the printer. It asks S along some predetermined channel c for the access key, here e, to R, and only upon receipt of this key can R be executed. We have

$$c(x) \,.\, \overline{x} \,.\, Q \mid (\boldsymbol{\nu} e)(\overline{c}e \,.\, S \mid e \,.\, R) \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} e)(\overline{e} \,.\, Q \mid S \mid e \,.\, R) \quad \xrightarrow{\tau} \quad (\boldsymbol{\nu} e)(Q \mid S \mid R)$$

The first transition means that Q receives an access to R and the second that this access is used. We can informally think of this as if the agent R is transmitted (represented by its key e) from S to Q, so in a sense this gives us the power of a higher-order communication where the objects are agents and not only names. But our calculus is more general since a server can send e to many clients, meaning that these will share R (rather than receiving separate copies of R). And R can have several keys that make it do different things, for example R can be $e_1 . R_1 \mid e_2 . R_2 \cdots$, and the server can send only some of the keys to clients and retain some for itself, or send different keys to different clients representing different access privileges.

A related matter is if S wishes to send two names d and e to a client, and insure that the same client receives both names. If there are several clients then the simple solution of transmitting d and e along predetermined channels may mean that one client receives d and another e. A better solution is to first establish a private channel with a client and then send d and e along that channel. The private channel is simply a restricted name:

$$(\boldsymbol{\nu} p)\overline{c}p . \overline{p}d . \overline{p}e . S$$

A client interacting with C must be prepared to receive a name, and then along that name receive d and e:

$$c(p) . p(x) . p(y) . Q$$

Now, even if we have a composition with several clients and a server, the only possibility is that d and e end up with the same client. This feature is so common that we introduce an abbreviation for it:

$$\begin{array}{lll} \overline{c}\langle e_1 \cdots e_n \rangle . P & \text{means} & (\boldsymbol{\nu} p)\overline{c}p . \overline{p}e_1 . \cdots . \overline{p}e_n . P \\ c(x_1 \cdots x_n) . Q & \text{means} & c(p) . p(x_1) . \cdots . p(x_n) . Q \end{array}$$

where we choose $p \notin \mathtt{fn}(P, Q)$ and all x_i are pairwise distinct. We will then have

$$\overline{c}\langle e_1 \cdots e_n \rangle . P \mid c(x_1 \cdots x_n) . Q \ \xrightarrow{\tau} \cdots \xrightarrow{\tau} \ P \mid Q\{e_1 \ldots e_n / x_1 \ldots x_n\}$$

The idea to establish private links in this way has many other uses. Suppose for example that Q wishes to execute P by transmitting on its trigger e, and then also wait until P has completed execution. One way to represent this is to send to P a private name for signalling completion, as in

$$(\boldsymbol{\nu} r)\overline{e}r . r . Q \mid e(x) . P \ \xrightarrow{\tau} \ (\boldsymbol{\nu} r)(r . Q \ \mid \ P\{r/x\})$$

Here Q must wait until someone signals on r before continuing. This someone can only be P since no other is in the scope of r. This scheme is quite general, for example P can delegate to another agent the task to restart Q, by sending r to it as an object in an interaction.

The π-calculus has been used to succinctly describe many aspects of concurrent and functional programming, and also of high-level system description where mobility plays an important role. We shall not attempt an overview of all applications here. In the rest of this paper we concentrate on some central aspects of the theory of the calculus.

3 Variants of the Calculus

The calculus can be varied in many ways. There are many useful subcalculi which imply a somewhat simpler theory, and we shall also briefly consider two popular extensions: the polyadic and the higher-order calculi.

3.1 Match and Mismatch

The Match and Mismatch operators are absent in many presentations of the calculus. This means a certain loss of expressiveness. But in a sense, equality test of names can be implemented without Match. Consider the following typical use of a test: an agent P receives a name and continues as Q if the name is y and as R if the name is z:

$$P \quad = \quad a(x)\,.\,(\texttt{if}\ x = y\ \texttt{then}\ Q \quad + \quad \texttt{if}\ x = z\ \texttt{then}\ R)$$

Without the Match a similar effect can be achieved exploiting the Parallel operator, where a communication is possible only if the subjects of the input and output actions are the same:

$$P \quad = \quad a(x)\,.\,(\overline{x} \quad | \quad (y\,.\,Q \quad + \quad z\,.\,R))$$

If the received name is y then, after reception, a communication between the first and second parallel component is possible, after which Q can execute. Similarly, if it is z then a communication between the first and third component enables R. This will work provided no other agent can interfere by interacting on the names y and z. So it is not a general encoding of Match, although many specific instances can be emulated in this way. Consider for example an implementation of Boolean values: There are agents $True_a$ and $False_a$, emitting Boolean values along a, and an agent $Case_a(Q, R)$, receiving a Boolean along a and enacting Q or R depending on the value of the Boolean. A straightforward encoding would use two special names t and f:

$$\begin{array}{lcl} \mathit{True}_a & = & \overline{a}t \\ \mathit{False}_a & = & \overline{a}f \\ \mathit{Case}_a(Q,R) & = & a(x)\,.\,(\texttt{if}\ x = t\ \texttt{then}\ Q \ + \ \texttt{if}\ x = f\ \texttt{then}\ R) \end{array}$$

So e.g. $\mathit{Case}_a(Q,R) \mid \mathit{True}_a \xrightarrow{\tau} \texttt{if}\ t = t\ \texttt{then}\ Q \ + \ \texttt{if}\ t = f\ \texttt{then}\ R$, and this agent will behave as Q. The same effect can be achieved without Match, at the expense of an extra communication. The idea is that *Case* emits two new names and *True* and *False* respond by signalling on one of them:

$$\begin{array}{lcl} \mathit{True}_a & = & a(xy)\,.\,\overline{x} \\ \mathit{False}_a & = & a(xy)\,.\,\overline{y} \\ \mathit{Case}_a(Q,R) & = & (\boldsymbol{\nu} xy)\overline{a}\langle xy\rangle\,.\,(x\,.\,Q \ + \ y\,.\,R) \end{array}$$

where x and y do not occur in Q or R. It will now hold that $\mathit{Case}_a(Q,R) \mid \mathit{True}_a$ evolves to

$$(\boldsymbol{\nu} xy)(\overline{x} \ \mid \ (x\,.\,Q \ + \ y\,.\,R))$$

and the *only* possible continuation, in any environment, is a τ to Q, since x and y are local. In this way many instances of the Match construct, including all that aim to capture tests over ordinary data types, can be encoded.

The Mismatch `if` $x \neq y$ `then` cannot be implemented in a similar way. Omitting Mismatch may be thought to drastically reduce the expressive power but it turns out the effect is not very severe. For example, if the possible values of a certain data type are u, v and w, the Mismatch `if` $x \neq u$ `then` P can be replaced by `if` $x = v$ `then` $P +$ `if` $x = w$ `then` P. In practice, the data types where equality test is admitted almost always have a predetermined finite range, or can be modelled through a finite set of constructors. Mismatch also makes some portions of the theory a little more complicated. For these reasons many of the versions of the π-calculus do not use it. However, it appears to be necessary for many axiomatisations (cf. Sections 8 and 9).

3.2 Sum

The Sum operator $+$ is sometimes regarded as unrealistic from an implementation perspective. The problem is that it represents a form of synchronous global choice. Consider:

$$(a\,.\,P_1 + \overline{b}\,.\,P_2) \mid (\overline{a}\,.\,Q_1 + b\,.\,Q_2)$$

Clearly this agent can evolve to either $P_1|Q_1$ (through a communication along a) or to $P_2|Q_2$ (along b). The choice of which path should be taken involves both parallel components, and is resolved synchronously. If parallel components are distant then this needs a non-trivial protocol, and it can therefore be argued that the general form of Sum is not a realistic primitive operator.

There are two main reasons for including the Sum operator. One is that it admits representations of automata in a straightforward way. Automata have proved useful for high-level descriptions of communicating systems and are present in many modelling languages. The basic idea is that the behaviour of a component is thought of as a directed graph where the nodes represent the reachable states and the edges, labelled by actions, represent the possibilities to move between states. This can be represented in the π-calculus as follows: For each state choose a unique Identifier, and introduce the Definition

$$A \overset{\text{def}}{=} \sum_{i=1}^{n} \alpha_i\,.\,A_i$$

for each state A where the outgoing edges are labelled $\alpha_1, \ldots, \alpha_n$ leading to states $A_1, \ldots, A_n$.

The other reason is related to axiomatisations of the behavioural equivalences (see Section 8). All axiomatisations so far use a version of the expansion law, which replaces Parallel by Sum, as in

$$a \mid b = a \,.\, b + b \,.\, a$$

No complete axiomatisation is known for a calculus with Parallel and without Sum.

In situations where high-level modelling is not called for and where complete axiomatisation is not an issue, variants of the π-calculus without Sum have been used, for example to describe the semantics of programming languages. Apart from omitting the computationally questionable primitive of synchronous global choice, which would make an implementation of the calculus difficult, it simplifies the theory to only have one binary operator.

Many presentations of the calculus use *guarded Sums* in place of Sum, meaning that the operands of the Sum must be Prefix forms. The general format of a guarded sum is $\sum_{i=1}^{n} \alpha_i \,.\, P_i$ (i.e., it is an n-ary operator). So for example $a \,.\, P + \overline{b} \,.\, Q$ is a guarded Sum, but $a \,.\, P + (Q|R)$ is not. With guarded Sum some parts of the theory become simpler, notably the weak bisimulation equivalence is a congruence (see Section 7.4) for the same reason as in ordinary process algebra. The loss of expressiveness is not dramatic: guarded Sum is sufficient both for representing automata and for formulating the expansion theorem. But using n-ary operators for arbitrarily large n may be considered awkward by those who seek a minimal theory. Note that an n-ary guarded Sum cannot be reduced to a sequence of binary guarded Sums, so the n-ary operators are actually necessary for all n. In contrast, for the ordinary sum, $\sum_{i=1}^{n} P_i$ is defined as $n-1$ binary Sums.

It is interesting that some cases of guarded Sum can be emulated by the Parallel operator. Consider for example $\tau \,.\, P + \tau \,.\, Q$. This represents a kind of internal choice: one of the branches will be taken but the environment cannot affect which one. The agent behaves in the same way as

$$(\boldsymbol{\nu} a)(\overline{a} \mid a \,.\, P \mid a \,.\, Q)$$

assuming $a \notin \mathtt{fn}(P, Q)$. For another example consider the agent $Case_a(Q, R)$ from the previous section; it was defined as

$$Case_a(Q, R) \quad = \quad (\boldsymbol{\nu} xy)\overline{a}\langle xy\rangle \,.\, (x \,.\, Q \ + \ y \,.\, R)$$

Here the Sum can simply be replaced by a Parallel composition

$$Case_a(Q, R) \quad = \quad (\boldsymbol{\nu} xy)\overline{a}\langle xy\rangle \,.\, (x \,.\, Q \ \mid \ y \,.\, R)$$

without affecting the behaviour, assuming $True_a$ and $False_a$ are defined as before. It will now hold that $Case_a(Q, R) \mid True_a$ evolves to

$$(\boldsymbol{\nu} xy)(\overline{x} \ \mid \ x \,.\, Q \ \mid \ y \,.\, R) \xrightarrow{\tau} Q \ \mid \ (\boldsymbol{\nu} y)(y \,.\, R)$$

and this agent will behave as Q, since R is guarded by a private name.

3.3 The Polyadic Calculus

A straightforward extension is to allow multiple objects in communications: outputs of type $\overline{a}\langle y_1 \cdots y_n \rangle \,.\, P$ and inputs of type $a(x_1 \cdots x_n) \,.\, Q$ where the x_i are pairwise distinct. We here also admit the case $n = 0$ when there is no object at all, and then we elide the brackets writing $\overline{a}$ for $\overline{a}\langle\rangle$ and a for $a()$. As was seen in Section 2.3 this does not really increase the expressiveness of the calculus since polyadic interactions can be emulated by sequences of monadic interactions over a private link.

The semantics for a polyadic calculus is only notationally more complex than for the monadic calculus. As expected we will be able to infer

$$a(\tilde{x}) \,.\, P \mid \overline{a}\langle\tilde{y}\rangle \,.\, Q \xrightarrow{\tau} P\{\tilde{y}/\tilde{x}\} \mid Q$$

where the substitution is a simultaneous substitution of all y_i for x_i (note that all x_i must be different since they are objects in the same input Prefix).

The question then arises how to treat agents such as $a(xy) \,.\, P \mid \overline{a}\langle u \rangle \,.\, Q$ where the arity of the output is not the same as the arity of the input. Such an incompatibility should be caught by a type system. The idea is that each name is assigned a *sort*, containing information about the objects that can be passed along that name. If $\mathcal{S}$ is a set of sorts, a *sort context* Δ is a function from $\mathcal{N}$ to $\mathcal{S}$, in other words $\Delta(a)$ is the sort of a.

In the simplest system a sort would just be a natural number, $\mathcal{S} = \mathbf{N}$, such that $\Delta(a)$ denotes the arity of a, i.e., the number of objects in any Prefix where a is the subject. Formally we write $\Delta \vdash P$ to mean that P conforms to Δ, and rules for inferring $\Delta \vdash P$ can easily be given by induction over the structure of P. For example,

$$\frac{\Delta \vdash P,\ \Delta(a) = n}{\Delta \vdash \overline{a}\langle x_1 \cdots x_n \rangle \,.\, P} \qquad\qquad \frac{\Delta \vdash P,\ \Delta \vdash Q}{\Delta \vdash P|Q}$$

With this idea the agent $a(xy) \,.\, P \mid \overline{a}\langle u \rangle \,.\, Q$ is ill-formed in the sense that it cannot conform to any sort context: $\Delta(a)$ is required to be 2 in one component and 1 in the other component. However, although this simple scheme works for this particular example it is not in general sufficient to catch all incompatible arities that may arise when an agent executes. Consider:

$$a(u) \,.\, u(z) \mid \overline{a}\langle x \rangle \,.\, \overline{x}\langle vv \rangle$$

A sorting assigning 2 to x and 1 to all other names works fine here. Yet the agent can evolve through an interaction along a to

$$x(z) \mid \overline{x}\langle vv \rangle$$

which is ill-formed. To be able to catch not only the immediately obvious arity conflicts but also any such conflicts that can arise during execution more information must be added to the sorts. For each name, the number of objects passed along that name is not enough, also the sort of each such object must be included. In the example above, the left component requires the sort of a to be one object (corresponding to u) which has sort 1 because of the subterm $u(z)$. The right component requires the sort of a to be one object (corresponding to x) of sort 2 because of the subterm $\overline{x}\langle vv\rangle$. With this refined notion of sort an agent such as $a(u) . u(z) \mid \overline{a}x . \overline{x}\langle vv\rangle$ is ill-formed.

Of course arity conflicts can lie arbitrarily deep, meaning that the sort of a must contain information about the objects which are passed along the objects which are passed along . . . which are passed along a to arbitrary depth. One way to set this up is the following. To each sort S in $\mathcal{S}$ associate a fixed *object sort* $ob(S)$ in $\mathcal{S}^*$ i.e., the object sort is a (possibly empty) sequence of sorts. The intention is that if a has sort S where $ob(S) = \langle S_1 \dots S_n\rangle$, and $a(x_1 \dots x_n)$ is a Prefix, then each x_i has sort S_i. With a slight abuse of notation the sorting rule for input then becomes:

$$\frac{\Delta \cup \{\tilde{x} \mapsto ob(\Delta(a))\} \vdash P}{\Delta \vdash a(\tilde{x}) . P}$$

It should be read as follows: In order to establish that $a(x_1 \dots x_n) . P$ conforms to Δ, find the object sorts $S_1 \dots S_n$ of a according to Δ, and verify that P conforms to Δ where also each x_i is assigned sort S_i. The rule for output is:

$$\frac{ob(\Delta(a)) = \Delta(y_1) \dots \Delta(y_n), \quad \Delta \vdash P}{\Delta \vdash \overline{a}\tilde{y} . P}$$

In order to establish that $\overline{a}\langle y_1 \dots y_n\rangle . P$ conforms to Δ it is enough to show that it assigns $y_1 \dots y_n$ the object sorts of a, and that P conforms to Δ. In this way agents such as $\overline{a}\,a . P$, where a name is sent along itself, can also be given a sorting: if S is the sort of a then $ob(S) = \langle S\rangle$.

There is a large variety of type systems which include more information, such as if names can be used for input or output or both, and there are notions of subtypes and polymorphism. Binding occurrences in Restrictions are either explicitly sorted as in $(\boldsymbol{\nu} x : S)P$, or a sort inference system is used to compute a suitable sort. Whether the latter is possible depends on the details of the sort system.

In conclusion, although polyadic interactions do not really increase the expressiveness it adds convenience and clarity when using the calculus, and efficiently implementable sort systems can ascertain that no mismatching arities will ever occur when agents evolve.

3.4 Recursion and Replication

In the π-calculus as in most other process algebras the mechanism for describing iterative or arbitrarily long behaviour is recursion: a recursive Definition

$$A(\tilde{x}) \stackrel{\mathrm{def}}{=} P$$

where A occurs in P can be thought of as the definition of a recursive procedure A with formal parameters $\tilde{x}$, and the agent $A(\tilde{y})$ is then an invocation with actual parameters $\tilde{y}$. Sometimes this is notated through an explicit fixpoint constructor: if P is an agent then fix $X \,.\, P$ is an agent. Here the *agent variable* X may occur in P, and fix $X \,.\, P$ means the same as the agent Identifier A with the Definition $A \stackrel{\mathrm{def}}{=} P\{A/X\}$. Fixpoints and Definitions are thus only notational variants. In large specifications the Definitions tend to be more readable, but the fixpoints sometimes allow a more optimal formulation for theory development.

For some purposes the special case of *Replication* is convenient. If P is an agent then $!P$, the Replication of P, is given by the definition

$$!P \stackrel{\mathrm{def}}{=} P \mid !P$$

or using fixpoints: $!P$ is the agent

$$\mathrm{fix}\, X \,.\, (P \mid X)$$

In other words, $!P$ represents an unbounded number of copies of P — the recursion can obviously be unfolded an arbitrary number of times:

$$!P \;\equiv\; P \mid !P \;\equiv\; P \mid P \mid !P \;\equiv\; P \mid P \mid P \mid !P \quad \text{etc.}$$

For example, an agent which can receive inputs along i and forward them on o is:

$$M \;=\; !i(x) \,.\, \overline{o}x$$

Suppose this agent receives first u and then v along i. Then, by unfolding the Replication, $M \equiv i(x) \,.\, \overline{o}x \mid i(x) \,.\, \overline{o}x \mid M$, the agent will evolve to $\overline{o}u \mid \overline{o}v \mid M$, ready to receive more messages along i but also to emit u and v in arbitrary order.

Any agent using a finite family of Definitions can be encoded by Replication as follows. Consider

$$\begin{array}{rcl} A_1(\tilde{x}_1) & \stackrel{\mathrm{def}}{=} & P_1 \\ & \vdots & \\ A_n(\tilde{x}_n) & \stackrel{\mathrm{def}}{=} & P_n \end{array}$$

used in an agent Q. Then there is a corresponding agent Q' which behaves as Q and only uses Replication. Q' is constructed as follows. Introduce names

$a_1, \ldots, a_n$, one for each Identifier, and let $\widehat{P}$ be the agent obtained by replacing any invocation $A_i(\tilde{y})$ with an output $\overline{a}_i\langle\tilde{y}\rangle$ in P. Let

$$D \quad = \quad !a_1(\tilde{x}_1) \,.\, \widehat{P_1} \mid \cdots \mid !a_n(\tilde{x}_n) \,.\, \widehat{P_n}$$

Thus D is an agent which emulates all the Definitions in the sense that it can interact with any such output. Finally

$$Q' \quad = \quad (\boldsymbol{\nu} a_1 \cdots a_n)(D | \widehat{Q})$$

will then behave in the same way as Q, with the only difference that an unfolding of a Definition will be emulated by an interaction between $\widehat{Q}$ and D. For example, let $N \overset{\text{def}}{=} i(x) \,.\, \overline{o}x \,.\, N$, which differs from M above in that messages are delivered in order and only one message is stored at a time. The agent with only Replication which behaves as N is

$$(\boldsymbol{\nu} a)(!a \,.\, i(x) \,.\, \overline{o}x \,.\, \overline{a} \quad | \quad \overline{a})$$

3.5 The Asynchronous Calculus

The π-calculus, as most other process algebras, is based on the paradigm of synchronous communication; an interaction means that one component emits a name at the same time as another component receives it. In contrast, in asynchronous communication there is an unpredictable delay between output and input, during which the message is in transit. This can be modelled by inserting an agent representing an asynchronous communication medium between sender and receiver. The properties of the medium (whether it has a bound on the capacity, whether it preserves the order of messages etc.) is then determined by its definition. For example, an unbounded medium not preserving the order of messages is:

$$M \quad \overset{\text{def}}{=} \quad i(x) \,.\, (\overline{o}x \mid M)$$

(The same behaviour is expressed with Replication in Section 3.4.) When M receives u along i it evolves to $\overline{o}u \mid M$, and can at any time deliver u along o, and also continue to accept more messages.

Interestingly, this particular form of asynchrony is also captured by a subcalculus of π in which there is no need to explicitly represent media. The subcalculus consists of the agents satisfying the following requirements:

1. Only $\mathbf{0}$ can follow an output Prefix.

2. An output Prefix may not occur as an unguarded operand of $+$.

The first requirement disallows agents such as $\overline{a}x \,.\, \overline{b}y$, where an agent other than $\mathbf{0}$ follows $\overline{a}x$. The second requirement disallows $\overline{a}x + b(y)$, but allows $\tau \,.\, \overline{a}x + b(y)$.

This subcalculus is known as the *asynchronous* π-calculus, and the rationale behind it is as follows. An unguarded output Prefix $\overline{a}x$ occurring in a term represents a message that has been sent but not yet received. The action of sending the message is placing it in an unguarded position, as in the following

$$\tau\,.\,(\overline{a}x \mid P) \xrightarrow{\tau} \overline{a}x \mid P$$

After this transition, $\overline{a}x$ can interact with a receiver, and the sender proceeds concurrently as P. Because of requirement 1 the fact that a message has been received cannot be detected by P unless the receiver explicitly sends an acknowledgement. Because of requirement 2 a message cannot disappear unless it is received. Therefore, $\tau\,.\,(\overline{a}x \mid P)$ can be paraphrased as "send $\overline{a}x$ asynchronously and continue as P."

Of course, with a scheme for sending explicit acknowledgements synchronous communication can be emulated, so the loss of expressiveness from the full π-calculus is largely pragmatic. In agents like

$$\tau\,.\,(\boldsymbol{\nu} b)(\overline{a}b \mid b(x)\,.\,P)$$

the scope of b is used to protect this kind of acknowledgement. Here the agent can do an asynchronous output of $\overline{a}b$, but it is blocked from continuing until it receives an acknowledgement along b:

$$(\boldsymbol{\nu} b)(\overline{a}b \mid b(x)\,.\,P) \mid a(x)\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu} b)(b(x)\,.\,P \mid Q\{b/x\})$$

The acknowledgement can only arrive from the recipient (Q) of the message, since there is no other agent that can send along the restricted name b.

The asynchronous calculus has been used successfully to model programming languages where the underlying communication discipline is asynchronous. The theory is slightly different since only "asynchronous" observers are deemed relevant. For example, the agent $a(x)\,.\,\overline{a}x$ is semantically the same as τ since the actions of receiving a datum and then emitting it on the same channel cannot be detected by any other asynchronous process or observer. An interesting consequence is that a general form of Sum, the input-guarded Sum $y_1(x)\,.\,P_1 + \cdots + y_n(x)\,.\,P_n$ can be encoded as:

$$\begin{array}{lll}(\boldsymbol{\nu} a)(\overline{a}t \mid (& y_1(x)\,.\,a(z)\,.\,(\overline{a}f \mid \texttt{if } z = t \texttt{ then } P_1 \texttt{ else } \overline{y}_1 x) \\ & \mid \\ & \vdots \\ & \mid \\ & y_n(x)\,.\,a(z)\,.\,(\overline{a}f \mid \texttt{if } z = t \texttt{ then } P_n \texttt{ else } \overline{y}_n x)))\end{array}$$

The idea is that a acts as a lock. Initially t is emitted on a; the first component to interact with a will continue to emit f on a, all other components will resend the datum along y_i, thereby undoing the input along y_i. This only works in the asynchronous calculus, where receiving and then emitting the same message is, from an observer's point of view, the same as doing nothing. (An alert reader will complain that `if...then...else` is defined using Sum. This is no great matter, it can be taken as primitive or defined as the Parallel composition of the branches.)

3.6 The Higher-Order Calculus

Finally we shall look at an extension of the π-calculus called the *higher-order* π-calculus. Here the objects transmitted in interactions can also be agents. A higher-order output Prefix form is of the kind

$$\overline{a}\langle P\rangle \,.\, Q$$

meaning "send the agent P along a and then continue as Q." The higher-order input Prefix form is of the kind

$$a(X) \,.\, Q$$

meaning "receive an agent for X and continue as Q." Of course Q may here contain X, and the received agent will then replace X in Q. For example, an interaction is

$$\overline{a}\langle \overline{b}u \,.\, \mathbf{0}\rangle \,.\, b(x) \;\mid\; a(X) \,.\, (X|\overline{c}v) \quad \xrightarrow{\tau} \quad b(x) \mid \overline{b}u \mid \overline{c}v$$

The first component sends $\overline{b}u \,.\, \mathbf{0}$ along a, the second component receives it and executes it in parallel with $\overline{c}v$. The transmitted agent $\overline{b}u$ is then free to interact with the remainder of the first component $b(x)$, sending u to it along b. Since names and agents can both be objects in an interaction, a sorting discipline such as in Section 3.3, extended with a sort for "agent", can be employed to weed out unwanted combinations such as $\overline{a}\langle P\rangle \,.\, Q \mid a(x) \,.\, x(u)$.

Formally, we introduce a new category of *agent variables*, ranged over by X, and extend the definition of agents to include the agent variables and the higher-order Prefix forms. The notion of replacing an agent variable by an agent, $P\{Q/X\}$, is as expected (and the same as in the fixpoint construct of Section 3.4), and the higher-order interaction rule gives that

$$a(X) \,.\, P \mid \overline{a}\langle Q\rangle \,.\, R \quad \xrightarrow{\tau} \quad P\{Q/X\} \mid R$$

The substitution $P\{Q/X\}$ is defined with alpha-conversion such that free names in Q do not become bound in $P\{Q/X\}$. To see the need for this consider

$$a(X) \,.\, b(c) \,.\, X \mid \overline{a}\langle c(z) \,.\, \mathbf{0}\rangle \,.\, R$$

Clearly the name c in the right hand component is not the same as the bound c to the left. The agent is alpha-equivalent to

$$a(X)\,.\,b(d)\,.\,X \mid \overline{a}\langle c(z)\,.\,\mathbf{0}\rangle\,.\,R$$

where these names are dissociated. Therefore a transition to $b(c)\,.\,c(z) \mid R$ is not possible. In this sense the binding corresponds to *static* binding: the scope of a name is determined by where it occurs in the agent. An alternative scheme of *dynamic* binding, where the scope is determined only when the name is actually used, is possible but much more complicated semantically. For example, such bindings cannot use alpha-conversion.

Note that an agent variable may occur more than once in an agent. For example,

$$a(X)\,.\,(X|X) \mid \overline{a}\langle Q\rangle\,.\,R \quad \xrightarrow{\tau} \quad Q \mid Q \mid R$$

where the transmitted agent Q has been "duplicated" by the interaction. This lends the higher-order calculus considerable expressive power. Recursion and Replication are now derivable constructs (as in the lambda-calculus where the fixpoint combinator $\mathbf{Y}$ is expressible). Consider

$$D \;=\; a(X)\,.\,(X \mid \overline{a}\langle X\rangle)$$

D accepts an agent along a, and will start that agent and also retransmit it along a. Now let P be any agent with $a \notin \mathtt{fn}(P)$, and

$$R_P \;=\; (\boldsymbol{\nu} a)(D \mid \overline{a}\langle P|D\rangle)$$

Then R_P behaves as !P since it will spawn an arbitrary number of P:

$$R_P \xrightarrow{\tau} (\boldsymbol{\nu} a)((P|D) \mid \overline{a}\langle P|D\rangle) \;\equiv\; P \mid R_P \xrightarrow{\tau} P \mid P \mid R_P \xrightarrow{\tau} \cdots$$

Obviously, for modelling programming languages with higher-order constructs (such as functions with functions as parameters, or processes that can migrate between sites) the higher-order calculus is suitable. However, its theory is considerably more complicated. In some situations it will then help to encode it into the usual calculus with the device from Section 2.3. Instead of transmitting an agent P we transmit a new name which can be used to trigger P. Since the receiver of P might invoke P several times (because the corresponding agent variable occurs at several places) P must be replicated. The main idea of the encoding $[\![\bullet]\!]$ from higher-order to the ordinary calculus is as follows, where we assume a previously unused name x for each agent variable X:

$$\begin{array}{lcll}
[\![(\overline{a}\langle P\rangle\,.\,Q)]\!] & = & (\boldsymbol{\nu} p)\overline{a}p\,.\,([\![Q]\!] \mid\ !p\,.\,[\![P]\!]) & \text{where } p \notin \mathtt{fn}(P,Q) \\
[\![a(X)\,.\,P]\!] & = & a(x)\,.\,[\![P]\!] & \\
[\![X]\!] & = & \overline{x} &
\end{array}$$

Consider the example above, $a(X) . (X|X) \mid \overline{a}\langle Q\rangle . R \quad \xrightarrow{\tau} \quad Q \mid Q \mid R$, where Q and R in turn contain no higher-order Prefixes. Using the encoding, and assuming q is not free in Q or R, we get a similar behaviour:

$$\begin{array}{rl} & a(x) . (\overline{x} \mid \overline{x}) \mid (\boldsymbol{\nu} q)\overline{a}q . (R \mid !q . Q) \\ \xrightarrow{\tau} & (\boldsymbol{\nu} q)(\overline{q} \mid \overline{q} \mid R \mid !q . Q) \\ \xrightarrow{\tau}\xrightarrow{\tau} & (\boldsymbol{\nu} q)(\mathbf{0} \mid \mathbf{0} \mid R \mid Q \mid Q \mid !q . Q) \\ \equiv & R|Q|Q \mid (\boldsymbol{\nu} q)!q . Q \end{array}$$

where the rightmost component $(\boldsymbol{\nu} q)!q . Q$ will never be able to execute because it is guarded by a private name, so the whole term will behave as $Q|Q|R$ as expected.

4 Operational Semantics

The standard way to give an operational semantics to a process algebra is through a labelled transition system, where transitions are of kind $P \xrightarrow{\alpha} Q$ for some set of actions ranged over by α. The π-calculus follows this norm and most of the rules of transitions are similar to other algebras. As expected, for an agent $\alpha . P$ there will be a transition labelled α leading to P. Also as expected, the Restriction operator will not permit an action with the restricted name as subject, so $(\boldsymbol{\nu} a)\overline{a}u . P$ has no transitions and is therefore semantically equivalent to $\mathbf{0}$. But what action should

$$(\boldsymbol{\nu} u)\overline{a}u . P$$

have? Clearly it must have *some* action; inserted in a context $a(x) . Q \mid (\boldsymbol{\nu} u)\overline{a}u . P$ it will enable an interaction since, assuming $u \notin \texttt{fn}(Q)$, this term is structurally congruent to $(\boldsymbol{\nu} u)(a(x) . Q \mid \overline{a}u . P)$ and there is an interaction between the components. So $(\boldsymbol{\nu} u)\overline{a}u . P$ is not, intuitively, something that behaves as $\mathbf{0}$. On the other hand it is clearly distinct from $\overline{a}u . P$ and it can therefore not be given the action $\overline{a}u$. For example,

$$(a(x) . \texttt{if } x = u \texttt{ then } Q) \mid \overline{a}u \xrightarrow{\tau} \texttt{if } u = u \texttt{ then } Q$$

which can continue as Q, while

$$\begin{array}{l} (a(x) . \texttt{if } x = u \texttt{ then } Q) \mid (\boldsymbol{\nu} u)\overline{a}u \ \equiv \\ (\boldsymbol{\nu} v)((a(x) . \texttt{if } x = u \texttt{ then } Q) \mid \overline{a}v) \ \xrightarrow{\tau} \\ (\boldsymbol{\nu} v)\texttt{if } v = u \texttt{ then } Q \end{array}$$

and there are no further actions (remember that scope extension requires that u is alpha-converted before the scope is extended.)

The solution is to give $(\boldsymbol{\nu} u)\overline{a}u$ a new kind of action called *bound output* written $\overline{a}\boldsymbol{\nu} u$. The intuition is that a local name represented by u is transmitted along a, extending the scope of u to the recipient. In summary, the *actions* ranged over by α consists of four classes:

1. The internal action τ.
2. The (free) output actions of kind $\overline{a}x$.
3. The input actions of kind $a(x)$.
4. The bound output actions $\overline{a}\boldsymbol{\nu}x$.

The three first kinds correspond precisely to the Prefixes in the calculus. For the sake of symmetry we introduce a fourth kind of Prefix $\overline{a}\boldsymbol{\nu}x$, for $a \neq x$, corresponding to the bound output action. The bound output Prefix is merely a combination of Restriction and output as defined by $\overline{a}\boldsymbol{\nu}x\,.\,P = (\boldsymbol{\nu}x)\overline{a}x\,.\,P$. In this way we can continue to let α range over both actions and Prefixes, where $\mathtt{fn}(\overline{a}\boldsymbol{\nu}x) = \{a\}$ and $\mathtt{bn}(\overline{a}\boldsymbol{\nu}x) = \{x\}$.

An input transition $P \xrightarrow{a(x)} Q$ means that P can receive some name u along a, and then evolve to $Q\{u/x\}$. In that action x does not represent the value received, rather it is a reference to the places in Q where the received name will appear. When examining further transitions from Q, all possible instantiations for this x must be considered. Those familiar with functional programming and the lambda-calculus might think of the transition as $P \xrightarrow{a} \lambda x\, Q$, making it clear that the derivative, after the arrow, has a functional parameter x. (The reader may at this point wonder about an alternative way to treat input by including the received name in the action, as in $a(x)\,.\,P \xrightarrow{au} P\{u/x\}$. This is a viable alternative and will be discussed in Section 5.3.)

Similarly, the bound output transition $P \xrightarrow{\overline{a}\boldsymbol{\nu}x} Q$ signifies an output of a local name and x indicates where in Q this name occurs. Here x is not a functional parameter, it just represents something that is distinct from all names in the environment.

The labelled transition semantics is given in Table 3. The rule STRUCT makes explicit our intuition that structurally congruent agents count as the same for the purposes of the semantics. This simplifies the system of transition rules. For example, SUM is sufficient as it stands in Table 3. The dual rule

$$\textsc{sum}_2 \quad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'}$$

is redundant since it can be inferred from SUM and STRUCT. Similar arguments hold for the rules PAR and COM.

In the rule PAR, note the extra condition that Q does not contain a name bound in α. This conforms to the intuition that bound names are just references to occurrences; in the conclusion $P|Q \xrightarrow{\alpha} P'|Q$ the action should not refer to any occurrence in Q. To see the need for the condition consider the inference

$$\textsc{par} \quad \frac{a(x)\,.\,P \xrightarrow{a(x)} P}{(a(x)\,.\,P) \mid Q \xrightarrow{a(x)} P \mid Q}$$

$$\text{STRUCT}\quad \frac{P' \equiv P,\ P \xrightarrow{\alpha} Q,\ Q \equiv Q'}{P' \xrightarrow{\alpha} Q'}$$

$$\text{PREFIX}\quad \frac{}{\alpha\,.\,P \ \xrightarrow{\alpha}\ P}$$

$$\text{SUM}\quad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'}$$

$$\text{MATCH}\quad \frac{P \xrightarrow{\alpha} P'}{\texttt{if } x = x \texttt{ then } P \xrightarrow{\alpha} P'}$$

$$\text{MISMATCH}\quad \frac{P \xrightarrow{\alpha} P',\ x \neq y}{\texttt{if } x \neq y \texttt{ then } P \xrightarrow{\alpha} P'}$$

$$\text{PAR}\quad \frac{P \xrightarrow{\alpha} P',\ \texttt{bn}(\alpha) \cap \texttt{fn}(Q) = \emptyset}{P|Q \xrightarrow{\alpha} P'|Q}$$

$$\text{COM}\quad \frac{P \xrightarrow{a(x)} P',\ Q \xrightarrow{\overline{a}u} Q'}{P|Q \xrightarrow{\tau} P'\{u/x\}|Q'}$$

$$\text{RES}\quad \frac{P \xrightarrow{\alpha} P',\ x \notin \alpha}{(\boldsymbol{\nu} x)P \xrightarrow{\alpha} (\boldsymbol{\nu} x)P'}$$

$$\text{OPEN}\quad \frac{P \xrightarrow{\overline{a}x} P',\ a \neq x}{(\boldsymbol{\nu} x)P \xrightarrow{\overline{a}\boldsymbol{\nu} x} P'}$$

Table 3: The operational semantics.

Combined with an output $\overline{a}u\,.\,R \xrightarrow{\overline{a}u} R$ we get

$$\textsc{com}\ \frac{(a(x)\,.\,P) \mid Q \xrightarrow{a(x)} P \mid Q, \quad \overline{a}u\,.\,R \xrightarrow{\overline{a}u} R}{((a(x)\,.\,P) \mid Q) \mid \overline{a}u\,.\,R \xrightarrow{\tau} (P \mid Q)\{u/x\} \mid R}$$

This is clearly only correct if $x \notin \mathtt{fn}(Q)$, otherwise the free x in Q would be affected by the substitution. If $x \in \mathtt{fn}(Q)$ then a similar derivation is possible after first alpha-converting x in $a(x)\,.\,P$ to some name not free in Q.

The rule OPEN is the rule generating bound outputs. It is interesting to note that bound output actions do not appear in the COM rule and therefore cannot interact directly with inputs. Such interactions are instead inferred by using structural congruence to pull the Restriction outside both interacting agents (possibly after an alpha-conversion), as in the following where we assume that $u \notin \mathtt{fn}(P)$:

$$\textsc{struct}\ \frac{\textsc{res}\ \dfrac{a(x)\,.\,P \mid \overline{a}u\,.\,Q \xrightarrow{\tau} P\{u/x\} \mid Q}{(\boldsymbol{\nu} u)(a(x)\,.\,P \mid \overline{a}u\,.\,Q) \xrightarrow{\tau} (\boldsymbol{\nu} u)(P\{u/x\} \mid Q)}}{a(x)\,.\,P \mid (\boldsymbol{\nu} u)\overline{a}u\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu} u)(P\{u/x\} \mid Q)}$$

In view of this it might be argued that bound output transitions and the rule OPEN can be omitted altogether, since they have no impact on inferring interactions. Although technically this argument is valid there are other reasons for including the bound output. One is of philosophical nature: we think of the agent $(\boldsymbol{\nu} u)\overline{a}u$ as, intuitively, being able to do something, namely exporting a local name, and if we do not dignify that with a transition we introduce an incompleteness in the semantics in that not all behaviour is manifested by transitions. Another reason is of technical convenience: when it comes to developing behavioural equivalences (see Section 6) the bound output transitions will turn out to be indispensable.

5 Variants of the Semantics

We will here consider alternative operational semantics. Some are mere presentational variants. But different semantics have different strengths, for example one may be suitable for an automatic tool and another may boost intuition, so there is a point to the diversity.

5.1 The Role of Structural Congruence

Through the rule STRUCT we can regard an inference of $P \equiv Q$ as a step in an inference of a transition. If the *only* purpose of the congruence is to facilitate inference of labelled transitions then there is a possible trade-off between what to include in the structural congruence and what to include in the transition rules. As we saw the dual $\textsc{sum}_2$ of SUM is unnecessary, but an alternative solution is to omit commutativity from the structural congruence and introduce SUM$_2$. A similar choice occurs for the Parallel combinator: structural commutativity can be omitted at the price of introducing the duals of PAR and COM.

For some operators there is even a choice between defining it entirely through the congruence or through the transition rules. For example, the Match operator can either be defined with the rule MATCH or with an additional structural rule

$$\texttt{if}\ x = x\ \texttt{then}\ P \quad \equiv \quad P$$

In Table 3 we chose to give it with a transition rule because of the symmetry with MISMATCH. Note that Mismatch cannot be defined through the perhaps expected structural rule

$$\texttt{if}\ x \neq y\ \texttt{then}\ P \quad \equiv \quad P \quad \text{if}\ x \neq y$$

since that rule would be obviously unsound for a Mismatch under an input Prefix:

$$a(x)\,.\,\texttt{if}\ x \neq y\ \texttt{then}\ P \quad \text{and} \quad a(x)\,.\,P$$

represent two different behaviours!

Another such choice occurs for Identifiers. They are here defined through a structural congruence rule $A(\tilde{y}) \equiv P\{\tilde{y}/\tilde{x}\}$ if $A(\tilde{x}) \overset{\text{def}}{=} P$. Alternatively we could have given the transition rule:

$$\text{IDE}\ \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\alpha} P',\ \ A(\tilde{x}) \overset{\text{def}}{=} P}{A(\tilde{y}) \xrightarrow{\alpha} P'}$$

Perhaps the most dramatic part of the structural congruence is the scope extension which, together with COM, make possible transitions like

$$a(x)\,.\,P \mid (\boldsymbol{\nu} u)\overline{a}u\,.\,Q \xrightarrow{\tau} (\boldsymbol{\nu} u)(P\{u/x\} \mid Q)$$

(provided $u \notin \texttt{fn}(P)$). Interestingly, there is a way to achieve the same effect without the scope extension. It involves a new transition rule CLOSE which represents an interaction between a bound output and an input:

$$\text{CLOSE}\ \frac{P \xrightarrow{a(u)} P',\ \ Q \xrightarrow{\overline{a}\boldsymbol{\nu} u} Q'}{P|Q \xrightarrow{\tau} (\boldsymbol{\nu} u)(P'|Q')}$$

Let us see how the transition from $a(x)\,.\,P \mid (\boldsymbol{\nu} u)\overline{a}u\,.\,Q$ mentioned above is derived. It involves an alpha-conversion of x to u in the first operand:

$$\text{STRUCT}\ \frac{a(x)\,.\,P \equiv a(u)\,.\,P\{u/x\},\ \ a(u)\,.\,P\{u/x\} \xrightarrow{a(u)} P\{u/x\}}{a(x)\,.\,P \xrightarrow{a(u)} P\{u/x\}}$$

OPEN in the second component gives:

$$\text{OPEN}\ \frac{\overline{a}u \,.\, Q \xrightarrow{\overline{a}u} Q}{(\boldsymbol{\nu} u)\overline{a}u \,.\, Q \xrightarrow{\overline{a}\boldsymbol{\nu} u} Q}$$

and combining the two conclusions we get:

$$\text{CLOSE}\ \frac{a(x) \,.\, P \xrightarrow{a(u)} P\{u/x\}, \quad (\boldsymbol{\nu} u)\overline{a}u \,.\, Q \xrightarrow{\overline{a}\boldsymbol{\nu} u} Q}{a(x) \,.\, P \mid (\boldsymbol{\nu} u)\overline{a}u \,.\, Q \xrightarrow{\tau} (\boldsymbol{\nu} u)(P\{u/x\} \mid Q)}$$

Note how the scoping represented by $\boldsymbol{\nu}$ moves from the second operand onto the transition arrow and reappears in the term in the conclusion!

In this way it is possible to remove all of the structural congruence except alpha-conversion, at the expense of introducing more transition rules. In fact, it is even possible to remove the alpha-conversion by building it into a special rule, or into the rules that generate bound actions. For example, the rule for input would become

$$\frac{}{a(x) \,.\, P \xrightarrow{a(u)} P\{u/x\}}$$

for any $u \notin \texttt{fn}(P)$, and similarly the rule OPEN would need a modification.

Therefore, the choice whether to use a structural congruence at all and, if so, how many rules it should contain is largely one of convenience. Historically, the first presentations of the calculus did not use structural congruence and an advantage is then that some proofs by induction over inference of transition become clearer. Today most presentations use some form of structural congruence, at least including alpha-conversion, since the definitions become more compact and transparent. For example, the scope extension in structural congruence is easier to understand than the effect of the CLOSE and OPEN rules.

5.2 Symbolic Transitions

In the transitional semantics an input action $P \xrightarrow{a(x)} Q$ represents that anything can be received along a. Consequently, if we want to explore further transitions from Q we must take into account all possible substitutions for x. Since there are infinitely many names it appears we have to explore the behaviour of $Q\{u/x\}$ for infinitely many names u. In general, when a number of such inputs have been performed, we may be interested in the behaviour not merely of an agent P but of $P\sigma$ for substitutions σ involving several names. This kind of infinite branching is awkward for proof systems and prohibits efficient tool support for automated analysis.

Therefore, an alternative way to present the semantics is to let a transition from P contain information about the behaviour not only of P itself but also of $P\sigma$ for any σ. The key observation is that even though there are infinitely many names a syntactically finite agent can only subject them to a bounded number of tests, and the conditions which enable a transition can be recorded. For example, consider

$$P \;=\; \overline{a}u\,.\,Q \mid b(x)\,.\,R$$

We can schematically express the transitions of $P\sigma$ succinctly as follows. For any σ the agent $P\sigma$ will have an input and an output transition (with subject $\sigma(a)$ and $\sigma(b)$ respectively). In addition, if $\sigma(a) = \sigma(b)$ it will have a τ-transition arising from a communication between the components. This is the idea behind the so called *symbolic* semantics, where these transitions come out as:

$$\begin{array}{lcl} P & \xrightarrow{\textbf{true},\, \overline{a}u} & Q \mid b(x)\,.\,R \\ P & \xrightarrow{\textbf{true},\, b(x)} & \overline{a}u\,.\,Q \mid R \\ P & \xrightarrow{a=b,\, \tau} & Q \mid R\{u/x\} \end{array}$$

The label on a transition here has two components. The first is a condition which must hold for the transition to be enabled, and the second is as usual the action of the transition.

In the π-calculus the conditions have a particularly simple form. The only tests an agent can perform on names are for equality (arising from the Parallel combinator, as above, and from Match) and inequality (arising from Mismatch). Of course several of these tests may affect a transition, so a condition will in general be a conjunction of equalities and inequalities on names. For example, we will have that

$$\texttt{if}\ x = y\ \texttt{then}\ \overline{a}u\,.\,Q \mid \texttt{if}\ x \neq b\ \texttt{then}\ b(y)\,.\,R \xrightarrow{x=y \wedge x \neq b \wedge a=b,\, \tau} Q \mid R\{u/y\}$$

Formally we let M, N range over conjunctions of name equalities and inequalities, including the empty conjunction which is written **true**. Symbolic transitions are of the form $P \xrightarrow{M,\,\alpha} Q$, meaning "if M holds then P has an action α leading to Q." We let μ range over pairs (M, α), so a symbolic transition is written $P \xrightarrow{\mu} Q$. For $\mu = (M, \alpha)$ we let $\texttt{bn}(\mu)$ mean $\texttt{bn}(\alpha)$ and we let $\mu \wedge N$ mean $(M \wedge N,\ \alpha)$. The rules for symbolic transitions are given in Table 4 and are similar in structure to the rules in Section 4 for ordinary transitions. The only subtle point is the effect of Restriction on a condition in the rules S-RES and S-OPEN. Suppose P has a symbolic transition

$$P \xrightarrow{M,\,\alpha} Q$$

with a condition M that mentions x in a conjunct $x \neq y$. This means that the transition from P only holds for substitutions assigning x and y different names, perhaps because P is of the form $\texttt{if}\ x \neq y\ \texttt{then}\ \ldots$ What is then the condition M' of the inferred symbolic transition

$$(\boldsymbol{\nu} x)P \xrightarrow{M',\,\alpha'} Q'$$

$$\text{S}-\text{STRUCT}\quad \frac{P' \equiv P,\ P \xrightarrow{\mu} Q,\ Q \equiv Q'}{P' \xrightarrow{\mu} Q'}$$

$$\text{S}-\text{PREFIX}\quad \frac{}{\alpha\,.\,P \xrightarrow{\mathbf{true},\,\alpha} P}$$

$$\text{S}-\text{SUM}\quad \frac{P \xrightarrow{\mu} P'}{P+Q \xrightarrow{\mu} P'}$$

$$\text{S}-\text{MATCH}\quad \frac{P \xrightarrow{\mu} P' \quad x,y \notin \mathtt{bn}(\mu)}{\mathtt{if}\ x=y\ \mathtt{then}\ P \xrightarrow{\mu\wedge x=y} P'}$$

$$\text{S}-\text{MISMATCH}\quad \frac{P \xrightarrow{\mu} P',\quad x,y \notin \mathtt{bn}(\mu)}{\mathtt{if}\ x\neq y\ \mathtt{then}\ P \xrightarrow{\mu\wedge x\neq y} P'}$$

$$\text{S}-\text{PAR}\quad \frac{P \xrightarrow{\mu} P',\ \mathtt{bn}(\mu)\cap\mathtt{fn}(Q)=\emptyset}{P|Q \xrightarrow{\mu} P'|Q}$$

$$\text{S}-\text{COM}\quad \frac{P \xrightarrow{M,\,a(x)} P',\ Q \xrightarrow{N,\,\overline{b}u} Q'}{P|Q \xrightarrow{M\wedge N\wedge a=b,\,\tau} P'\{u/x\}|Q'}$$

$$\text{S}-\text{RES}\quad \frac{P \xrightarrow{M,\,\alpha} P',\ x\notin\alpha}{(\boldsymbol{\nu}x)P \xrightarrow{M-x,\,\alpha} (\boldsymbol{\nu}x)P'}$$

$$\text{S}-\text{OPEN}\quad \frac{P \xrightarrow{M,\,\overline{a}x} P',\ a\neq x}{(\boldsymbol{\nu}x)P \xrightarrow{M-x,\,\overline{a}\boldsymbol{\nu}x} P'}$$

Table 4: The symbolic transition semantics.

To see what it should be we must ask how substitutions involving x can enable that transition. The requirement from M is that x should not be assigned the same name as y. But since substitutions only affect free names, no substitution will assign y the same name as the bound x in $(\boldsymbol{\nu} x)P$. Therefore, the inferred transition from $(\boldsymbol{\nu} x)P$ will be possible no matter how x is treated by the substitution. Hence the conjunct $x \neq y$ should not be present in M'.

Suppose instead that M says that x is the same as some other name y, i.e., M contains a conjunct $x = y$, meaning "this transition only holds for substitutions assigning x and y the same name." Then the inferred transition from $(\boldsymbol{\nu} x)P$ will hold for *no* substitution, for the same reason. In that case M' can be chosen to be any unsatisfiable condition, for example $z \neq z$. The only other types of conjuncts in M mentioning x are the trivial $x = x$, which is always true and can be removed at will, and the contradictory $x \neq x$, which implies that the condition is unsatisfiable.

In conclusion, the condition M' can be defined as M where conjuncts $x = x$ and $x \neq y$ are removed, and conjuncts $x = y$ are replaced with an unsatisfiable conjunct $z \neq z$, for all y different from x. We denote this by "$M - x$" in Table 4. For example $(x \neq y \wedge z = u) - x$ is $(z = u)$, and $(x = y \wedge z = u) - x$ is $z \neq z$ (it is not satisfiable).

As an example inference consider the symbolic transition from

$$(\boldsymbol{\nu} x)\texttt{if } x \neq y \texttt{ then } \alpha \,.\, P$$

First by S-PREFIX

$$\alpha \,.\, P \xrightarrow{\mathbf{true}, \alpha} P$$

and then by S-MISMATCH

$$\texttt{if } x \neq y \texttt{ then } \alpha \,.\, P \xrightarrow{x \neq y, \alpha} P$$

and finally by S-RES, since $(x \neq y) - x = \mathbf{true}$ and assuming $x \notin \alpha$:

$$(\boldsymbol{\nu} x)\texttt{if } x \neq y \texttt{ then } \alpha \,.\, P \xrightarrow{\mathbf{true}, \alpha} (\boldsymbol{\nu} x)P$$

In a similar way we get that

$$(\boldsymbol{\nu} x)\texttt{if } x = y \texttt{ then } \alpha \,.\, P \xrightarrow{z \neq z, \alpha} (\boldsymbol{\nu} x)P$$

In other words, the first transition holds for all substitutions and the second for no substitution, i.e., is never possible. This is as it should since no substitution can make x and y equal in an agent where x is bound.

We can now state how the symbolic semantics corresponds to the transitions in Section 4. Write $\sigma \models M$ to denote that for any conjunct $a = b$ in M it holds that $\sigma(a) = \sigma(b)$ and for any conjunct $a \neq b$ it holds that $\sigma(a) \neq \sigma(b)$. The correspondence between the semantics is that $P \xrightarrow{M,\alpha} P'$ implies that for all σ such that $\sigma \models M$ there is a transition $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. Conversely, if $P\sigma \xrightarrow{\beta} P''$ then for some M, α, P' it holds $P \xrightarrow{M,\alpha} P'$ where $\sigma \models M$ and $P'' \equiv P'\sigma$ and $\beta = \alpha\sigma$. The proof is through induction over the inference systems for transitions. Therefore, a tool or proof system is justified in using the symbolic semantics.

5.3 The Early Semantics

As mentioned in Section 4 there is an alternative way to treat the semantics for input, the so called *early* semantics, by giving the input Prefix the rule for all u:

$$\textsc{E-INPUT}\ \frac{}{a(x)\,.\,P \xrightarrow{au} P\{u/x\}}$$

Here there is a fifth kind of action, the *free input* action of kind au, meaning "input the name u along a". Note the difference between

$P \xrightarrow{a(x)} Q$ meaning P inputs something to replace x in Q, and
$P \xrightarrow{ax} Q$ meaning P receives the name x and continues as Q

In the early semantics the COM rule needs a modification, since the input action in the premise refers to the transmitted name:

$$\textsc{E-COM}\ \frac{P \xrightarrow{au} P', \quad Q \xrightarrow{\overline{a}u} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

There is no substitution in the conclusion since that has already been performed at an earlier point in the inference of this transition, namely when the input action was inferred. Hence the name "early" semantics. Consequently the semantics in Section 4 is called the *late* semantics: the substitution emanating from an interaction is inferred as late as possible, namely when inferring the interaction in the COM-rule. In the early semantics there is no need for the bound input actions, so there will still only be four kinds of action. However, if scope extension in the structural congruence is omitted in favour of the CLOSE rule as described in Section 5.1 then the early semantics also needs the bound input actions, and hence all five kinds.

To see the similarity and difference between late and early, consider the inference of an interaction between $a(x)\,.\,P$ and $\overline{a}u\,.\,Q$. Both semantics yield the same τ-transition. First the late semantics:

$$\textsc{COM}\ \frac{\textsc{PREFIX}\ \dfrac{}{a(x)\,.\,P \xrightarrow{a(x)} P} \qquad \textsc{PREFIX}\ \dfrac{}{\overline{a}u\,.\,Q \xrightarrow{\overline{a}u} Q}}{a(x)\,.\,P \mid \overline{a}u\,.\,Q \xrightarrow{\tau} P\{u/x\} \mid Q}$$

Then the early semantics:

$$\textsc{E-COM}\ \dfrac{\textsc{E-INPUT}\ \dfrac{}{a(x)\,.\,P \xrightarrow{au} P\{u/x\}} \qquad \textsc{PREFIX}\ \dfrac{}{\overline{a}u\,.\,Q \xrightarrow{\overline{a}u} Q}}{a(x)\,.\,P \mid \overline{a}u\,.\,Q \xrightarrow{\tau} P\{u/x\} \mid Q}$$

The τ-transitions that can be inferred with the early semantics are exactly those that can be inferred with the late semantics. The proof of this is by induction over the depth of inference of the transition. The induction hypothesis needs to state that not only τ actions but also input and output actions correspond in the two semantics, where the correspondence between input and free input is that $P \xrightarrow{au} P'$ iff $\exists P'', w : P \xrightarrow{x(w)} P'' \wedge P' \equiv P''\{u/w\}$.

In view of this it is a matter of taste which semantics to adopt. It could be argued that the early semantics more closely follows an operational intuition since, after all, an agent performs an input action only when it actually receives a particular value. On the other hand, experimental evidence indicates that proof systems and decision procedures using the late semantics are slightly more efficient, and as we shall see in Section 7 it allows a wider spectrum of behavioural equivalences.

5.4 Reductions

Another idea for the operational semantics is to represent interactions using an unlabelled transition system. The idea is that $P \longrightarrow Q$, pronounced "P reduces to Q", is the same as $P \xrightarrow{\tau} Q$. The difference is that reductions are inferred directly from the syntax of the agent, as opposed to τ-transitions which are inferred from input and output transitions. This is accomplished by a counterpart to the COM rule which explicitly mentions the Prefixes that give rise to the reduction:

$$\textsc{R-COM}\ \dfrac{}{(\cdots + a(x)\,.\,P) \mid (\cdots + \overline{a}u\,.\,Q) \longrightarrow P\{u/x\} \mid Q}$$

In addition, the rules PAR and RES need counterparts for reductions; they are quite simple since there is no label on the transition:

$$\textsc{R-PAR}\ \dfrac{P \longrightarrow P'}{P|Q \longrightarrow P'|Q} \qquad \textsc{R-RES}\ \dfrac{P \longrightarrow P'}{(\boldsymbol{\nu} x)P \longrightarrow (\boldsymbol{\nu} x)P'}$$

and of course the structural rule is as before:

$$\textsc{R-STRUCT}\ \dfrac{P' \equiv P,\ P \longrightarrow Q,\ Q \equiv Q'}{P' \longrightarrow Q'}$$

That's it! Those four rules suffice for a subcalculus, formed by the agents satisfying:

1. All Sums are guarded Sums, i.e., of kind $\alpha_1 . P_1 + \cdots + \alpha_n . P_n$.

2. There are no unguarded `if`-operators.

3. There are no τ-Prefixes.

These restrictions are not very severe. In practice Sums are almost always guarded (this excludes agents like $(P|Q)+R$ which are seldom used). The second condition is easy to satisfy since any `if`-operator not under an input Prefix can be evaluated; the agent `if` $x = y$ `then` P can be replaced by P if $x = y$ and by $\mathbf{0}$ otherwise, and conversely for `if` $x \neq y$ `then` P. The importance of the two first conditions is that structural congruence can always be used to pull the two active Prefixes into a position where R-COM is possible. The third condition is also not dramatic. The silent Prefix is derivable in the sense that $(\boldsymbol{\nu} a)(a . P \mid \overline{a})$ behaves exactly like $\tau . P$.

The fact that $\longrightarrow$ corresponds to $\xrightarrow{\tau}$ in the ordinary semantics is then easily proved by induction on inference of $\longrightarrow$. Each such inference only uses R-COM once and R-STRUCT a number of times.

Clearly, for ease of presentation the reduction semantics is superior to the labelled transition semantics, and is therefore often taken as a point of departure when describing the π-calculus. It is then important to remember that the price for the simplicity is loss of information: the reduction semantics records only the completed interactions within an agent, and ignores the potential that agent shows for interacting with an environment. For example $\overline{a}u . \mathbf{0}$ and $\overline{b}u . \mathbf{0}$ and $(\boldsymbol{\nu} u)\overline{a}u . \mathbf{0}$ all have no reductions and are therefore in some sense semantically identical, even though they have different "potential" interactions.

For this reason proof systems and decision procedures based directly on the reduction semantics are not optimal. However, one great advantage is that it works equally well for the higher-order calculus (Section 3.6), where a labelled semantics is complicated with agents decorating the arrows.

5.5 Abstractions and Concretions

The input transition $P \xrightarrow{a(x)} Q$ can intuitively be thought of as $P \xrightarrow{a} \lambda x Q$ where $\lambda x Q$ is a function from names to agents. This idea can be elaborated to give an alternative presentation of the transition system. It requires a few more definitions but the rules become simpler.

Define *agent abstractions*, ranged over by F, to be of kind $\lambda x P$, and dually *agent concretions*, ranged over by C, to be of kind $[x]P$. The latter is just a notation for the pair (x, P) and allows the output transition $P \xrightarrow{\overline{a}x} Q$ to be written $P \xrightarrow{\overline{a}} [x]Q$. We write an input Prefix $a(x) . P$ as $a . F$, where $F = \lambda x P$, and similarly an output Prefix $\overline{a}x . P$ as $\overline{a} . C$, with $C = [x]P$. Let agents, abstractions and concretions collectively be called the *extended agents*, ranged over by E, and let c range over names x, overlined names $\overline{x}$, and τ. A transition will then be written $P \xrightarrow{c} E$.

We further allow Restrictions $(\boldsymbol{\nu}x)$ to operate on abstractions and concretions with the additional scope extension laws

$$(\boldsymbol{\nu}y)\lambda xP \;\equiv\; \lambda x(\boldsymbol{\nu}y)P \quad\text{and}\quad (\boldsymbol{\nu}y)[x]P \;\equiv\; [x](\boldsymbol{\nu}y)P$$

provided $x \neq y$; we also allow alpha-conversion in the usual sense, where λx (but not $[x]$) counts as a binding occurrence of x. And we define Parallel composition for extended agents by

$$\begin{array}{lcl} P \mid \lambda xQ & \equiv & \lambda x(P|Q) \\ P \mid [u]Q & \equiv & [u](P|Q) \\ P \mid (\boldsymbol{\nu}x)[x]Q & \equiv & (\boldsymbol{\nu}x)[x](P|Q) \end{array}$$

where $x \notin \mathtt{fn}(P)$. With these conventions, abstractions are structurally congruent to forms of kind λxP, and concretions are congruent to one of the two forms $[x]P$ or $(\boldsymbol{\nu}x)[x]P$, where P is an agent.

The transition rules for extended agents are given in Table 5. It is interesting to compare with the standard semantics in Table 3. The new rules are simpler mainly in two ways. First, since transitions carry no objects, X-PAR does not need to deal with a side condition on bound objects. Second, the rule X-RES fulfils the function of both RES and OPEN. The price for this simplification is the added complexity of the structural congruence which here has extra rules dealing with abstractions and concretions. To see this connection consider a derivation of

$$(\boldsymbol{\nu}u)(\overline{a}u\,.\,P \mid u(b)\,.\,Q) \xrightarrow{\overline{a}\boldsymbol{\nu}u} P \mid u(b)\,.\,Q$$

which uses PREFIX, PAR and OPEN in the standard semantics. With extended agents the inference goes by X-PREFIX and X-PAR to derive

$$\overline{a}\,.\,[u]P \mid u\,.\,\lambda bQ \xrightarrow{\overline{a}} [u]P \mid u\,.\,\lambda bQ$$

and then through X-STRUCT, the new part about Parallel composition of extended agents,

$$\overline{a}\,.\,[u]P \mid u\,.\,\lambda bQ \xrightarrow{\overline{a}} [u](P \mid u\,.\,\lambda bQ)$$

and finally X-RES

$$(\boldsymbol{\nu}u)(\overline{a}\,.\,[u]P \mid u\,.\,\lambda bQ) \xrightarrow{\overline{a}} (\boldsymbol{\nu}u)[u](P \mid u\,.\,\lambda bQ)$$

In conclusion, the use of extended agents is conceptually attractive since some of the “bookkeeping” activity in an inference, like keeping track of bound names and their scope, is factored out from the transition rules and included in the structural congruence. Also, the representation of transitions in these rules has proved suitable for automatic tools, in particular if used in conjunction with a symbolic semantics as in Section 5.2, where transitions are of the kind $P \xrightarrow{M,\,c} E$. The drawback is that the format is more unfamiliar to those accustomed to other process algebras.

$$\text{X} - \text{STRUCT} \quad \frac{P' \equiv P,\ P \xrightarrow{c} E,\ E \equiv E'}{P' \xrightarrow{c} Q'}$$

$$\text{X} - \text{PREFIX} \quad \frac{}{c\,.\,E \ \xrightarrow{c}\ E}$$

$$\text{X} - \text{SUM} \quad \frac{P \xrightarrow{c} E}{P + Q \xrightarrow{c} E}$$

$$\text{X} - \text{MATCH} \quad \frac{P \xrightarrow{c} E}{\texttt{if}\ x = x\ \texttt{then}\ P \xrightarrow{c} E}$$

$$\text{X} - \text{MISMATCH} \quad \frac{P \xrightarrow{c} E,\ x \neq y}{\texttt{if}\ x \neq y\ \texttt{then}\ P \xrightarrow{c} E}$$

$$\text{X} - \text{PAR} \quad \frac{P \xrightarrow{c} E}{P|Q \xrightarrow{c} E|Q}$$

$$\text{X} - \text{COM} \quad \frac{P \xrightarrow{a} \lambda x P',\ Q \xrightarrow{\overline{a}} [u]Q'}{P|Q \xrightarrow{\tau} P'\{u/x\}|Q'}$$

$$\text{X} - \text{RES} \quad \frac{P \xrightarrow{c} E,\ x \neq c, \overline{x} \neq c}{(\boldsymbol{\nu} x)P \xrightarrow{c} (\boldsymbol{\nu} x)E}$$

Table 5: Transitions for extended agents.

6 Bisimilarity and Congruence

We shall here look at one of the most fundamental behavioural equivalences, namely strong bisimilarity. It turns out not to be a congruence — it is not preserved by input prefix — but fortunately the largest contained congruence has a simple characterisation.

6.1 Bisimilarity

In most process algebras a family of equivalence relations on agents is based on bisimulations, and the π-calculus is no exception. The generic definition of a bisimulation is that it is a symmetric binary relation $\mathcal{R}$ on agents satisfying

$$P\mathcal{R}Q \text{ and } P \xrightarrow{\alpha} P' \quad \text{implies} \quad \exists Q' : \ Q \xrightarrow{\alpha} Q' \ \wedge P'\mathcal{R}Q'$$

The intuition is that if P can do an action then Q can do the same action and the derivatives lie in the same relation. Two agents are said to be bisimilar if they are related by some bisimulation; this means that they can indefinitely mimic the transitions of each other.

For the π-calculus extra care has to be taken for actions with bound objects. Consider

$$P = a(u), \qquad Q = a(x) \,.\, (\boldsymbol{\nu} v)\overline{v}u$$

Intuitively these represent the same behaviour: they can do an input along a and then nothing more. However, Q has the name u free where P has not. Therefore, x in Q cannot be alpha-converted to u, and the transition $P \xrightarrow{a(u)} \mathbf{0}$ cannot be simulated by Q. Such a difference between P and Q is not important since, if P has an action $a(u)$, then by alpha-conversion it also has a similarly derived action $a(w)$ for infinitely many w. Clearly it is sufficient for Q to simulate only the bound actions where the bound object is not free in Q. This argument applies to both input and bound output actions.

Also, input (but not bound output) actions mean that the bound object is a placeholder for something to be received. Therefore, if $P \xrightarrow{a(x)} P'$ then the behaviour of P' must be considered under all substitutions $\{u/x\}$, and we must require that for each such substitution Q' is related to P', or in other words, that they are related for each value received.

In the following we will use the phrase "$\mathtt{bn}(\alpha)$ is fresh" in a definition to mean that the name in $\mathtt{bn}(\alpha)$, if any, is different from any free name occurring in any of the agents in the definition.

Definition 1 *A* (strong) bisimulation *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

(i) *If $\alpha = a(x)$ then $\exists Q' : \ Q \xrightarrow{a(x)} Q' \ \wedge \ \forall u : \ P'\{u/x\}\mathcal{R}Q'\{u/x\}$*

(ii) *If α is not an input then $\exists Q' :\ Q \xrightarrow{\alpha} Q' \ \wedge P'\mathcal{R}Q'$*

P and Q are (strongly) bisimilar*, written $P \dot{\sim} Q$, if they are related by a bisimulation.*

It follows that $\dot{\sim}$, which is the union of all bisimulations, is a bisimulation. We also immediately have that $P \equiv Q$ implies $P \dot{\sim} Q$, by virtue of the rule STRUCT.

At this point it might be helpful to consider a few examples. Requirement (i) is quite strong as demonstrated by

$$P_1 = a(x)\,.\,P + a(x)\,.\,\mathbf{0}, \quad P_2 = a(x)\,.\,P + a(x)\,.\,\mathtt{if}\ x = u\ \mathtt{then}\ P$$

Assume that $P \dot{\not\sim} \mathbf{0}$. Then $P_1 \dot{\not\sim} P_2$ since the transition $P_1 \xrightarrow{a(x)} \mathbf{0}$ cannot be simulated by P_2. For example, $P_2 \xrightarrow{a(x)} \mathtt{if}\ x = u\ \mathtt{then}\ P$ does not suffice since, for the substitution $\{u/x\}$, the derivatives are not bisimilar. Similarly,

$$P_1 = a(x)\,.\,P + a(x)\,.\,\mathbf{0}, \quad P_2 = a(x)\,.\,P + a(x)\,.\,\mathbf{0} + a(x)\,.\,\mathtt{if}\ x = u\ \mathtt{then}\ P$$

are not bisimilar because of the transition $P_2 \xrightarrow{a(x)} \mathtt{if}\ x = u\ \mathtt{then}\ P$. Neither $\mathbf{0}$ nor P is bisimilar to $\mathtt{if}\ x = u\ \mathtt{then}\ P$ for *all* substitutions of x, since $\mathbf{0}$ fails for $\{u/x\}$ and P fails for the identity substitution. Using the intuition that an input $P \xrightarrow{a(x)} Q$ can be thought of as $P \xrightarrow{a} \lambda x Q$, clause (i) says that the derivatives (which are functions from names to agents) must be pointwise bisimilar.

As expected we have

$$a \mid \overline{b} \ \dot{\sim}\ a\,.\,\overline{b} + \overline{b}\,.\,a$$

and

$$a \mid \overline{a} \ \dot{\sim}\ a\,.\,\overline{a} + \overline{a}\,.\,a + \tau$$

This demonstrates that $\dot{\sim}$ is not in general closed under substitutions, i.e., from $P \dot{\sim} Q$ we cannot conclude that $P\sigma \dot{\sim} Q\sigma$. For this reason we have

$$c(a)\,.\,(a \mid \overline{b}) \ \dot{\not\sim}\ c(a)\,.\,(a\,.\,\overline{b} + \overline{b}\,.\,a)$$

demonstrating that $\dot{\sim}$ is not preserved by input Prefix, i.e., that from $P \dot{\sim} Q$ we cannot conclude that $a(x)\,.\,P \dot{\sim} a(x)\,.\,Q$.

However, we have the following results:

Proposition 2 *If $P \dot{\sim} Q$ and σ is injective then $P\sigma \dot{\sim} Q\sigma$.*

Proposition 3 $\dot{\sim}$ *is an equivalence.*

Proposition 4 $\dot{\sim}$ *is preserved by all operators except input Prefix.*

Proposition 2 is proved by establishing that transitions are preserved by injective substitutions, i.e., that for injective σ,

$$P \xrightarrow{\alpha} P' \text{ implies } P\sigma \xrightarrow{\alpha\sigma} P'\sigma$$

where the substitution σ applied to an action α is defined not to affect $\mathtt{bn}(\alpha)$. This result is established by induction of the inference of $P \xrightarrow{\alpha} P'$ and each rule in Table 3 gives rise to one induction step. It is then straightforward to show that $\{(P\sigma, Q\sigma) : P \dot{\sim} Q, \sigma \text{ injective}\}$ is a bisimulation.

Proposition 3 is not difficult. Reflexivity and symmetry are immediate and for transitivity it suffices to show that $\dot{\sim}\ \dot{\sim}$ is a strong bisimulation.

The proof of Proposition 4 goes by examining each operator in turn. It is a bit complicated for the cases of Restriction and Parallel. For example, we would like to show that

$$\{((\boldsymbol{\nu}x)P, (\boldsymbol{\nu}x)Q) : P \dot{\sim} Q\} \cup \dot{\sim}$$

is a bisimulation, by examining the possible transitions from $(\boldsymbol{\nu}x)P$, where the rules RES and OPEN are applicable. Unfortunately this simple idea does not quite work since also the STRUCT rule is applicable, so we have to include all structurally congruent pairs in the bisimulation. A proof idea is the following. Say that an agent is *normal* if all bound names are distinct and all unguarded Restrictions are at the top level, i.e., of the form $(\boldsymbol{\nu}\tilde{x})P$ where P has no unguarded Restrictions. By Proposition 1 any P is structurally congruent to a normal agent. Now extend the transition rules with the symmetric counterparts of PAR and COM; in view of the commutativity in the structural congruence this does not affect the transitions. That makes it possible to prove the lemma

> If $P \xrightarrow{\alpha} P'$ and $P \equiv N$ where N is normal, then by an inference of no greater depth, $N \xrightarrow{\alpha} N'$ and $P' \equiv N'$.

The proof is through induction on the inference of $P \equiv N$ and involves a tedious examination of all rules for the structural congruence. Now we can show that the relation on normal agents

$$\{((\boldsymbol{\nu}\tilde{x})(P|Q), (\boldsymbol{\nu}\tilde{x})(P'|Q')) : P \dot{\sim} P', Q \dot{\sim} Q'\}$$

is a bisimulation up to $\dot{\sim}$ (as explained below), by showing that an action from one side is simulated by an action from the other side, through induction on the inference of the action. There are eight inductive steps through combinations of the PAR, COM, RES and OPEN laws to consider. All cases are routine. Because of the lemma, actions derived through the STRUCT rule can be ignored in the induction since they also have a shorter inference. Finally we can take $P = P' = \mathbf{0}$ to show that Restriction preserves bisimilarity, and $\tilde{x}$ the empty sequence to show the same for Parallel.

The notion of "bisimulation up to" is a standard proof trick in these circumstances. It involves modifying Definition 1 by replacing $\mathcal{R}$ in the consequents of

(i) and (ii) by $\stackrel{\cdot}{\sim} \mathcal{R} \stackrel{\cdot}{\sim}$, i.e., the relational composition of $\stackrel{\cdot}{\sim}$, $\mathcal{R}$, and $\stackrel{\cdot}{\sim}$. The requirement is thus weakened: the derivatives must be bisimilar to a pair in $\mathcal{R}$, in contrast to Definition 1 which requires that the derivatives themselves lie in $\mathcal{R}$. It then holds that if two agents are related by a bisimulation up to $\stackrel{\cdot}{\sim}$ then they are bisimilar. The proof of this is very similar to the corresponding result for other process algebras.

6.2 Congruence

The fact that bisimilarity is not preserved by input Prefix makes us seek the largest congruence included in bisimilarity. The definition is fortunately simple:

Definition 2 *Two agents P and Q are* (strongly) congruent*, written $P \sim Q$, if $P\sigma \stackrel{\cdot}{\sim} Q\sigma$ for all substitutions σ.*

For an example, although $a \mid \overline{b} \stackrel{\cdot}{\sim} a\,.\,\overline{b} + \overline{b}\,.\,a$ these agents are not strongly congruent since the substitution $\{a/b\}$ makes them non-bisimilar. However, we do have

$$a \mid \overline{b} \;\sim\; a\,.\,\overline{b} + \overline{b}\,.\,a + \texttt{if}\ a = b\ \texttt{then}\ \tau$$

and the reader may by this example realise the crucial role played by the Match operator in an expansion law which reduces a Parallel composition to a Sum.

Proposition 5 *Strong congruence is the largest congruence in bisimilarity.*

We first have to show that strong congruence is a congruence, i.e., that it is preserved by all operators. For all operators except input Prefix this is immediate from the definition and Proposition 4. For input Prefix we show that

$$\{(a(x)\,.\,P, a(x)\,.\,P') : P \sim P'\} \cup \sim$$

is a strong bisimulation and closed under substitutions. The latter is trivial since $\sim$ is closed under substitutions. For the bisimulation part, the transition to consider is $a(x)\,.\,P \xrightarrow{a(x)} P$ which obviously is simulated by $a(x)\,.\,P' \xrightarrow{a(x)} P'$; we have to show that for all u, $P\{u/x\}$ is related to $P'\{u/x\}$, and this follows from $P \sim P'$.

We next show that it is the largest such congruence. Assume that P and Q are related by some congruence $\sim'$ in $\stackrel{\cdot}{\sim}$. We shall show that this implies that $P\sigma \stackrel{\cdot}{\sim} Q\sigma$ for any σ, in other words that $\sim' \subseteq \sim$. Without loss of generality we can assume that $\sigma(x) \neq x$ for only finitely many names x since the effect of σ on names not free in P or Q is immaterial. So let $\sigma = \{y_1 \ldots y_n \,/\, x_1 \ldots x_n\}$. Choose $a \notin \texttt{fn}(P, Q)$. By $P \sim' Q$ and the fact that $\sim'$ is a congruence we get

$$(\boldsymbol{\nu} a)(a(x_1 \ldots x_n)\,.\,P \mid \overline{a}\langle y_1 \ldots y_n\rangle) \quad \sim' \quad (\boldsymbol{\nu} a)(a(x_1 \ldots x_n)\,.\,Q \mid \overline{a}\langle y_1 \ldots y_n\rangle)$$

and by the fact that $\sim'$ is in $\stackrel{\cdot}{\sim}$ we get, following the τ-transition from both sides, that $P\sigma \stackrel{\cdot}{\sim} Q\sigma$.

7 Variants of Bisimilarity

There are several ways in which the definition of equivalence can be varied. We here consider some variants that are particular to the π-calculus, in the sense that they differ in how the name substitutions arising from input are treated. We also briefly consider the weak bisimulation equivalences.

7.1 Early Bisimulation

Bisimulations can also be defined from the early semantics in Section 5.3. This leads to an equivalence which is slightly larger than $\dot{\sim}$, i.e., it equates more agents, although that difference is hardly ever of practical significance.

For clarity let the transitions in the early semantics be indexed by E, so for example we have $a(x) \,.\, P \xrightarrow{au}_{\mathrm{E}} P\{u/x\}$. As mentioned in that section the τ-transitions are the same with the late and early semantics, in other words $\xrightarrow{\tau}$ is the same as $\xrightarrow{\tau}_{\mathrm{E}}$, and similarly for output transitions. So the distinction is only relevant for input transitions.

In the early input transition the object represents the value received, and therefore there is no need for clause (i) in the definition of bisimulation with arbitrary instantiation of the bound object. The definition of early bisimulation is in that respect simpler.

Definition 3 *A* (strong) early bisimulation *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha}_{\mathrm{E}} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

$$\exists Q' :\ Q \xrightarrow{\alpha}_{\mathrm{E}} Q' \ \wedge P'\mathcal{R}Q'$$

P and Q are (strongly) early bisimilar*, written $P \dot{\sim}_{\mathrm{E}} Q$, if they are related by an early bisimulation.*

To avoid confusion we will in the following refer to Definition 1 as *late* bisimulation and $\dot{\sim}$ as *late* bisimilarity, and correspondingly to $\sim$ as *late* congruence. A simple example highlights the difference between late and early.

$$P_1 = a(x) \,.\, P + a(x) \,.\, \mathbf{0}, \quad P_2 = a(x) \,.\, P + a(x) \,.\, \mathbf{0} + a(x) \,.\, \mathtt{if}\ x = u\ \mathtt{then}\ P$$

The summand $a(x) \,.\, \mathtt{if}\ x = u\ \mathtt{then}\ P$ is the only difference between P_1 and P_2. For late bisimilarity this difference is important. It gives rise to the transition $\xrightarrow{a(x)}$ leading to $\mathtt{if}\ x = u\ \mathtt{then}\ P$ and no transition in P_1 can simulate this for all instances of x. But it is not important for early bisimilarity, and indeed $P_1 \dot{\sim}_{\mathrm{E}} P_2$. To see this consider the transitions that the extra summand generates, they are

$$P_2 \xrightarrow{aw}_{\mathrm{E}} \mathtt{if}\ w = u\ \mathtt{then}\ P\{w/x\}$$

for all w. For the case that $w = u$, P_1 simulates through $P_1 \xrightarrow{au}_{\mathrm{E}} P\{u/x\}$, and the derivatives are bisimilar since `if` $u = u$ `then` $P\{u/x\}$ has the same transitions as $P\{u/x\}$. For the other cases, i.e., $w \neq u$, P_1 simulates with the transition $P_1 \xrightarrow{aw}_{\mathrm{E}} \mathbf{0}$, and the derivatives are bisimilar since `if` $w = u$ `then` $P\{w/x\}$ has no transitions.

Again it helps to think of late input transitions as resulting in functions from names to agents. Consider the pictorial representation of P_1 and P_2 in Figure 1 where late input transitions are arrows and instantiations of the bound object are dashed lines (for simplicity we consider only two instantiations u and w where in reality there would be one for each name). Late bisimulation requires that the derivatives simulate after each input $\xrightarrow{a}$ and then again after each instantiation. Therefore P_1 and P_2 are not late bisimilar. In contrast, an early input means the same as an input $\xrightarrow{a}$ *and an instantiation* in the same go. The intermediate level, before the instantiation, does not exist in the early semantics. Therefore P_1 and P_2 are early bisimilar. For example the bottom alternative from P_2, going through $\xrightarrow{aw}$ leading to `if` $w = u$ `then` $P\{w/u\}$, is simulated by P_1 going through $\xrightarrow{aw}$ to $\mathbf{0}$.

It is also possible to define the early bisimulation through the late semantics. Figure 1 and the definition of late bisimulation provide the intuition: Consider the part of the definition (clause (i)) which says

there should exist *some* Q' such that for *all* u ...

For early bisimulation, the corresponding part should say

for *all* u there should exist *some* Q' such that ...

This commutation of the quantifiers weakens the statement, since the choice of Q' here can depend on u. And indeed, with it we obtain a bisimulation which precisely coincides with early bisimulation:

Definition 4 *An* early bisimulation with late semantics *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

(i) *If $\alpha = a(x)$ then $\forall u\ \exists Q' :\ Q \xrightarrow{a(x)} Q'\ \wedge\ P'\{u/x\}\mathcal{R}Q'\{u/x\}$*

(ii) *If α is not an input then $\exists Q' :\ Q \xrightarrow{\alpha} Q'\ \wedge P'\mathcal{R}Q'$*

It is then easy to prove that a relation is an early bisimulation precisely when it is an early bisimulation with late semantics. So it is a matter of convenience which of the two definitions to use. The converse idea (to capture late bisimilarity using early semantics) is however not possible.

Early bisimilarity satisfies the same propositions as late bisimilarity in Section 6: it is preserved by injective substitutions and all operators except input

Late input and instantiation:

$$\begin{tikzcd}
 & & \mathbf{0} \\
 & \lambda x\mathbf{0} \arrow[ur, dotted, "u"] \arrow[dr, dotted, "w"'] & \\
 & & \mathbf{0} \\
P_2 \arrow[uur, "a"] \arrow[dr, "a"'] \arrow[ddddr, "a"] & & P\{u/x\} \\
 & \lambda xP \arrow[ur, dotted, "u"] \arrow[dr, dotted, "w"'] & \\
 & & P\{w/x\} \\
 & & \texttt{if } u = u \texttt{ then } P\{u/x\} \\
 & \lambda x \texttt{ if } x = u \texttt{ then } P \arrow[ur, dotted, "u"] \arrow[dr, dotted, "w"'] & \\
 & & \texttt{if } w = u \texttt{ then } P\{w/x\}
\end{tikzcd}$$

$$\begin{tikzcd}
 & & \mathbf{0} \\
 & \lambda x\mathbf{0} \arrow[ur, dotted, "u"] \arrow[dr, dotted, "w"'] & \\
 & & \mathbf{0} \\
P_1 \arrow[uur, "a"] \arrow[dr, "a"'] & & P\{u/x\} \\
 & \lambda xP \arrow[ur, dotted, "u"] \arrow[dr, dotted, "w"'] & \\
 & & P\{w/x\}
\end{tikzcd}$$

Early input includes instantiation:

$$\begin{tikzcd}
 & \mathbf{0} \\
 & \mathbf{0} \\
P_2 \arrow[uur, "au"] \arrow[ur, "aw"] \arrow[r, "au"] \arrow[dr, "aw"] \arrow[ddr, "au"] \arrow[dddr, "aw"] & P\{u/x\} \\
 & P\{w/x\} \\
 & \texttt{if } u = u \texttt{ then } P\{u/x\} \\
 & \texttt{if } w = u \texttt{ then } P\{w/x\}
\end{tikzcd}$$

$$\begin{tikzcd}
 & \mathbf{0} \\
 & \mathbf{0} \\
P_1 \arrow[uur, "au"] \arrow[ur, "aw"] \arrow[r, "au"] \arrow[dr, "aw"] & P\{u/x\} \\
 & P\{w/x\}
\end{tikzcd}$$

Figure 1: The difference between late and early bisimilarity.
$P_1 = a(x) \, . \, P + a(x) \, . \, \mathbf{0}$, and $P_2 = P_1 + a(x) \, . \, \texttt{if } x = u \texttt{ then } P$.

Prefix; and *early congruence* $\sim_{\mathrm{E}}$, defined by $P \sim_{\mathrm{E}} Q$ if for all σ, $P\sigma \overset{\cdot}{\sim}_{\mathrm{E}} Q\sigma$, is the largest congruence in $\overset{\cdot}{\sim}_{\mathrm{E}}$. The proofs are very similar. Early congruence is larger than late congruence, i.e., $\sim \subseteq \sim_{\mathrm{E}}$, and the example in Figure 1 suffices to prove that the inclusion is strict.

It is hard to imagine any practical situation where the difference between late and early congruence is important. Since the theoretical properties are very similar the choice between them is largely a matter of taste. It could be argued that early congruence to a higher degree conforms to our operational intuition, and that the definition is less complex. On the other hand, late congruence seems to lead to more efficient verifications in automated tools.

7.2 Barbed Congruence

In some sense the notion of bisimilarity, be it late or early, assumes the observer is unrealistically powerful: since there is a difference between the actions $\overline{a}u$ and $\overline{a}\boldsymbol{\nu}u$ it is possible to directly detect if an emitted name is local. It is therefore natural to ask what is the effect of reducing this discriminatory power. This leads to the idea of *barbed bisimulation* where, intuitively, the observer is limited to sensing if an action is enabled on a given channel. Although barbed bisimilarity is weaker (i.e., equates more agents) than early bisimilarity it turns out that the congruence obtained from barbed bisimulation is the same as early congruence.

Definition 5 *A name a is* observable *at P, written $P \downarrow a$, if a is the subject of some action from P. A* barbed bisimulation *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ implies that*

(i) *If $P \xrightarrow{\tau} P'$ then $\exists Q' : Q \xrightarrow{\tau} Q' \wedge P'\mathcal{R}Q'$*

(ii) *If $P \downarrow a$ then $Q \downarrow a$*

P and Q are barbed bisimilar*, written $P \overset{\cdot}{\sim}_{\mathrm{B}} Q$, if they are related by a barbed bisimulation.*

The idea behind a barbed bisimulation is that the only unquestionable execution steps are the internal ones, i.e., the τ-actions, since these do not require participation of the environment. Therefore labelled transitions are not needed to define barbed bisimulation. It is enough with the reduction relation of Section 5.4 and the observability predicate, which like reductions can be defined directly from the syntax as $P \downarrow a$ if P contains an unguarded Prefix with subject a not under the scope of a Restriction $\boldsymbol{\nu}a$.

Barbed bisimilarity is uninteresting by itself. For example,

$$\overline{a}u \;\overset{\cdot}{\sim}_{\mathrm{B}}\; \overline{a}v \;\overset{\cdot}{\sim}_{\mathrm{B}}\; (\boldsymbol{\nu}u)\overline{a}u$$

identifying three clearly different agents. Of more interest is the congruence it generates.

Definition 6 *Two agents P and Q are* barbed congruent*, written $P \sim_{\mathrm{B}} Q$, if for all contexts $\mathcal{C}$ it holds that $\mathcal{C}(P) \stackrel{\cdot}{\sim}_{\mathrm{B}} \mathcal{C}(Q)$.*

In other words, barbed congruence is the largest congruence in barbed bisimilarity. Clearly the three agents above are not barbed congruent. Take the context

$$\mathcal{C}(P) = P \mid a(x) \, . \, \overline{x}$$

Then $\mathcal{C}(\overline{a}u) \xrightarrow{\tau} \overline{u} \downarrow u$ and this cannot be simulated by $\mathcal{C}(\overline{a}v)$ and not by $\mathcal{C}((\boldsymbol{\nu} u)\overline{a}u)$. In fact barbed congruence coincides with early congruence:

Proposition 6 $\sim_{\mathrm{B}} = \sim_{\mathrm{E}}$

This lends considerable support to early congruence as the natural equivalence relation. The main idea is to prove that $\sim_{\mathrm{B}}$ is an early bisimulation. Suppose $P \sim_{\mathrm{B}} Q$ and $P \xrightarrow{\alpha} P'$, then a cushioning context $\mathcal{C}$ can be found such that $\mathcal{C}(P) \xrightarrow{\tau} \mathcal{C}'(P')$, so $\mathcal{C}(Q)$ must simulate this transition. But $\mathcal{C}$ is cleverly constructed so that the only possible way that $\mathcal{C}(Q)$ can do this is because of a transition $Q \xrightarrow{\alpha} Q'$. We shall not go further into the proof here which is quite involved.

Variants of barbed congruence use variants of the observability predicate, leading to a spectrum of possibilities. For example, a weaker variant is to use $P \downarrow$ to mean $\exists a : P \downarrow a$, and amend clause (ii) in the definition of barbed bisimilarity to $P \downarrow$ iff $Q \downarrow$. With this definition Proposition 6 still holds, though its counterpart for weak bisimulation (cf. Section 7.4) is less clear. Omitting clause (ii) entirely Proposition 6 no longer holds. For example, then $!\tau|Q$ and $!\tau|R$ would be barbed congruent for all Q and R.

Also, in subcalculi of π the barbed congruence may be different (because there are fewer contexts). As one example, in the asynchronous subcalculus (Section 3.5) and where observations $P \downarrow a$ mean that P can do an output action along a (in the asynchronous calculus it makes sense to regard inputs as unobservable) it holds that

$$a(x) \, . \, \overline{a}x + \tau \;\sim_{\mathrm{B}}\; \tau$$

7.3 Open Bisimulation

As has been demonstrated neither late nor early bisimilarity is a congruence; to gain a congruence we must additionally require bisimilarity under all substitutions of names. Interestingly, there is a way to define a congruence directly through bisimulations. This is the idea behind *open* bisimulation, which incorporates the quantification over all substitutions in the clause for bisimulation. This leads to a congruence finer than late congruence (i.e., it equates fewer agents) although that difference is not very significant in practice.

To begin we consider a subcalculus without Restriction and Mismatch. The definition will subsequently be extended to accommodate Restriction (no simple extension for Mismatch is known).

Definition 7 *A* (strong) open bisimulation *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following for all substitutions σ: $P\mathcal{R}Q$ and $P\sigma \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

$$\exists Q' : \; Q\sigma \xrightarrow{\alpha} Q' \text{ and } P'\mathcal{R}Q'$$

P and Q are (strongly) open bisimilar*, written $P \overset{\cdot}{\sim}_{\mathrm{O}} Q$, if they are related by an open bisimulation.*

A technical remark: the condition on $\mathtt{bn}(\alpha)$ is unnecessary in view of the fact that we deal with a subcalculus without Restriction, but it is harmless and is included for the sake of symmetry with other similar definitions.

In open bisimulation there is no need for a special treatment of input actions since the quantification over substitutions recurs anyway when further transitions are examined: If $P \xrightarrow{a(x)} P'$ is simulated by $Q \xrightarrow{a(x)} Q'$, then the requirement implies that all transitions from $P'\sigma$ must be simulated by $Q'\sigma$ for all σ, including those substitutions that instantiate x to another name. This also demonstrates the pertinent difference between open and late: where late requires the agents to continue to bisimulate under all substitutions for the bound input object, open requires them to bisimulate under *all* substitutions — not only those affecting the bound input object.

It is straightforward to show that an open bisimulation is also a late bisimulation, and hence $P \overset{\cdot}{\sim}_{\mathrm{O}} Q$ implies $P \overset{\cdot}{\sim} Q$. Furthermore, directly from the definition we get that open bisimilarity is closed under substitution, in the sense that $P \overset{\cdot}{\sim}_{\mathrm{O}} Q$ implies $P\sigma \overset{\cdot}{\sim}_{\mathrm{O}} Q\sigma$; together these facts give us that $P \overset{\cdot}{\sim}_{\mathrm{O}} Q$ implies $P \sim Q$. But the converse is not true as demonstrated by the following example:

$$\begin{aligned} Q &\equiv \tau + \tau\,.\,\tau \\ R &\equiv \tau + \tau\,.\,\tau + \tau\,.\,\mathtt{if}\ x = y\ \mathtt{then}\ \tau \end{aligned}$$

Q and R are late congruent. To see that $Q\sigma$ is late bisimilar to $R\sigma$ for all σ, consider two possibilities for σ. Either $\sigma(x) = \sigma(y)$, in which case $R\sigma \xrightarrow{\tau}$ $\mathtt{if}\ \sigma(x) = \sigma(y)\ \mathtt{then}\ \tau$ is simulated by $Q\sigma \xrightarrow{\tau} \tau$; or $\sigma(x) \neq \sigma(y)$, in which case it is simulated by $Q\sigma \xrightarrow{\tau} \mathbf{0}$. But Q and R are not open bisimilar. To see this, choose σ as the identity substitution and consider $R \xrightarrow{\tau} \mathtt{if}\ x = y\ \mathtt{then}\ \tau$. There are two possibilities for Q to make a τ transition. One is $Q \xrightarrow{\tau} \mathbf{0}$. But then open bisimulation requires that the two derivatives $\mathtt{if}\ x = y\ \mathtt{then}\ \tau$ and $\mathbf{0}$ bisimulate again under *all* substitutions σ', and this does not hold for $\sigma' = \{x/y\}$ since $(\mathtt{if}\ x = y\ \mathtt{then}\ \tau)\{x/y\} \xrightarrow{\tau} \mathbf{0}$ whereas $\mathbf{0}\{x/y\}$ has no transition. Similarly, the other transition $Q \xrightarrow{\tau} \tau$ fails for the identity substitution, since $\mathtt{if}\ x = y\ \mathtt{then}\ \tau$ has no transition, in contrast to τ.

This example may seem artificial and, as for the difference between late and early, the difference between open and late is probably of little practical significance. Open bisimilarity is an equivalence and a congruence, so by analogy with the other equivalences, writing $\sim_{\mathrm{O}}$ for the largest congruence in $\dot{\sim}_{\mathrm{O}}$, we get $\dot{\sim}_{\mathrm{O}} = \sim_{\mathrm{O}}$ (the proofs are technically similar to the proof for late congruence). It can also be characterised in a few other informative ways. One is through substitution closed ground bisimulations. A ground bisimulation is a relation $\mathcal{R}$ satisfying

$$P\mathcal{R}Q \text{ and } P \xrightarrow{\alpha} P' \text{ implies } \exists Q' : \; Q \xrightarrow{\alpha} Q' \text{ and } P'\mathcal{R}Q'$$

In other words, it does not introduce substitutions at all. Now it holds that $\dot{\sim}_{\mathrm{O}}$ is the largest substitution closed (i.e., $P\mathcal{R}Q$ implies $P\sigma\mathcal{R}Q\sigma$) ground bisimulation. Another characterisation is with *dynamic* bisimulations: these are relations satisfying

$$P\mathcal{R}Q \text{ and } \mathcal{C}(P) \xrightarrow{\alpha} P' \text{ implies } \exists Q' : \; \mathcal{C}(Q) \xrightarrow{\alpha} Q' \text{ and } P'\mathcal{R}Q'$$

for all contexts $\mathcal{C}$. It can be shown that dynamic bisimilarity is the same as open bisimilarity.

At first it may seem that the Restriction operator behaves well with open bisimulation since the substitutions only affect free names. Consider

$$P \;=\; (\boldsymbol{\nu}x)\texttt{if}\; x = y \;\texttt{then}\; \tau \quad \text{and} \quad Q \;=\; \mathbf{0}$$

Neither $P\sigma$ nor $Q\sigma$ have any transitions, for any substitution σ, since the substitution cannot affect the bound name x. Therefore $P \;\dot{\sim}_{\mathrm{O}}\; Q$. Similarly with

$$P \;=\; (\boldsymbol{\nu}x)\tau\,.\,\texttt{if}\; x = y \;\texttt{then}\; \tau \quad \text{and} \quad Q \;=\; \tau$$

we have $P \;\dot{\sim}_{\mathrm{O}}\; Q$, since $P \xrightarrow{\tau} (\boldsymbol{\nu}x)\texttt{if}\; x = y \;\texttt{then}\; \tau$ and this derivative is similarly immune to substitutions. The problem arises because of bound output actions. Consider

$$P \;=\; (\boldsymbol{\nu}x)\overline{a}x\,.\,\texttt{if}\; x = y \;\texttt{then}\; \tau \quad \text{and} \quad Q \;=\; (\boldsymbol{\nu}x)\overline{a}x$$

Here $P \xrightarrow{\overline{a}\boldsymbol{\nu}x} \texttt{if}\; x = y \;\texttt{then}\; \tau$. Clearly Q cannot simulate by $Q \xrightarrow{\overline{a}\boldsymbol{\nu}x} \mathbf{0}$ because the two derivatives, $\texttt{if}\; x = y \;\texttt{then}\; \tau$ and $\mathbf{0}$, are not bisimilar for the substitution $\{x/y\}$. Here the bound output action has lifted the Restriction so x becomes vulnerable to a substitution.

But we would certainly want to identify P and Q. The name x in P is a local name and no matter what an environment does on receiving it, that name cannot become the same as y, so the `if` test will never be positive. Also P and Q are late congruent. Therefore in the presence of Restriction, Definition 7 is too discriminating: it considers too many substitutions.

One way to amend it is to use the auxiliary concept of *distinctions*. The intuition behind a distinction is to record those pairs of names that will always be distinct, no matter what actions take place. Definition 7 is modified so that it only considers substitutions that respect distinctions, in the sense that two names required to be distinct are not substituted by the same name. Following a bound output transition the derivatives are related under a larger distinction, representing the fact that the bound object will be kept different from any other name.

Formally a distinction is a symmetric and irreflexive binary relation on names. In the following we let D range over finite distinctions (infinite distinctions are not necessary and may cause problems with alpha-conversion if they include all names). We say that a substitution σ *respects* a distinction D if aDb implies $\sigma(a) \neq \sigma(b)$. We further let $D\sigma$ be the relation $\{(\sigma(a), \sigma(b)) : aDb\}$. When N is a set of names we let $D + (x, N)$ represent $D \cup (\{x\} \times N) \cup (N \times \{x\})$, in other words it is D extended with the fact that x is distinct from all names in N.

Definition 8 *A* distinction-indexed *family of binary relations on agents is a set of binary agent relations $\mathcal{R}_D$, one for each distinction D. A* (strong) open bisimulation *is such a family satisfying the following for all substitutions σ which respect D: $P\mathcal{R}_D Q$ and $P\sigma \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

(i) *If $\alpha = \overline{a}\boldsymbol{\nu}x$ then $\exists Q' : \; Q\sigma \xrightarrow{\overline{a}\boldsymbol{\nu}x} Q' \;\wedge\; P'\mathcal{R}_{D'}Q'$*
where $D' = D\sigma + (x, \mathtt{fn}(P\sigma, Q\sigma))$

(ii) *If α is not a bound output then $\exists Q' : \; Q\sigma \xrightarrow{\alpha} Q' \;\wedge P'\mathcal{R}_{D\sigma}Q'$*

P and Q are open D-bisimilar*, written $P \dot{\sim}_{\mathrm{O}}^{D} Q$, if they are related by R_D in an open bisimulation, and they* open bisimilar*, written $P \dot{\sim}_{\mathrm{O}} Q$ if they are* open $\emptyset$-bisimilar*.*

Here the proviso that $\mathtt{bn}(\alpha)$ is fresh means it is not in $\mathtt{fn}(P\sigma, Q\sigma)$. The overloading of " $\dot{\sim}_{\mathrm{O}}$ " is motivated by the fact that for the subcalculus without Restriction definitions 7 and 8 coincide. In essence, the distinction keeps track of what substitutions are admissible, excluding those that do not respect the distinction. After any action except a bound output, the distinction recurs in that the substituted names must still be distinct. After a bound output, additionally the bound object must be distinct from any free name in the involved agents.

As an example we can now demonstrate that $P \dot{\sim}_{\mathrm{O}} Q$ where

$$P \;=\; (\boldsymbol{\nu}x)\overline{a}x\,.\,\mathtt{if}\ x = y\ \mathtt{then}\ \tau \quad \text{and} \quad Q \;=\; (\boldsymbol{\nu}x)\overline{a}x$$

Consider the transition $P \xrightarrow{\overline{a}\boldsymbol{\nu}x}$ $\mathtt{if}\ x = y\ \mathtt{then}\ \tau$ and the simulating transition $Q \xrightarrow{\overline{a}\boldsymbol{\nu}x} \mathbf{0}$. The definition then requires, by clause (i), that $\mathtt{if}\ x = y\ \mathtt{then}\ \tau \dot{\sim}_{\mathrm{O}}^{D} \mathbf{0}$ for $D = \{(x, y), (y, x)\}$ and this is indeed the case: for a substitution σ respecting D, the agent $(\mathtt{if}\ x = y\ \mathtt{then}\ \tau)\sigma$ has no transitions.

$$\begin{tikzcd}
\dot{\sim}_{\mathrm{O}} = \sim_{\mathrm{O}} \arrow[d] & \\
\sim \arrow[dd] \arrow[dr] & \\
 & \dot{\sim} \arrow[dd] \\
\sim_{\mathrm{E}} = \sim_{\mathrm{B}} \arrow[dr] \arrow[ddr] & \\
 & \dot{\sim}_{\mathrm{E}} \arrow[d] \\
 & \dot{\sim}_{\mathrm{B}}
\end{tikzcd}$$

Figure 2: Barbed, early, late and open bisimilarity and the corresponding congruences. An arrow means strict inclusion, and when it goes from the left column to the right it also means "largest congruence in".

Distinctions will however not admit Mismatch in a sensible way. For example, we would want to equate the agents

$$\texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,\texttt{if}\ x \neq y\ \texttt{then}\ \tau \quad \text{and} \quad \texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,\tau$$

but the requirement that derivatives bisimulate under all substitutions will make them inequivalent. In order to admit Mismatch a more fundamental redefinition of the semantics is necessary, where the Mismatch is explicitly noted in the transition, for example by using a symbolic semantics as in Section 5.2.

In conclusion, the definition of $\dot{\sim}_{\mathrm{O}}$ loses some of its simplicity in the presence of Restriction and it is still not applicable with Mismatch. It gains interest in that it appears more efficient for automatic verification and in that distinctions sometimes have an independent motivation to represent "constant" names such as values in some global data type or ports which are necessarily distinct. For example, if Booleans are represented by two globally available names t and f then clearly it is uninteresting to consider substitutions making them equal. Similarly, the question whether a buffer $B(i,o)$ with input port i and output o behaves as another buffer $B_2(i,o)$ might be relevant only for the case that $i \neq o$.

The various definitions of strong bisimilarity and congruence are related in Figure 2. In summary, barbed bisimilarity is much too weak to be interesting, while the difference between early and late is very small and has to do with the precise interpretation of input transitions. Barbed congruence is the same as early congruence. The early and late congruences are obtained as bisimilarity for all substitutions, and again the difference between them is small. Open bisimilarity (defined only for the subcalculus without Mismatch) is a congruence and is more discriminating than late congruence since it requires bisimilarity for all substitutions after every transition.

7.4 Weak Bisimulation

The main idea of weak bisimulation is that τ transitions are regarded as unobservable. We therefore define $\Longrightarrow$ to mean $(\xrightarrow{\tau})^*$, i.e., zero or more τ transitions, $\overset{\alpha}{\Longrightarrow}$ to mean $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and $\overset{\widehat{\alpha}}{\Longrightarrow}$ to mean $\overset{\alpha}{\Longrightarrow}$ if $\alpha \neq \tau$ and $\Longrightarrow$ if $\alpha = \tau$. The intention is to modify strong bisimilarity in the standard way, replacing $\xrightarrow{\alpha}$ by $\overset{\widehat{\alpha}}{\Longrightarrow}$. The only complication is in the treatment of input actions. These call for substitutions of the bound object, and when simulating $a(x)$ by $\xrightarrow{\tau}\cdots\xrightarrow{\tau}\xrightarrow{a(x)}\xrightarrow{\tau}\cdots\xrightarrow{\tau}$ it turns out to be important that the substitution is applied immediately after the input action $\xrightarrow{a(x)}$, before further τ transitions.

Definition 9 *A* weak (late) bisimulation *is a symmetric binary relation $\mathcal{R}$ on agents satisfying the following: $P\mathcal{R}Q$ and $P \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

(i) *If $\alpha = a(x)$ then* $\exists Q'' : \quad Q \Longrightarrow\xrightarrow{a(x)} Q'' \wedge$
$\forall u \exists Q' : \quad Q''\{u/x\} \Longrightarrow Q' \ \wedge\ P'\{u/x\}\mathcal{R}Q'$

(ii) *If α is not an input then $\exists Q' : \ Q \overset{\widehat{\alpha}}{\Longrightarrow} Q' \ \wedge P'\mathcal{R}Q'$*

P and Q are weakly (late) bisimilar*, written $P \dot{\approx} Q$, if they are related by a weak bisimulation.*

Clause (i) says how an input transition should be simulated: first do any number of τ, then the input reaching Q'', and then, for each name u received, continue to some Q'. The effect is that the choice of Q'' is independent of u, and the choice of Q' may depend on u. Let us see an example of the significance of this:

$$\begin{array}{lcll} P_1 & = & a(x)\,.\,\mathtt{if}\ x = v\ \mathtt{then}\ P\ + & a(x)\,.\,(\tau + \tau\,.\,P + \tau\,.\,\mathtt{if}\ x = v\ \mathtt{then}\ P) \\ P_2 & = & & a(x)\,.\,(\tau + \tau\,.\,P + \tau\,.\,\mathtt{if}\ x = v\ \mathtt{then}\ P) \\ P_3 & = & & a(x)\,.\,(\tau + \tau\,.\,P) \end{array}$$

First, observe that P_2 is even *strongly* bisimilar to P_3, since

$$\tau + \tau\,.\,P \ \sim\ \tau + \tau\,.\,P + \tau\,.\,\mathtt{if}\ x = v\ \mathtt{then}\ P$$

(any substitution makes the third summand behave as one of the first two). Now consider a transition

$$P_1 \xrightarrow{a(x)} \texttt{if } x = v \texttt{ then } P$$

Can P_2 simulate this? Yes, by the transition

$$P_2 \xrightarrow{a(x)} \tau + \tau \,.\, P + \tau \,.\, \texttt{if } x = v \texttt{ then } P$$

and now applying $\{u/x\}$ by one τ-transition to $(\texttt{if } x = v \texttt{ then } P)\{u/x\}$. Since $P_2 \sim P_3$ we also expect P_3 to simulate it, and this requires the full glory of clause (i), with different Q' for different u. The simulating transition is

$$P_3 \xrightarrow{a(x)} \tau + \tau \,.\, P$$

and now applying $\{u/x\}$, by one τ transition to either P or $\mathbf{0}$, depending on if $u = v$ or not. So here we have $P_1 \dot{\approx} P_2 \dot{\approx} P_3$.

This example also highlights why a simplification of clause (i) to

$$\exists Q' : Q \overset{a(x)}{\Longrightarrow} Q' \;\wedge\; \forall u : \; P'\{u/x\}\mathcal{R}Q'\{u/x\}$$

fails. With such a clause we would have that $P_1 \dot{\approx} P_2$ and $P_2 \dot{\approx} P_3$, yet $P_1 \dot{\not\approx} P_3$. In other words, weak bisimilarity would not be transitive. The reason is that P_3 no longer can simulate $P_1 \xrightarrow{a(x)} \texttt{if } x = v \texttt{ then } P$, since neither $\mathbf{0}$, P, nor $\tau + \tau \,.\, P$ is bisimilar to $\texttt{if } x = v \texttt{ then } P$ for all substitutions of x.

Another attempt at simplification is to replace $\xrightarrow{\alpha}$ by $\overset{\widehat{\alpha}}{\Longrightarrow}$ everywhere, also in the antecedent:

> $P\mathcal{R}Q$ and $P \overset{\widehat{a(x)}}{\Longrightarrow} P'$ where x is fresh implies that $\exists Q' : \; Q \overset{\widehat{a(x)}}{\Longrightarrow} Q'$ and $\forall u : \; P'\{u/x\}\mathcal{R}Q'\{u/x\}$

Then the so defined weak bisimilarity would be transitive. But it is inadequate for another reason. Consider

$$\begin{array}{rcl} P_1 & = & a(x) \,.\, \texttt{if } x \neq v \texttt{ then } (P + \tau \,.\, \texttt{if } x \neq v \texttt{ then } R) \\ P_2 & = & a(x) \,.\, \texttt{if } x \neq v \texttt{ then } (P + \tau \,.\, R) \end{array}$$

Here P_1 and P_2 are even *strong* late bisimilar. But with the attempted simplification they would not be weak late bisimilar since $P_1 \overset{a(x)}{\Longrightarrow} \texttt{if } x \neq v \texttt{ then } R$ cannot be simulated by P_2.

So it seems that clause (i) in Definition 9 is unavoidable. Then $\dot{\approx}$ is an equivalence and every strong bisimulation is also a weak bisimulation, as expected. Of course $\dot{\approx}$ is not a congruence, for two independent reasons. The first is that it is not preserved by input Prefix, similarly to the situation for $\dot{\sim}$. The second is that it is not preserved by $+$, similarly to the situation for observation equivalence in CCS. These two concerns can be addressed independently in the standard ways: as in Section 6 closure under all substitutions is required to preserve input, and as in CCS simulating an initial τ transition by $\overset{\tau}{\Longrightarrow}$ (rather than $\Longrightarrow$) gives a relation preserved by $+$. In conclusion *weak late congruence* is defined as follows.

Definition 10 *P and Q are* weak (late) congruent *if, for all substitutions σ, $P\sigma \xrightarrow{\alpha} P'$ where* $\mathtt{bn}(\alpha)$ *is fresh implies that*

(i) *If $\alpha = a(x)$ then* $\exists Q'' : \quad Q\sigma \Longrightarrow\xrightarrow{a(x)} Q'' \wedge$
$\forall u \exists Q' : \ Q''\{u/x\} \Longrightarrow Q' \ \wedge \ P'\{u/x\} \approx Q'$

(ii) *If α is not an input then* $\exists Q' : \ Q\sigma \overset{\alpha}{\Longrightarrow} Q' \ \wedge P' \dot{\approx} Q'$

and conversely $Q\sigma \xrightarrow{\alpha} Q'$ implies similar transitions from $P\sigma$.

Other strong bisimulation equivalences also have weak counterparts in a similar way. For example weak early bisimilarity is obtained by replacing $Q \xrightarrow{\alpha} Q'$ in Definition 3 by $Q \overset{\widehat{\alpha}}{\Longrightarrow} Q'$; since there are no bound inputs, the complication with clause (i) in Definition 9 above does not arise. Weak barbed congruence is obtained by replacing $Q \xrightarrow{\tau} Q'$ by $Q \Longrightarrow Q'$ and $Q \downarrow a$ by $Q \Longrightarrow\downarrow a$ in Definition 5. The weak barbed congruence coincides with weak early congruence for image-finite agents (this is a mild technical condition which roughly means that the agent can only reach a finite number of different derivatives after $\overset{\alpha}{\Longrightarrow}$). Also weak open bisimulation is obtained by replacing $Q\sigma \xrightarrow{\alpha} Q'$ by $Q\sigma \overset{\widehat{\alpha}}{\Longrightarrow} Q'$ in Definition 7.

8 Algebraic Theory

In this section we consider algebraic axiomatisations of late strong bisimilarity and congruence. This means that we identify a set of axioms for equality between agents, and that together with equational reasoning these imply all true equalities. We get one such set of axioms for bisimilarity and another set for congruence. These results only hold for the finite subcalculus, i.e., we do not include Identifiers or Replication. (In the whole calculus such a result cannot be obtained since bisimilarity, and also congruence, is not recursively enumerable and hence has no decidable axiomatisation.)

8.1 Bisimilarity

Initially we restrict attention to the finite subcalculus without Parallel composition. The axioms for strong late bisimilarity are given in Table 6. We also implicitly use the laws of equational reasoning, i.e., that equality between agents is reflexive, symmetric and transitive. Note that substitutivity (that an agent can replace an equal agent in any expression) is *not* implied, since bisimilarity is not a congruence.

The axioms deserve little comment. STR says that all laws for structural congruence can be used. CONGR1 says that all operators except input Prefix preserve

STR	If $P \equiv Q$ then $P = Q$	
CONGR1	If $P = Q$ then $\overline{a}u\,.\,P = \overline{a}u\,.\,Q$	
	$\tau\,.\,P = \tau\,.\,Q$	
	$P + R = Q + R$	
	$(\boldsymbol{\nu}x)P = (\boldsymbol{\nu}x)Q$	
CONGR2	If $P\{y/x\} = Q\{y/x\}$ for all $y \in \mathtt{fn}(P, Q, x)$ then $a(x)\,.\,P = a(x)\,.\,Q$	
S	$P + P \;=\; P$	
M1	$\mathtt{if}\ x = x\ \mathtt{then}\ P \;=\; P$	
M2	$\mathtt{if}\ x = y\ \mathtt{then}\ P \;=\; \mathbf{0}$	if $x \neq y$
MM1	$\mathtt{if}\ x \neq x\ \mathtt{then}\ P \;=\; \mathbf{0}$	
MM2	$\mathtt{if}\ x \neq y\ \mathtt{then}\ P \;=\; P$	if $x \neq y$
R1	$(\boldsymbol{\nu}x)\alpha\,.\,P \;=\; \alpha\,.\,(\boldsymbol{\nu}x)P$	if $x \notin \alpha$
R2	$(\boldsymbol{\nu}x)\alpha\,.\,P \;=\; \mathbf{0}$	if x is the subject of α
R3	$(\boldsymbol{\nu}x)(P + Q) \;=\; (\boldsymbol{\nu}x)P + (\boldsymbol{\nu}x)Q$	

Table 6: Axioms for strong late bisimilarity.

bisimilarity, and CONGR2 says that to infer bisimilarity of input Prefixes it is sufficient to establish bisimilarity under substitutions for the bound object. Note that CONGR2 mentions only a finite number of such substitutions: names free in the agents under consideration plus one more name represented by x. In view of Proposition 2 this is sufficient since all other names can be obtained through an injective substitution on x. Laws M1–MM2 serve to evaluate `if` constructs, and therefore a clause for Match and Mismatch in CONGR1 is unnecessary. The reader may remember from Section 5.1 that MM2 is not admissible in the structural congruence and may therefore be surprised to see this apparently unsuitable law in Table 6. The explanation is that MM2 is unsuitable for a congruence, since prefiguring both sides of it by an input prefix may invalidate it, and that bisimilarity is not a congruence; prefiguring with an input prefix requires the law CONGR2.

R1–R2 mean that a Restriction can be pushed through a Prefix or disappear; the only exception is when x is the free object of α in which case neither R1 nor R2 applies. Observe that R1 can be regarded as scope extension over Prefix; as was mentioned in Section 2.2 an option is to have this law as part of the structural congruence. Finally R3 means that Restriction distributes over Sum. This law is more powerful than scope extension over sum, since it splits one binder into two.

It is easily seen that all laws are sound, so if $P = Q$ is provable then it must hold that $P \dot{\sim} Q$. We shall now also prove the converse. In the following it is important to remember that α ranges also over bound output Prefixes of kind $\overline{a}\boldsymbol{\nu}x$. Let the *depth* of an agent be the maximal nesting of its Prefixes.

EXP Let $P = \sum_i \alpha_i . P_i$ and $Q = \sum_j \beta_j . Q_j$ where $\mathtt{bn}(\alpha_i) \cap \mathtt{fn}(Q) = \emptyset$ and $\mathtt{bn}(\beta_j) \cap \mathtt{fn}(P) = \emptyset$ for all i, j, and none of α_i or β_j is a bound output Prefix. Then

$$P|Q \quad = \quad \sum_i \alpha_i . (P_i|Q) \; + \; \sum_j \beta_j . (P|Q_j) \; + \sum_{\alpha_i \, comp \beta_j} \tau . R_{ij}$$

where the relation $\alpha_i comp \beta_j$ and R_{ij} are defined as follows: either $\alpha_i = a(x)$ and $\beta_j = \overline{a}u$ in which case $R_{ij} = P_i\{u/x\}|Q_j$, or conversely $\alpha_i = \overline{a}u$ and $\beta_j = a(x)$ in which case $R_{ij} = P_i|Q_j\{u/x\}$.

Table 7: Expansion law for strong bisimilarity.

Proposition 7 *Using the axioms in Table 6 every agent P is provably equal to a* head normal form *(hnf) of kind $\sum_i \alpha_i . P_i$ of no greater depth.*

The proof is by induction over the structure of the agent and all cases are easy. If P is a Match or Mismatch then M1–MM2 applies; if it is a Restriction then R3 is used to distribute it onto the summands and R1–R2 to push it through the Prefixes or form part of a bound output Prefix.

Proposition 8 *If $P \dot{\sim} Q$ then $P = Q$ is provable from the axioms in Table 6.*

The proof is by induction on the depths of P and Q. By Proposition 7 we can assume P and Q are head normal forms. The base case $P = Q = \mathbf{0}$ is trivial. For the inductive step we prove that for each summand in P there is a provably equal summand in Q and vice versa. For example, take a summand $a(x) . P'$ in P. Assume by alpha-conversion that all top-level input actions have the same bound object x. Then from $P \dot{\sim} Q$ and $P \xrightarrow{a(x)} P'$ we get $Q \xrightarrow{a(x)} Q'$ such that $P'\{u/x\} \dot{\sim} Q'\{u/x\}$ for all u. By induction they are also provably equal for all u. So $a(x) . Q'$ is a summand of Q and from CONGR2 we get that it is provably equal to $a(x) . P'$. The other cases are similar and simpler. So, each summand of P is provably equivalent to a summand of Q and therefore, by the law S (and STR), P is provably equivalent to Q.

We now turn to the Parallel operator. The idea behind the axiomatisation is to introduce an expansion law through which the composition of two head normal forms is provably equal to a head normal form. With this law Proposition 7 will continue to hold for the calculus with Parallel, and therefore the proof of Proposition 8 needs not change. The expansion law is given in Table 7.

The proof of Proposition 7 is now extended with an inductive step for $P = Q|R$ where Q and R are hnfs. First use scope extension (Proposition 1) to pull any unguarded Restrictions in $Q|R$ to top level, obtaining an agent of kind $(\boldsymbol{\nu}\tilde{x})(Q'|R')$ where Q' and R' are hnfs with no bound output Prefixes. Then apply EXP to $Q'|R'$ gaining an agent of type $(\boldsymbol{\nu}\tilde{x})P'$ where P' is a hnf, and finally use scope extension to push back the Restrictions inwards, gaining a hnf.

8.2 Congruence

Following Definition 2 we immediately obtain an axiomatisation of $\sim$ by adding that definition as a law. Interestingly, there is an alternative axiomatisation that does not involve quantification over substitutions and does not refer back to bisimilarity. Again we begin with the subcalculus without Parallel.

It is informative to begin by examining the axioms in Table 6 to see what needs to change. The rule STR is still valid and useful, and the rules CONGR1–CONGR2 can be replaced by a simpler rule saying that $\sim$ is a congruence. The problematic laws are M2 and MM2 which are unsound for congruence. For example, even though $\texttt{if}\ x = y\ \texttt{then}\ P$ and $\mathbf{0}$ are bisimilar when $x \neq y$ they are not necessarily congruent since a substitution $\{x/y\}$ makes them non-bisimilar. In an axiom system for $\sim$ we cannot rely on axioms which eliminate unguarded Match and Mismatch operators.

A concise presentation of the axioms depends on the notion of a *Generalised* match operator written $\texttt{if}\ M\ \texttt{then}\ P$, where M is a conjunction of conditions of type $x = y$ and $x \neq y$. The Generalised match is simply defined to be a nested sequence of Matches and Mismatches, one for each condition:

$$\texttt{if}\ m_1 \wedge \cdots \wedge m_k\ \texttt{then}\ P \ = \ \texttt{if}\ m_1\ \texttt{then}\ (\cdots(\texttt{if}\ m_k\ \texttt{then}\ P)\cdots)$$

where each m_i is of type $x = y$ or $x \neq y$. We say that M logically implies N if all substitutions that make the conditions in M true also make the conditions in N true, and that they are logically equivalent, written $M \Leftrightarrow N$, if they imply each other. This allows us to use the compact law GM1:

$$\texttt{if}\ M\ \texttt{then}\ P = \texttt{if}\ N\ \texttt{then}\ P \qquad \text{if } M \Leftrightarrow N$$

in place of a set of laws for nestings of Matches and Mismatches. The axioms for strong late congruence are given in Table 8.

We comment briefly on the new axioms. GM2 is a form of case analysis, allowing us to split an agent to a Sum of two mutually exclusive conditions. Writing out the definition of `if ...then ...else` this law is

$$\texttt{if}\ x = y\ \texttt{then}\ P \ + \ \texttt{if}\ x \neq y\ \texttt{then}\ P \quad = \quad P$$

GM3 is a kind of distributive law for Generalised match over Sum. GM4 says that a test, once passed, can be done again after an action α, since that action cannot invalidate the outcome of the test. Here the side condition on $\texttt{bn}(\alpha)$ is important. GM5 embodies the essence of a match: if x and y have been deemed equal then one can substitute the other (the substitution is defined to not affect $\texttt{bn}(\alpha)$). Since $x = y \Leftrightarrow y = x$ we get from GM1 that it does not matter whether the substitution is $\{y/x\}$ or $\{x/y\}$. And combined with GM3 and GM4 we get all instances of a stronger law $\texttt{if}\ x = y\ \texttt{then}\ \alpha\,.\,P \ = \ \texttt{if}\ x = y\ \texttt{then}\ (\alpha\,.\,P)\{x/y\}$ where the substitution affects the whole Prefix form. Finally GM6 is the essence

STR	If $P \equiv Q$ then $P = Q$	
CONGR	"=" is preserved by all operators	
S	$P + P \; = \; P$	
MM1	$\texttt{if}\ x \neq x\ \texttt{then}\ P \; = \; \mathbf{0}$	
GM1	$\texttt{if}\ M\ \texttt{then}\ P \; = \; \texttt{if}\ N\ \texttt{then}\ P$	if $M \Leftrightarrow N$
GM2	$\texttt{if}\ x = y\ \texttt{then}\ P\ \texttt{else}\ P \; = \; P$	
GM3	$\texttt{if}\ M\ \texttt{then}\ (P_1 + P_2) \; = \; \texttt{if}\ M\ \texttt{then}\ P_1 + \texttt{if}\ M\ \texttt{then}\ P_2$	
GM4	$\texttt{if}\ M\ \texttt{then}\ \alpha\,.\,P \; = \; \texttt{if}\ M\ \texttt{then}\ (\alpha\,.\,\texttt{if}\ M\ \texttt{then}\ P)$	if $\texttt{bn}(\alpha) \notin M$
GM5	$\texttt{if}\ x = y\ \texttt{then}\ \alpha\,.\,P \; = \; \texttt{if}\ x = y\ \texttt{then}\ (\alpha\{x/y\})\,.\,P$	
GM6	$(\boldsymbol{\nu} x)\texttt{if}\ x = y\ \texttt{then}\ P \; = \; \mathbf{0}$	if $x \neq y$
R1	$(\boldsymbol{\nu} x)\alpha\,.\,P \; = \; \alpha\,.\,(\boldsymbol{\nu} x)P$	if $x \notin \alpha$
R2	$(\boldsymbol{\nu} x)\alpha\,.\,P \; = \; \mathbf{0}$	if x is the subject of α
R3	$(\boldsymbol{\nu} x)(P + Q) \; = \; (\boldsymbol{\nu} x)P + (\boldsymbol{\nu} x)Q$	

Table 8: Axioms for strong late congruence.

of Restriction: a restricted name can never be made equal to another name so the Match will always come out as false.

As can be seen the law MM1 is inherited from the axioms for bisimilarity. Of course also M1 is valid but it is not necessary as an axiom since it can be derived from GM2 and MM1. Similarly, a counterpart for Mismatch to GM6, namely

$$\text{GM6}^* \qquad (\boldsymbol{\nu} x)\texttt{if}\ x \neq y\ \texttt{then}\ P \; = \; (\boldsymbol{\nu} x)P \qquad \text{if } x \neq y$$

is derivable from GM2 and GM6. Thus a Restriction can always be pushed through a generalised match. Also, note that $\texttt{if}\ x = y\ \texttt{then}\ \mathbf{0} = \mathbf{0}$ is derivable from MM1 and GM2.

For an example of a derivation consider $(\boldsymbol{\nu} x)\overline{a}x\,.\,\texttt{if}\ x = y\ \texttt{then}\ P$ which, as argued in Section 7.3, is congruent to $(\boldsymbol{\nu} x)\overline{a}x\,.\,\mathbf{0}$. The proof from axioms is:

$$\begin{array}{lr}
(\boldsymbol{\nu} x)\overline{a}x\,.\,\texttt{if}\ x = y\ \texttt{then}\ P \quad = & [\text{GM6}*] \\
(\boldsymbol{\nu} x)\texttt{if}\ x \neq y\ \texttt{then}\ \overline{a}x\,.\,\texttt{if}\ x = y\ \texttt{then}\ P \quad = & [\text{GM4}] \\
(\boldsymbol{\nu} x)\texttt{if}\ x \neq y\ \texttt{then}\ \overline{a}x\,.\,\texttt{if}\ x \neq y\ \texttt{then if}\ x = y\ \texttt{then}\ P \quad = & [\text{GM6}*] \\
(\boldsymbol{\nu} x)\overline{a}x\,.\,\texttt{if}\ x \neq y\ \texttt{then if}\ x = y\ \texttt{then}\ P \quad = & \\
(\boldsymbol{\nu} x)\overline{a}x\,.\,\texttt{if}\ x \neq y \wedge x = y\ \texttt{then}\ P \quad = & [\text{GM1}] \\
(\boldsymbol{\nu} x)\overline{a}x\,.\,\texttt{if}\ x \neq x\ \texttt{then}\ P \quad = & [\text{MM1}] \\
(\boldsymbol{\nu} x)\overline{a}x\,.\,\mathbf{0} &
\end{array}$$

It is easy to establish that all laws are sound for congruence. The proof of completeness uses another kind of head normal form and is more involved than the corresponding proof for bisimilarity, and we shall here only sketch it. It relies on the notion of *complete* conjunctions. Formally, a conjunction M is complete on a set of names V if M implies either $x = y$ or $x \neq y$ for all names x, y in V. In other words, M expresses unambiguously which names are the same and which are not. We say that M *agrees* with a substitution σ if M implies $x = y$ if and only if $\sigma(x) = \sigma(y)$. The main use of complete conjunctions is that if M is complete on $\mathtt{fn}(P, Q)$ and agrees with σ, then $(\mathtt{if}\ M\ \mathtt{then}\ P) \sim (\mathtt{if}\ M\ \mathtt{then}\ Q)$ holds if and only if $P\sigma \dot{\sim} Q\sigma$. In this way the substitutions arising from the transitional semantics can be internalised and represented by Generalised matches.

Definition 11 *P is in* head normal form on a finite set of names V (V-hnf) *if*

$$P \quad = \quad \sum_i \mathtt{if}\ M_i\ \mathtt{then}\ \alpha_i\,.\,P_i$$

where for all i, $\mathtt{bn}(\alpha_i) \notin V$, and each M_i is complete on V.

So a V-hnf is a Sum of Generalised matches where each conjunction is complete on V.

Proposition 9 *For any V, any agent P is provably equivalent to a V-hnf of no greater depth.*

The proof is by induction over the structure of P. If $P = \alpha\,.\,P'$ then use GM2 repeatedly to generate a Sum $\sum_i \mathtt{if}\ M_i\ \mathtt{then}\ \alpha\,.\,P'$ where the M_i are complete. If $P = (\boldsymbol{\nu} x)P'$ then by induction P' is a V-hnf $\sum_i \mathtt{if}\ M_i\ \mathtt{then}\ \alpha_i\,.\,P_i$. Use R3 to distribute $(\boldsymbol{\nu} x)$ onto all summands, GM6 and GM6* and STR to push the Restriction through all M_i and finally R1–R2 to push it through all α_i (unless x is the free object of α_i, in which case $(\boldsymbol{\nu} x)$ forms part of a bound output Prefix). The other cases are simple from the distributive laws.

Proposition 10 *If $P \sim Q$ then $P = Q$ is provable from the axioms in Table 8.*

The proof is by induction on the depths of P and Q and by the previous proposition we can assume that P and Q are $\mathtt{fn}(P, Q)$-hnfs. The idea of the inductive step is as before to show that for each summand in P there is a provably equal summand in Q. We indicate the main lines for free output summands—the case for bound actions is a little bit more involved though the idea is the same. So let $\mathtt{if}\ M\ \mathtt{then}\ \alpha\,.\,P'$ be a summand in P with α a free output action. Choose a substitution σ that agrees with M. Then $P\sigma \xrightarrow{\alpha\sigma} P'\sigma$. From $P \sim Q$ we get that $Q\sigma$ must simulate this transition. Let $\mathtt{if}\ N\ \mathtt{then}\ \beta\,.\,Q'$ be the summand of Q that generates the simulating transition, which implies that $P'\sigma \dot{\sim} Q'\sigma$. Then N also agrees with σ, and since both M and N are complete and agree with σ we get $M \Leftrightarrow N$. Therefore by GM1 we can replace N by M in the summand of Q. And by GM5 we can replace β by α, because they can only differ in names identified by σ, meaning that M implies the equality of those names. There thus remains

EXP2 Let $P = \sum_i \texttt{if } M_i \texttt{ then } \alpha_i \,.\, P_i$ and $Q = \sum_j \texttt{if } N_j \texttt{ then } \beta_j \,.\, Q_j$ where $\texttt{bn}(\alpha_i) \cap \texttt{fn}(Q) = \emptyset$ and $\texttt{bn}(\beta_j) \cap \texttt{fn}(P) = \emptyset$ for all i, j, and none of α_i or β_j is a bound output Prefix. Then

$$\begin{aligned} P|Q \quad = \quad & \sum_i \texttt{if } M_i \texttt{ then } \alpha_i \,.\, (P_i|Q) \; + \; \sum_j \texttt{if } N_j \texttt{ then } \beta_j \,.\, (P|Q_j) \\ & + \sum_{\alpha_i \, opp \beta_j} \texttt{if } M_i \wedge N_i \wedge a_i = b_j \texttt{ then } \tau \,.\, R_{ij} \end{aligned}$$

where the relation $\alpha_i opp \beta_j$, a_i, b_j and R_{ij} are defined as follows: either $\alpha_i = a_i(x)$ and $\beta_j = \overline{b}_j u$ in which case $R_{ij} = P_i\{u/x\}|Q_j$, or conversely $\alpha_i = \overline{a}_i u$ and $\beta_j = b_j(x)$ in which case $R_{ij} = P_i|Q_j\{u/x\}$.

Table 9: Expansion law for strong congruence.

to prove $\texttt{if } M \texttt{ then } \alpha \,.\, P' = \texttt{if } M \texttt{ then } \alpha \,.\, Q'$. We know that $P'\sigma \dot{\sim} Q'\sigma$. Since M is complete it holds that this implies $\texttt{if } M \texttt{ then } P' \sim \texttt{if } M \texttt{ then } Q'$, whence by induction these are provably equivalent. We now apply GM4 (for the first time in the proof!) to write $\texttt{if } M \texttt{ then } \alpha \,.\, P'$ as $\texttt{if } M \texttt{ then } \alpha \,.\, \texttt{if } M \texttt{ then } P'$. That means it is provably equivalent to $\texttt{if } M \texttt{ then } \alpha \,.\, \texttt{if } M \texttt{ then } Q'$, and again by GM4 to $\texttt{if } M \texttt{ then } \alpha \,.\, Q'$.

Finally we consider also the Parallel operator. By analogy with the case for bisimilarity it is enough to include an axiom which implies that the Parallel composition of two V-hnfs is a V-hnf. Unfortunately the axiom EXP in Table 7 is not sound for congruence, since a substitution may identify two names and thereby enable a communication. The amended expansion law for congruence is given in Table 9. Note the essential use of Matches to determine if a communication is possible. With this law propositions 9 and 10 hold. Incidentally, the law is sound and complete also for bisimilarity, i.e., it can be used in place of EXP for the purpose of Proposition 8.

9 Variants of the Theory

We here briefly mention the axiomatisations of the equivalences treated in Section 7.

9.1 Early Bisimilarity and Congruence

Early bisimilarity equates more agents than late and therefore we seek more axioms. It turns out that we need to add only one axiom to Table 6 to obtain a complete system. This characteristic axiom for early is:

EARLY
$$\begin{aligned} & a(x) \,.\, P + a(x) \,.\, Q \;= \\ & a(x) \,.\, P + a(x) \,.\, Q + a(x) \,.\, \texttt{if } x = y \texttt{ then } P \texttt{ else } Q \end{aligned}$$

The example from Section 7.1:

$$P_1 = a(x)\,.\,P + a(x)\,.\,\mathbf{0}, \quad P_2 = a(x)\,.\,P + a(x)\,.\,\mathbf{0} + a(x)\,.\,\texttt{if}\ x = u\ \texttt{then}\ P$$

is an instance of EARLY with $Q = \mathbf{0}$. So clearly this axiom is unsound for late bisimilarity. The soundness for early bisimilarity is straightforward from the definition.

If EARLY is adjoined to Table 6 we get a complete axiomatisation of early bisimilarity. The proof of this is different from the corresponding proof of Proposition 8 since it is no longer the case that each summand of P has a bisimilar summand of Q. Therefore, the hnfs corresponding to P and Q must first be saturated by applying EARLY, from left to right, as much as possible to generate new summands. Although the axiom can be applied ad infinitum it turns out that there are only a finite number of distinct new summands that need be generated this way. If instead EARLY is adjoined to Table 8 then we similarly get a complete axiomatisation of early congruence. The proof is a combination of the above idea and the proof of Proposition 10.

Interestingly, there is an alternative to EARLY as a characteristic law for early congruence, namely

$$\text{EARLY2} \quad \text{if } \textstyle\sum_i \tau\,.\,P_i \;=\; \sum_j \tau\,.\,Q_j \quad \text{then} \quad \sum_i a(x)\,.\,P_i \;=\; \sum_j a(x)\,.\,Q_j$$

We can see that EARLY and EARLY2 are equipotent as follows. Consider

$$\tau\,.\,P + \tau\,.\,Q \;=\; \tau\,.\,P + \tau\,.\,Q + \tau\,.\,\texttt{if}\ x = y\ \texttt{then}\ P\ \texttt{else}\ Q$$

This equation is certainly true for $\sim$, since any substitution will make the third summand behave as either $\tau\,.\,P$ or $\tau\,.\,Q$. Hence all instances are derivable from the laws in Table 8. Applying EARLY2 to the equation we get EARLY. So any equation that can be derived with EARLY can also be derived with EARLY2. The converse follows by the fact that EARLY yields a complete axiomatisation for $\sim_{\mathrm{E}}$ and EARLY2 is sound.

Similarly, a version of EARLY2 for early bisimilarity is

EARLY2$'$

$$\text{if } \forall u : \textstyle\sum_i \tau\,.\,P_i\{u/x\} \;=\; \sum_j \tau\,.\,Q_j\{u/x\} \quad \text{then} \quad \sum_i a(x)\,.\,P_i \;=\; \sum_j a(x)\,.\,Q_j$$

This may appear more complicated than EARLY but it also replaces CONGR2 and an interesting point is that it does not introduce a Mismatch operator. In the subcalculus without Mismatch we can therefore axiomatise late and early bisimilarity (just drop MM1 and MM2 from Table 6). In contrast, for late and early congruence the Mismatch operator plays a significant role in forming complete Generalised matches, so axiomatisations without Mismatch must take a different route.

9.2 Open Bisimilarity

The remaining interesting equivalence in Figure 2, open bisimilarity (which also is open congruence), is special in that it is only defined for the subcalculus without Mismatch. The way to axiomatise it goes through the relations $\dot{\sim}^D_{\mathrm{O}}$ (open bisimilarity under distinction D) so the axiomatisation schematically gives rules for all D. The axioms in Table 8 need some modifications. All axioms mentioning Mismatch are dropped; this means that MM1 and GM2 are omitted and in GM1,3,4 M ranges over Generalised matches with no negative conjuncts. A new axiom

$$\text{GM7} \quad P + \texttt{if } x = y \texttt{ then } P \;=\; P$$

is added (in the original Table 8 this is derivable through S and GM2). Finally for the interplay between different distinctions we add the following laws, where $=_D$ means provable equality under distinction D,

$$\begin{array}{lll} \text{D1} & \texttt{if } x = y \texttt{ then } P \;=_D\; \mathbf{0} & \text{if } xDy \\ \text{D2} & P \;=_D\; Q \;\text{ implies }\; P \;=_{D'}\; Q & \text{if } D \subseteq D' \\ \text{D3} & P \;=_D\; Q \;\text{ implies }\; (\boldsymbol{\nu}x)P \;=_{D-x}\; (\boldsymbol{\nu}x)Q & \end{array}$$

Here $D - x$ means the distinction where all pairs containing x are removed from D. Because of these laws GM6 is derivable and needs not be taken as an axiom.

With these changes we obtain a complete axiomatisation of $\dot{\sim}^D_{\mathrm{O}}$ for all D. The proof is substantially more complicated than for Proposition 10. It uses a notion of head normal form with summands of type $\texttt{if } M \texttt{ then } \alpha\,.\,P$, and with the extra requirement that there are no unnecessary summands (a summand P is unnecessary if there is another summand Q and $Q \;\dot{\sim}_{\mathrm{O}}\; Q + P$). The proof that all agents are provably equal to such head normal forms uses, among other things, the new law GM7 in order to remove unnecessary summands. The idea of the completeness proof is that given two bisimilar head normal forms, each summand in one of them is provably equal to a summand in the other.

9.3 Weak Congruence

Finally, for the weak late and early congruences (axiomatisations of weak open congruence have not yet been investigated) it is enough to add the three standard so called τ-laws:

$$\begin{array}{lrcl} \text{T1} & \alpha\,.\,\tau\,.\,P & = & \alpha\,.\,P \\ \text{T2} & P + \tau\,.\,P & = & \tau\,.\,P \\ \text{T3} & \alpha\,.\,(P + \tau\,.\,Q) & = & \alpha\,.\,(P + \tau\,.\,Q) + \alpha\,.\,Q \end{array}$$

The proof follows the same lines as the corresponding proof in CCS: the head normal forms must be saturated in the sense that if $P \overset{\alpha}{\Longrightarrow} P'$ then P has a summand $\alpha\,.\,P'$. To see an example derivation which demonstrates the subtlety

	strong bisimilarity	**strong congruence**	**weak congruence**
late	Table 6	Table 8	Table 8 *and* T1, T2, T3
early	Table 6 *and* EARLY *or* EARLY2′	Table 8 *and* EARLY *or* EARLY2	Table 8 *and* EARLY *or* EARLY2 *and* T1, T2, T3
open	Table 8 *except laws for Mismatch* *and* GM7, D1, D2, D3		*not investigated*

Table 10: Axiomatisations of the bisimilarities and congruences.

of the interplay between τ-actions and the laws for Match and Mismatch, and highlights a technique used in the completeness proof, suppose we want to prove

$$a(x).(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) \;=\; a(x)\,.\,P$$

Clearly this holds, since $(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P)\{u/x\}$ is weakly bisimilar to $P\{u/x\}$ for all u. The complication is that $(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P)$ is not weakly congruent to P, since the former has an initial τ-transition for the substitution $\{y/x\}$. The proof technique is therefore to first infer

$$\tau\,.\,(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) \;=\; \tau\,.\,P$$

and then use CONGR, prefixing both sides by $a(x)$, and T1 to remove the τ. The inference of $\tau\,.\,(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) \;=\; \tau\,.\,P$ is as follows:

$$\begin{array}{ll}
\tau\,.\,(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) \;= & (\textsc{gm}2)\\
\texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) & \\
\qquad +\ \texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,(P + \texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P) \;= & (\textsc{gm}1,3,4)\\
\texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,(P + \tau\,.\,P) \;+\; \texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,P \;= & (\textsc{t}2)\\
\texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,\tau\,.\,P \;+\; \texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,P \;= & (\textsc{t}1)\\
\texttt{if}\ x = y\ \texttt{then}\ \tau\,.\,P \;+\; \texttt{if}\ x \neq y\ \texttt{then}\ \tau\,.\,P \;= & (\textsc{gm}2)\\
\tau\,.\,P &
\end{array}$$

The different axiomatisations are summarised in Table 10.

10 Sources

Early Developments

Computational models where mobility plays a predominant role have been present at least since the mid 1970's in the so called actor systems by Carl Hewitt and Gul Agha [18, 2]. Technically the π-calculus has its roots in process algebras like Robin Milner's CCS, originally developed in the late 1970's [22, 23]. The first efforts to extend CCS with the ability to pass communication channels were by Egidio Astesiano and Elena Zucca (1984) and by Uffe Engberg and Mogens Nielsen (1986) [4, 12]. These calculi turned out to be quite complex. The π-calculus in its present form was developed in the late 1980's by Milner, David Walker and myself; it was first presented at seminars in 1987, the first comprehensive technical reports appeared in 1989 and the journal publication in 1992 [30]. It builds on the article by Engberg and Nielsen, and simplifies it by using a single syntactical class of names to represent values, variables and channels. There it is shown how to encode data types and a version of the lambda calculus; it also explores the operational semantics (late semantics without use of structural congruence), defines late and early bisimilarity and congruence and gives the axiomatisation of late bisimilarity. The calculus is very similar to the one presented here, though there is no Mismatch operator and some differences in notation. For example Restriction $(\boldsymbol{\nu}x)$ is written (x), and $\texttt{if}\ x = y\ \texttt{then}\ P$ is written more compactly $[x = y]P$, a notation that many papers on the π-calculus follow.

Introductions and Overviews

Today the π-calculus and related theories make up a large and diverse field with hundreds of published papers. Perhaps the main sign of vitality is not the theoretical developments, but rather the numerous calculi, computational models and languages that borrow semantics or central concepts from π and are aimed at more focussed areas of application, such as PICT, Facile, Join, Ambients, Spi, POOL,... the list can be made very long [45, 46, 8, 15, 11, 1, 57]. A brief overview from 1998 by Uwe Nestmann and Björn Victor indicate the main issues and is accompanied by a well maintained and exhaustive searchable on-line bibliography founded in 1995 [35]. Nestmann also currently maintains the web page *Calculi for Mobile processes* at

`http://www.cs.auc.dk/mobility/`

with links to introductory papers, active researchers, departments, projects and other resources. In view of these efforts it seems excessive to here attempt a substantial overview or list of references. If a single introductory article for the layman should be mentioned it must be Milner's 1991 Turing Award Lecture [26].

A newcomer to the field may also appreciate my article from 1993 using graphs instead of formulas, with an emphasis on how various basic computational structures are represented through name-passing [39]. Otherwise a standard text is Milner's tutorial on the polyadic π-calculus (1991), and also his recent book (1999) is aimed at a non-specialist audience [27, 29]. A short introduction to a simple variant of the π-calculus in comparison to the lambda-calculus is given by Benjamin Pierce (1996) [43].

A reader seeking more detailed information concerning some specific topic is well advised to first consult Kohei Honda's annotated on-line bibliography (reachable from the URL above); it is not exhaustive and not updated as frequently as the one by Nestmann and Victor, but focuses on a few central issues and explains the impact of each paper.

Below I briefly mention the main sources for the aspects of the calculus elaborated in this introduction. The chronology refers to the first written account of a piece of work known to me; the corresponding conference paper or journal article is often published a couple of years later.

Variants of the Calculus

The Mismatch operator first appeared in 1990 and was used by Davide Sangiorgi and me in 1993 [38, 41]. The role of the Sum operator has been studied by Catuscia Palamidessi (1997) where she shows that it is necessary for the representation of certain distributed algorithms, and by Nestmann (1997) and Nestmann and Pierce (1996) where encodings of Sum for special cases are analysed (the encoding in Section 3.5 comes from the latter article) [37, 33, 34]. The polyadic calculus with the sort system given in Section 3.3 was introduced by Milner in his 1991 tutorial. The ideas for encoding the polyadic calculus into the monadic have been known since the first papers on the π-calculus but were not studied in detail until 1996 by Nobuko Yoshida and 1998 by Paola Quaglia and Walker [58, 47]. Type inference algorithms were first presented independently by Simon Gay and by Vasco Vasconcelos and Honda in 1993 [16, 55]. More elaborate systems for types and sorts have been presented by many others. Prominent issues are more refined types, for example distinguishing different uses of a channel for input and output, subtyping, polymorphism and higher-order types. A survey of such systems remains to be written although Honda's bibliography mentioned above explains many of the contributions. The slides of a good introductory tutorial by Pierce (1998) are available on-line from the author's home page (reachable from the URL for the mobility home page above) [44].

Replication in place of recursion was first suggested by Milner in a paper on encodings of the lambda-calculus 1990, and although the relationship between lambda and π has subsequently been treated by many authors that paper remains the principal reference [25]. The issue is to some extent connected to higher-order process calculi. Bent Thomsen, Gérard Boudol and Flemming Nielson were the first to study such calculi in detail (independently 1989, these calculi differ significantly from the π-calculus) [53, 9, 36, 54]. The connection with the lambda-calculus was later clarified by Sangiorgi, who also studied the encoding of the higher-order π-calculus into the standard calculus [49].

The significance of the asynchronous calculus was discovered independently by Kohei Honda and Mario Tokoro 1991 and by Boudol 1992 [19, 10]. A particularly interesting variant is the subcalculus without free output actions, which has a simpler theory and retains a surprising expressive power, as demonstrated by Sangiorgi and Michele Boreale (1996) [50, 6]. Sangiorgi has recently written a tutorial on the subject [52].

Variants of the Semantics

The first presentation of a structural congruence and reduction semantics is by Milner (1990) [25], inspired by the chemical abstract machines of Gérard Berry and Boudol from 1989 [5] and Milner also gave the semantics for abstractions and concretions (with a slightly different notation; Milner prefers $P \succ c\,.\,E$ to $P \xrightarrow{c} E$) in his 1991 tutorial. The early semantics was first presented by Milner, Walker and myself in 1991 [31].

Symbolic transitions and their use in decision procedures for bisimulation equivalences were introduced by Matthew Hennessy and Huimin Lin in 1992 for a process algebra where values (but not channel names) are transmitted between agents [17]. Another chapter in this handbook by Anna Ingólfsdóttir and Lin treats this issue in full. The semantics was subsequently adapted by Lin (1994) to the π-calculus [20, 21]. Similar ideas were developed in parallel by Boreale and Rocco de Nicola [7]. Symbolic transitions are also used by Faron Moller and Björn Victor in an automatic verification tool (1994) [56].

Variants of Bisimilarity

Early bisimilarity was mentioned in the first paper on the calculus and was studied in more depth by Milner, Walker and myself (1991) along with modal logics which clarify the relationship between late and early bisimulation [31]. The results on barbed congruence are primarily due to Milner and Sangiorgi (1992) and are developed in Sangiorgi's PhD Thesis [32, 48]. A version of barbed congruence for the asynchronous subcalculus is defined and axiomatised by Roberto Amadio, Ilaria Castellani and Sangiorgi (1996) [3]; the example at the very end of Section 7.2 is from that paper.

Open bisimulation (both strong and weak) and its axiomatisation (only strong) is due to Sangiorgi 1993 who also uses a symbolic semantics [51]. The weak late and early bisimulation equivalences and congruences were first formulated by Milner (1990) [24].

Algebraic Theory

Strong late bisimilarity was axiomatised in the first paper on the calculus. Sangiorgi and I axiomatised strong late congruence and early bisimilarity and congruence (through the law EARLY) in 1993; the axiomatisations given here mainly follow that work [41]. Axiomatisations of the weak congruences (late and early) have been treated by Lin in a style slightly different from what has been presented here [21]. The law EARLY2 also comes from this line of work. Inferences are there of kind $C \triangleright P = Q$ meaning "under condition C it holds that $P = Q$", which we would express as `if` C `then` $P =$ `if` C `then` Q, making the condition part of the agents. The fact that this yields an isomorphic proof system was only recently established [40]. An alternative presentation of weak late congruence and its axiomatisation is by Gianluigi Ferrari, Ugo Montanari and Paola Quaglia (1995) [14].

Unifying Efforts

In this growing and diversifying field it is also appropriate to mention a few efforts at unification. The action calculi by Milner (emanating from the work on action structures 1992) separate the concerns of parametric dependency and execution control [28]. The tiles structures by Ferrari and Montanari (emanating from work with Fabio Gaducci 1995) generalise the concepts of context and context composition [13]. The Fusion Calculus by Victor and myself (begun in 1997) identifies a single binding operator which can be used to derive both input and Restriction [42]. Although each has made some progress it is clear that much work remains to be done.

Acknowledgements

I thank Huimin Lin, Davide Sangiorgi, Thomas Noll, Gunnar Övergaard, and the anonymous referees for many helpful comments.

References

[1] M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 143:1–70, 1999. An extended abstract appeared in the *Proceedings of the Fourth ACM Conference on Computer and Communications Security (Zürich, April 1997)*. An extended version of this paper appears as Research Report 149, Digital Equipment Corporation Systems Research Center, January 1998, and, in preliminary form, as Technical Report 414, University of Cambridge Computer Laboratory, January 1997.

[2] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, 1986.

[3] R. M. Amadio, I. Castellani, and D. Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195(2):291–324, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 147–162.

[4] E. Astesiano and E. Zucca. Parametric channels via label expressions in CCS. *Theoretical Computer Science*, 33:45–64, 1984.

[5] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[6] M. Boreale. On the expressiveness of internal mobility in name-passing calculi. *Theoretical Computer Science*, 195(2):205–226, 1998. An extended abstract appeared in *Proceedings of CONCUR '96*, LNCS 1119: 163–178.

[7] M. Boreale and R. De Nicola. A symbolic semantics for the π-calculus. *Journal of Information and Computation*, 126(1):34–52, 1996. Available as Report SI 94 RR 04, Università "La Sapienza" di Roma; an extended abstract appeared in *Proceedings of CONCUR '94*, pages 299–314, LNCS 836.

[8] R. Borgia, P. Degano, C. Priami, L. Leth, and B. Thomsen. Understanding mobile agents via a non interleaving semantics for Facile. In R. Cousot and D. A. Schmidt, editors, *Proceedings of SAS '96*, volume 1145 of *LNCS*, pages 98–112. Springer, 1996. Extended version as Technical Report ECRC-96-4, 1996.

[9] G. Boudol. Towards a lambda-calculus for concurrent and communicating systems. In J. Díaz and F. Orejas, editors, *Proceedings of TAPSOFT '89, Volume 1*, volume 351 of *LNCS*, pages 149–161. Springer, 1989.

[10] G. Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sophia-Antipolis, May 1992.

[11] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of FoSSaCS '98*, volume 1378 of *LNCS*, pages 140–155. Springer, 1998.

[12] U. Engberg and M. Nielsen. A calculus of communicating systems with label-passing. Technical Report DAIMI PB-208, Comp. Sc. Department, Univ. of Aarhus, Denmark, 1986.

[13] G. Ferrari and U. Montanari. A tile-based coordination view of asynchronous π-calculus. In I. Pri ́vara and P. Ružička, editors, *Proceedings of MFCS '97*, volume 1295 of *LNCS*. Springer, Aug. 1994.

[14] G. Ferrari, U. Montanari, and P. Quaglia. The weak late π-calculus semantics as observation equivalence. In I. Lee and S. A. Smolka, editors, *Proceedings of CONCUR '95*, volume 962 of *LNCS*, pages 57–71. Springer, 1995.

[15] C. Fournet and G. Gonthier. The reflexive chemical abstract machine and the join-calculus. In J. G. Steele, editor, *Proceedings of POPL '96*, pages 372–385. ACM, Jan. 1996.

[16] S. J. Gay. A sort inference algorithm for the polyadic π-calculus. In *Proceedings of POPL '93*. ACM, January 1993.

[17] M. Hennessy and H. Lin. Symbolic bisimulations. *Theoretical Computer Science*, 138(2):353–389, 1995. Earlier version as Technical Report 1/92, School of Cognitive and Computing Sciences, University of Sussex, UK.

[18] C. Hewitt. Viewing control structures as patterns of passing messages. *Journal of Artificial Intelligence*, 8:323–364, 1977.

[19] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In P. America, editor, *Proceedings of ECOOP '91*, volume 512 of *LNCS*, pages 133–147. Springer, July 1991.

[20] H. Lin. Symbolic bisimulation and proof systems for the π-calculus. Technical Report 7/94, School of Cognitive and Computing Sciences, University of Sussex, UK, 1994.

[21] H. Lin. Complete inference systems for weak bisimulation equivalences in the π-calculus. In P. D. Mosses, M. Nielsen, and M. I. Schwarzbach, editors, *Proceedings of TAPSOFT '95*, volume 915 of *LNCS*, pages 187–201. Springer, 1995. Presented in the CAAP-section. Available as Technical Report ISCAS-LCS-94-11, Institute of Software, Chinese Academy of Sciences, 1994.

[22] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer-Verlag, 1980.

[23] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[24] R. Milner. Weak bisimilarity: Congruences and equivalences, 1990. π-calculus note RM10, Manuscript.

[25] R. Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992. Previous version as Rapport de Recherche 1154, INRIA Sophia-Antipolis, 1990, and in *Proceedings of ICALP '91*, LNCS 443.

[26] R. Milner. Elements of interaction. *Communications of the ACM*, 36(1):78–89, 1993. Turing Award Lecture.

[27] R. Milner. The polyadic π-calculus: A tutorial. In F. L. Bauer, W. Brauer, and H. Schwichtenberg, editors, *Logic and Algebra of Specification*, volume 94 of *Series F*. NATO ASI, Springer, 1993. Available as Technical Report ECS-LFCS-91-180, University of Edinburgh, October 1991.

[28] R. Milner. Calculi for interaction. *Acta Informatica*, 3(8):707–737, 1996.

[29] R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, May 1999.

[30] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, part I/II. *Journal of Information and Computation*, 100:1–77, Sept. 1992.

[31] R. Milner, J. Parrow, and D. Walker. Modal logics for mobile processes. *Theoretical Computer Science*, 114:149–171, 1993.

[32] R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *LNCS*, pages 685–695. Springer, 1992.

[33] U. Nestmann. What is a 'good' encoding of guarded choice? In C. Palamidessi and J. Parrow, editors, *Proceedings of EXPRESS '97*, volume 7 of *ENTCS*. Elsevier Science Publishers, 1997. Full version as report BRICS-RS-97-45, Universities of Aalborg and Århus, Denmark, 1997. Revised version accepted (1998) for *Journal of Information and Computation*.

[34] U. Nestmann and B. C. Pierce. Decoding choice encodings. In U. Montanari and V. Sassone, editors, *Proceedings of CONCUR '96*, volume 1119 of *LNCS*, pages 179–194. Springer, 1996. Revised full version as report ERCIM-10/97-R051, European Research Consortium for Informatics and Mathematics, 1997.

[35] U. Nestmann and B. Victor. Calculi for mobile processes: Bibliography and web pages. *Bulletin of the EATCS*, 64:139–144, February 1998.

[36] F. Nielson. The typed λ-calculus with first-class processes. In *Proc. PARLE'89*, volume 366 of *Lecture Notes in Computer Science*, pages 357–373. Springer-Verlag, 1989.

[37] C. Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of POPL '97*, pages 256–265. ACM, Jan. 1997.

[38] J. Parrow. Mismatching and early equivalence, 1990. π-calculus note JP13, Manuscript.

[39] J. Parrow. Interaction diagrams. *Nordic Journal of Computing*, 2:407–443, 1995. A previous version appeared in *Proceedings of A Decade in Concurrency*, LNCS 803: 477–508, 1993.

[40] J. Parrow. On the relationship between two proof systems for the pi-calculus, 1999. π-calculus note JP15, Manuscript. Available from the author.

[41] J. Parrow and D. Sangiorgi. Algebraic theories for name-passing calculi. *Journal of Information and Computation*, 120(2):174–197, 1995. A previous version appeared in *Proceedings of A Decade in Concurrency*, LNCS 803: 477–508, 1993.

[42] J. Parrow and B. Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *Proceedings of LICS '98*, pages 176–185. IEEE, Computer Society Press, July 1998.

[43] B. C. Pierce. Foundational calculi for programming languages. In A. B. Tucker, editor, *Handbook of Computer Science and Engineering*, chapter 139. CRC Press, 1996.

[44] B. C. Pierce. Type systems for concurrent calculi, Sept. 1998. Invited tutorial at *CONCUR*, Nice, France.

[45] B. C. Pierce and D. N. Turner. Concurrent objects in a process calculus. In T. Ito and A. Yonezawa, editors, *Proceedings of TPPP '94*, volume 907 of *LNCS*, pages 187–215. Springer, 1995.

[46] B. C. Pierce and D. N. Turner. Pict: A programming language based on the pi-calculus. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*, 1999. To appear.

[47] P. Quaglia and D. Walker. On encoding pπ in mπ. In V. Arvind and R. Ramanujam, editors, *18th Conference on Foundations of Software Technology and Theoretical Computer Science (Chennai, India, December 17–19, 1998)*, lncs. sv, Dec. 1998.

[48] D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, LFCS, University of Edinburgh, 1993. CST-99-93 (also published as ECS-LFCS-93-266).

[49] D. Sangiorgi. Bisimulation in higher-order process calculi. *Journal of Information and Computation*, 131:141–178, 1996. Available as Rapport de

Recherche RR-2508, INRIA Sophia-Antipolis, 1995. An early version appeared in *Proceedings of PROCOMET'94*, pages 207–224. IFIP. North Holland Publisher.

[50] D. Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1,2):235–274, 1996. Also as Rapport de Recherche RR-2539, INRIA Sophia-Antipolis, 1995. Extracts of parts of the material contained in this paper can be found in *Proceedings of TAPSOFT '95* and *ICALP '95*.

[51] D. Sangiorgi. A theory of bisimulation for the π-calculus. *Acta Informatica*, 33:69–97, 1996. Earlier version published as Report ECS-LFCS-93-270, University of Edinburgh. An extended abstract appeared in the *Proceedings of CONCUR '93*, LNCS 715.

[52] D. Sangiorgi. Asynchronous process calculi: The first-order and higher-order paradigms, 1999. To appear in *Theoretical Computer Science*.

[53] B. Thomsen. A calculus of higher order communicating systems. In *Proceedings of POPL '89*, pages 143–154. ACM, January 1989.

[54] B. Thomsen. Plain CHOCS. A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[55] V. T. Vasconcelos and K. Honda. Principal typing schemes in a polyadic π-calculus. In E. Best, editor, *Proceedings of CONCUR '93*, volume 715 of *LNCS*, pages 524–538. Springer, 1993.

[56] B. Victor and F. Moller. The Mobility Workbench — a tool for the π-calculus. In D. Dill, editor, *Proceedings of CAV '94*, volume 818 of *LNCS*, pages 428–440. Springer, 1994.

[57] D. Walker. Objects in the π-calculus. *Journal of Information and Computation*, 116(2):253–271, 1995.

[58] N. Yoshida. Graph types for monadic mobile processes. In V. Chandru and V. Vinay, editors, *Proceedings of FSTTCS '96*, volume 1180 of *LNCS*, pages 371–386. Springer, 1996. Full version as Technical Report ECS-LFCS-96-350, University of Edinburgh.