Applied π – A Brief Tutorial

Peter Sewell

Computer Laboratory, University of Cambridge
`Peter.Sewell@cl.cam.ac.uk`

July 28, 2000

Abstract

This note provides a brief introduction to π-calculi and their application to concurrent and distributed programming. Chapter 1 introduces a simple π-calculus and discusses the choice of primitives, operational semantics (in terms of reductions and of indexed early labelled transitions), operational equivalences, PICT-style programming and typing. Chapter 2 goes on to discuss the application of these ideas to distributed systems, looking informally at the design of distributed π-calculi with grouping and interaction primitives. Chapter 3 returns to typing, giving precise definitions for a simple type system and soundness results for the labelled transition semantics. Finally, Chapters 4 and 5 provide a model development of the metatheory, giving first an outline and then detailed proofs of the results stated earlier. The note can be read in the partial order $1.(2+3+4.5)$.

Contents

1 Pi Calculi **5**
1.1 An Introduction to π . . . 5
1.2 Modelling vs Programming: choices of primitives . . . 7
1.3 Styles of Operational Semantics . . . 9
1.4 Language Implementation: the PICT experiment . . . 11
1.5 Operational Congruences . . . 12
1.6 Typing . . . 14
1.7 Further Reading . . . 15

2 Distributed π calculi **17**

3 Simple Types **23**
3.1 Polyadicity and Tuples . . . 23
3.2 Typing . . . 25
3.3 Typing and Labelled Transitions . . . 26

4 Metatheory: Overview **29**
4.1 Basic Properties of the LTS . . . 29
4.2 Coincidence of the Two Semantics . . . 33
4.3 Strong Bisimulation and Congruence . . . 34
4.4 Type Soundness and Subject Reduction . . . 35

5 Metatheory: Detailed Proofs **39**
5.1 Basic Properties of the LTS . . . 39
5.2 Coincidence of the Two Semantics . . . 48
5.3 Strong Bisimulation and Congruence . . . 53
5.4 Type Soundness and Subject Reduction . . . 58

References **63**

Acknowledgements

These notes are partly based on lectures given at the *Instructional Meeting on Recent Advances in Semantics and Types for Concurrency: Theory and Practice* held in July 1998, supported by the MATHFIT initiative of the EPSRC and LMS, organised by Rajagopal Nagarajan, Bent Thomsen and Lone Leth Thomsen. They also draw on a chapter written for a forthcoming volume edited by Howard Bowman and John Derrick. I would like to thank these organisers and editors. Some of the technical development is based on joint work with Luca Cattani and Jan Vitek. I acknowledge support from EPSRC grants GR/K 38403, GR/L 62290, and a Royal Society University Research Fellowship.

Chapter 1

Pi Calculi

Concurrency and communication are fundamental aspects of distributed systems; a great deal of work in process calculi and other areas has developed techniques for programming, specification and reasoning about them. Another basic distributed phenomenon is *name generation* – many computational entities are dynamically created with fresh names; these names can often be communicated within or between machines. The *π-calculus* of Milner, Parrow and Walker [MPW92] generalised earlier process calculi by allowing fresh channel names to be dynamically created and communicated. This gives rise to great expressive power, allowing a very simple π-calculus to be used as the basis for a concurrent programming language. It also involved the development of semantic techniques which can be directly applied to distributed systems. This chapter introduces some of the theory of π-calculi and their applications to concurrent programming. It provides only a brief and somewhat idiosyncratic introduction – for more detailed texts and pointers into the literature one should refer to Section 1.7.

We begin in Section 1.1 with a core π-calculus, giving some examples and defining the operational semantics in a reduction-semantics style. In Section 1.2 we review the main design choices that give rise to the wide variety of π-calculi in use. Many are driven by a particular theoretical result or application, particularly by whether the focus is on modelling or programming. In Section 1.3 we return to the operational semantics, defining a labelled transition semantics and relating it to the earlier reduction semantics. In Section 1.4 we consider the (concurrent, but not distributed) Pict programming language, closely based on a π-calculus. In Section 1.5 we return again to semantics, defining operational congruences. A very brief introduction to typing for π-calculi is given in Section 1.6.

1.1 An Introduction to π

The π-calculus is a calculus (an idealised modelling/programming language) in which communication between parallel processes is fundamental. Communication is on named channels: a process that offers an output of value v on the channel named c, written $\overline{c}v$, may synchronise with a parallel process that is attempting to read from c, written $cw.P$. It differs from earlier process calculi in that new channel names can be created dynamically, passed as values along other channels, and then used themselves for communication. This gives rise to great expressive power – many computational formalisms, e.g. λ-calculi, can be smoothly translated into π-calculus.

Many different π-calculi have been introduced. Some of the differences are essentially minor choices of notation and style; some are important choices that are driven by the application or theory desired. In this section we introduce a core π-calculus which still exhibits the essential phenomenon of new channel creation.

Syntax We take an infinite set $\mathcal{N}$ of *names* of channels, ranged over by a, b etc. The *process terms* are then those defined by the grammar

$$
\begin{array}{lcll}
P, Q & ::= & 0 & \text{nil} \\
 & & P \mid Q & \text{parallel composition of } P \text{ and } Q \\
 & & \overline{c}v & \text{output } v \text{ on channel } c \\
 & & cw.P & \text{input from channel } c \\
 & & \mathbf{new}\ c\ \mathbf{in}\ P & \text{new channel name creation}
\end{array}
$$

In $cw.P$ the 'formal parameter' w binds in P; in $\mathbf{new}\ c\ \mathbf{in}\ P$ the c binds in P, with scope as far to the right as possible (so $\mathbf{new}\ c\ \mathbf{in}\ P \mid Q$ should be read as $\mathbf{new}\ c\ \mathbf{in}\ (P \mid Q)$). We will work up to alpha renaming of bound names, so whenever we write a term we actually mean its alpha equivalence class. We write $\mathrm{fn}(P)$ for the set of free names of P, defined by $\mathrm{fn}(0) = \emptyset$, $\mathrm{fn}(P \mid Q) = \mathrm{fn}(P) \cup \mathrm{fn}(Q)$, $\mathrm{fn}(\overline{c}v) = \{c, v\}$, $\mathrm{fn}(cw.P) = \{c\} \cup (\mathrm{fn}(P) - w)$, $\mathrm{fn}(\mathbf{new}\ c\ \mathbf{in}\ P) = \mathrm{fn}(P) - c$. We write $\{a/x\}P$ for the process term obtained from P by replacing all free occurrences of x by a, renaming as necessary to avoid capture.

Semantics – Examples The simplest form of semantics for this calculus consists of a *reduction relation* – a binary relation between process terms, written $P \longrightarrow Q$, indicating that P can perform a single step of computation to become Q. The definition of $\longrightarrow$ will be given later; here are some examples.

The calculus allows communication between an output and an input (on the same channel) in parallel. Here the value a is being sent along the channel x:

$$\overline{x}a \mid xu.\overline{y}u \quad \longrightarrow \quad \{a/u\}(\overline{y}u) = \overline{y}a$$

There can be many outputs on the same channel competing for the same input – only one will succeed, introducing nondeterminism:

$$
\begin{array}{ccl}
 & & \overline{x}b \mid \overline{y}a \\
\overline{x}a \mid \overline{x}b \mid xu.\overline{y}u & \begin{array}{c}\nearrow \\ \searrow\end{array} & \\
 & & \overline{x}a \mid \overline{y}b
\end{array}
$$

Similarly, there can be many inputs on the same channel competing for an output:

$$
\begin{array}{ccl}
 & & \overline{y}a \mid xu.\overline{z}u \\
\overline{x}a \mid xu.\overline{y}u \mid xu.\overline{z}u & \begin{array}{c}\nearrow \\ \searrow\end{array} & \\
 & & xu.\overline{y}u \mid \overline{z}a
\end{array}
$$

A restricted name is different from all other names outside its scope – below the x bound by the $\mathbf{new}\ x\ \mathbf{in}\ \_$ is different from the x outside. Note that (using alpha equivalence) the term on the left below is *the same term* as $\overline{x}a \mid \mathbf{new}\ x'\ \mathbf{in}\ (\overline{x'}b \mid x'u.\overline{y}u)$.

$$\overline{x}a \mid \mathbf{new}\ x\ \mathbf{in}\ (\overline{x}b \mid xu.\overline{y}u) \quad \longrightarrow \quad \overline{x}a \mid \mathbf{new}\ x\ \mathbf{in}\ \overline{y}b$$

A name received on a channel can then be used itself as a channel name for output or input – here y is received on x and then used to output c:

$$\overline{x}y \mid xu.\overline{u}c \quad \longrightarrow \quad \overline{y}c$$

Finally (and most subtlely), a restricted name can be sent outside its original scope. Here y is sent on channel x outside the scope of the **new** y **in** binder, which must therefore be moved (with care, to avoid capture of other instances of y). This is known as *scope extrusion*:

$$\begin{array}{rcl}(\mathbf{new}\ y\ \mathbf{in}\ \overline{x}y \mid yv.P) \mid xu.\overline{u}c & \longrightarrow & \mathbf{new}\ y\ \mathbf{in}\ yv.P \mid \overline{y}c \\ & \longrightarrow & \mathbf{new}\ y\ \mathbf{in}\ \{c/v\}P\end{array}$$

The combination of sending channel names and scope extrusion is the essential difference between the π-calculus and earlier process calculi such as ACP, CCS and CSP.

Semantics – Definition of Reduction The reduction relation can be defined rather simply, in two stages. First we define a *structural congruence*, written $\equiv$. This is an equivalence relation over process terms that allows the two parts of a potential communication to be brought syntactically adjacent. It is the smallest equivalence relation that is a congruence and satisfies the axioms:

$$\begin{array}{rcll}P \mid 0 & \equiv & P & \\ P \mid Q & \equiv & Q \mid P & \\ P \mid (Q \mid R) & \equiv & (P \mid Q) \mid R & \\ \mathbf{new}\ x\ \mathbf{in}\ \mathbf{new}\ y\ \mathbf{in}\ P & \equiv & \mathbf{new}\ y\ \mathbf{in}\ \mathbf{new}\ x\ \mathbf{in}\ P & \\ P \mid \mathbf{new}\ x\ \mathbf{in}\ Q & \equiv & \mathbf{new}\ x\ \mathbf{in}\ (P \mid Q) & x \notin \mathrm{fn}(P)\end{array}$$

The reduction relation $\longrightarrow$ is then the smallest binary relation over process terms satisfying the following.

$$\begin{array}{ll}\textsc{Com} & \dfrac{}{\overline{c}v \mid cw.P \longrightarrow \{v/w\}P} \\[2ex] \textsc{Par} & \dfrac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \\[2ex] \textsc{Res} & \dfrac{P \longrightarrow P'}{\mathbf{new}\ x\ \mathbf{in}\ P \longrightarrow \mathbf{new}\ x\ \mathbf{in}\ P'} \\[2ex] \textsc{Struct} & \dfrac{P \equiv P' \longrightarrow P'' \equiv P'''}{P \longrightarrow P'''}\end{array}$$

Note that reduction under input prefixes is not allowed – there is no rule

$$\frac{P \longrightarrow P'}{cw.P \longrightarrow cw.P'}$$

It is a useful exercise to check that the example reductions can actually be derived.

1.2 Modelling vs Programming: choices of primitives

A good deal of early work on process calculi was focussed around *modelling* protocols (and other systems) and reasoning about those models; this fed into work e.g. on Lotos and its descendents. Some recent developments from the π-calculus, in contrast, have used it as the basis for a *programming* language: a significant shift of emphasis which affects the design of the calculi used. The programming language view is discussed further in Section 1.4 below; this subsection reviews some of the main calculus-design choices. They are here described rather informally – in some cases precise results have been proved, showing that a particular calculus is encodable in another, but great care must be taken in interpreting such results. There are many possible senses of 'encoding'; the results do not automatically transfer from the precise calculi they have been proved for to minor variants thereof. In general, we have only the beginnings of a theory of expressiveness for π-based calculi (but see the recent EXPRESS meetings such as [PC98]).

Choice Many process calculi have explicit choice or summation operators for nondeterminism, allowing for example $P + Q$ which can behave either as P or as Q (this is imprecise – in fact several different operators are possible). Explicit choice is useful for modelling and reasoning about systems, especially for writing loose specifications and for developing complete axiomatisations of operational congruences, but seems to be both unnecessary and expensive to implement from the programming language view. In some cases a form of choice is encodable in a choice-free calculus – for example a purely internal choice can be encoded in the π-calculus given in Section 1.1 above by

$$[\![P \oplus Q]\!] \stackrel{def}{=} \textbf{new } c \textbf{ in } (\overline{c}x \mid cx.P \mid cx.Q) \qquad c, x \text{ not free in } P, Q$$

but in other cases non-encodability results can be proved (see work of Nestmann and Pierce [NP96] and of Palamidessi [Pal97]).

Asynchrony The calculus of Section 1.1 is *asynchronous* – it has only a bare output $\overline{c}v$ as opposed to an output prefix $\overline{c}v.P$ that starts P when the output has been received. Again, for programming it seems that prefixing (or *synchronous*) output can be uncommon; it can generally be encoded by using explicit acknowledgements (see work of Honda and Tokoro [HT91] and of Boudol [Bou92]). Moreover, the asynchronous calculi have a closer fit to the asynchronous message delivery of packet-switched networks, and so are used as starting points for distributed calculi. Note that this usage of 'synchronous' is different from that in work on SCCS (Synchronous CCS) by Milner – there the two sides of a parallel composition were required to execute in lock-step, whereas here it refers only to synchronisation of pairs of an output prefix and an input.

Replication The calculus of Section 1.1 is rather inexpressive – it contains no way to construct infinite computations, and so is clearly not Turing-powerful (to make this precise requires care). One can add either *recursion*, e.g. with process variables X and a recursion operator $\textbf{rec}\, X.P$, or *replication* $!P$, which loosely behaves as infinitely many copies of P in parallel. In some sense (again, not made precise here), the two are inter-encodable. Generally theoretical work is simpler with replication; describing actual systems may be simpler with either. A limited form of replication, allowing only replicated input terms such as $!cx.P$, is sometimes used.

Values The calculus of Section 1.1 is further limited in that only single names can be communicated on channels – it is *monadic*. Many variants are *polyadic*, allowing tuples of names to be sent, or allow more general data, e.g. arbitrary pairs or tuples. For precise results on encoding a polyadic calculus into a monadic one see work of Yoshida [Yos96] and of Quaglia and Walker [QW98]. The addition of basic values such as booleans and natural numbers is straightforward.

Higher-Order Processes A very significant extension is to allow communication not just of basic data but also of processes themselves, or even higher-order abstractions – see work of Sangiorgi [San93] and of Thomsen [Tho93]. Many authors have studied encodings of λ calculi within a π calculus; see e.g. the work of Milner [Mil92].

Matching For some purposes constructs to test equality (matching) and inequality (mismatching) of names are added, often written $[x = y]P$ and $[x \neq y]P$, or **if** $x = y$ **then** P **else** Q. These can have delicate effects on the theory of operational congruences.

Join Patterns π-calculus channels may have many receivers. It has been argued that for a programming language a better choice of primitive is the *join pattern* that syntactically gathers together all receivers on a channel (see work of Fournet, Gonthier and others, e.g. [FG96]). Join patterns combine restriction, replicated input and linearity – **def** $x(w).P$ **in** Q is similar to **new** x **in** $Q \,|\, !xw.P$. For expressiveness one must generalise to multi-inputs such as **def** $(x_1(w_1) \wedge .. \wedge x_n(w_n)).P$ **in** Q, or even to disjunctions of multi-inputs. An example reduction would be

$$\begin{array}{l}\textbf{def } (x_1(w_1) \wedge x_2(w_2)).P \textbf{ in } Q \,|\, \overline{x_1}3 \,|\, \overline{x_2}7 \\ \longrightarrow \\ \textbf{def } (x_1(w_1) \wedge x_2(w_2)).P \textbf{ in } Q \,|\{3,7/w_1,w_2\}P\end{array}$$

Concrete Syntax Many minor variations of concrete syntax are used in the literature. Most significantly, if one wishes a syntax with a good representation in plain ascii, then outputs and inputs can be written as $c!v$ and $c?w.P$ instead of the $\overline{c}v$ and $cw.P$ used in Section 1.1. The exclamation mark would then confusingly be used both for output and replication, so replication (or replicated input) can be written $*P$ (or $*c?w.P$) instead of $!P$ (or $!cw.P$). Restriction has been written either with a greek nu as $(\boldsymbol{\nu} c)P$ or as **new** c **in** P. Some work uses small round brackets to indicate binders and angle brackets to indicate free names, so writing outputs and inputs as $\overline{c}\langle v\rangle$ and $c(w).P$

1.3 Styles of Operational Semantics

The basic semantic theory for a π-calculus comprises four main parts which we describe in order of increasing sophistication, usefulness and technical complexity. Most simply, one can define the internal *reduction* relation of processes by means of reduction axioms and a structural congruence, as above. This builds on the Chemical Abstract Machine ideas of Berry and Boudol [BB92], and the π semantics of Milner [Mil92]. To describe the interactions of processes with their environment one requires more structure: some *labelled transition* relation, specifying the potential inputs and outputs of processes. Several different forms of labelled transition system are discussed in this subsection below. Reduction and labelled transition relations are both very intensional, keeping the algebraic structure of process terms. One can abstract from the internal states of processes by quotienting by some *operational congruence*, such as *bisimulation*, defined using the labelled transition relations or in terms of barbs (a degenerate form of labelled transitions). Bisimulation equivalence classes are still rather intensional, however. They may be appropriate when working with models of concurrent systems, and may provide useful proof techniques, but to obtain a clear relationship to the behaviour of programming languages one must abstract further, quotienting by an *observational congruence*. The identification of an appropriate notion of observation for a given programming language may be non-trivial. Operational congruences and observations are discussed further in Section 1.5 below. We now give a labelled transition semantics for the calculus of Section 1.1.

A Labelled Transition Semantics The labelled transition relation has the form

$$A \vdash P \xrightarrow{\ell} Q$$

where A is a finite set of names, $\mathrm{fn}(P) \subseteq A$, and ℓ is a *label*; it should be read as 'in a state where the names A may be known by process P and by its environment, the process P can do ℓ to become Q. The labels ℓ are

$$
\begin{array}{lll}
\ell \ ::= & \tau & \text{internal action} \\
& \overline{x}v & \text{output of } v \text{ on } x \\
& xv & \text{input of } v \text{ on } x
\end{array}
$$

The transition relation is defined as the smallest relation satisfying the rules below.

$$
\textsc{Out}\ \frac{}{A \vdash \overline{x}v \xrightarrow{\overline{x}v} 0}
\qquad
\textsc{In}\ \frac{}{A \vdash xp.P \xrightarrow{xv} \{^v/_p\}P}
$$

$$
\textsc{Par}\ \frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q}
\qquad
\textsc{Com}\ \frac{A \vdash P \xrightarrow{\overline{x}v} P' \quad A \vdash Q \xrightarrow{xv} Q'}{A \vdash P \mid Q \xrightarrow{\tau} \textbf{new } \{v\} - A \textbf{ in } (P' \mid Q')}
$$

$$
\textsc{Res}\ \frac{A, x \vdash P \xrightarrow{\ell} P' \quad x \notin \mathrm{fn}(\ell)}{A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\ell} \textbf{new } x \textbf{ in } P'}
\qquad
\textsc{Open}\ \frac{A, x \vdash P \xrightarrow{\overline{y}x} P' \quad y \neq x}{A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\overline{y}x} P'}
$$

$$
\textsc{Struct Right}\ \frac{A \vdash P \xrightarrow{\ell} P' \quad P' \equiv P''}{A \vdash P \xrightarrow{\ell} P''}
$$

In all rules with conclusion of the form $A \vdash P \xrightarrow{\ell} Q$ there is an implicit side condition $\mathrm{fn}(P) \subseteq A$. Symmetric versions of Par and Com are elided. Here the free names of a label are $\mathrm{fn}(\tau) = \{\}$, $\mathrm{fn}(\overline{x}v) = \mathrm{fn}(xv) = \{x, v\}$. We write A, x for $A \cup \{x\}$ where x is assumed not to be in A. If $A = \{a_1, \ldots a_n\}$ then **new** A **in** P denotes **new** a_1 **in** $\ldots$**new** a_n **in** P (note that if A is not empty or a singleton then strictly this is only well-defined up to structural congruence).

It is a good exercise to derive τ transitions for the example reductions above – especially for the scope extrusion example. Note that there is a transition

$$\{x\} \vdash \textbf{new } z \textbf{ in } \overline{x}z \xrightarrow{\overline{x}w} 0$$

for *any* $w \neq x$, as we are working up to alpha equivalence. A more substantial exercise is to extend both this and the reduction semantics with richer values, e.g. with arbitrary tupling.

This should now be compared with the reduction semantics. Firstly, one can show that structurally congruent processes have the same labelled transitions (equivalently, that a Struct Left rule is admissible).

Theorem 1 *If $P' \equiv P$ then $A \vdash P' \xrightarrow{\ell} Q$ iff $A \vdash P \xrightarrow{\ell} Q$.*

One can then show that the reduction and transition semantics give exactly the same internal steps.

Theorem 2 *If $\mathrm{fn}(P) \subseteq A$ then $P \longrightarrow Q$ iff $A \vdash P \xrightarrow{\tau} Q$.*

The reduction semantics is probably easier to understand – it is common when designing a new calculus to first specify its reductions. It does not, however, tell us how an arbitrary subprocess can interact with its environment; for that, and hence for explicit characterisations of operational congruences, we need the LTS. Moreover, the labelled transitions of a process P are defined inductively on its term structure, whereas the reductions are not – broadly, to show that some particular reduction exists it is easier to use the reduction semantics, but to enumerate *all* the reductions the LTS is appropriate.

One can choose to define either the explicitly-indexed transitions $A \vdash P \xrightarrow{\ell} Q$, as here, or un-indexed transitions of the form $P \xrightarrow{\mu} Q$. In the former, an output of a free name and an output of a new name can be distinguished by reference to A – we have

$$\{x,y\} \vdash \mathbf{new}\ z\ \mathbf{in}\ \overline{x}y \xrightarrow{\overline{x}y} 0 \quad \text{and} \quad \{x,y\} \vdash \mathbf{new}\ z\ \mathbf{in}\ \overline{x}z \xrightarrow{\overline{x}w} 0$$

with $y \in \{x,y\}$ and $w \notin \{x,y\}$ respectively. In the latter, the distinction must be carried in the label μ and the rules require more delicate side-conditions. The explicitly-indexed style seems (to this author) to be conceptually slightly clearer, though at some notational cost; it also leads to simpler notions of trace and generalises to typed systems with subtyping.

The transition system above is expressed as relations over the process syntax (as are most π-calculus semantics). In contrast one can take a more model-theoretic view, axiomatising the structure required of a π-LTS with an arbitrary set of states. This is developed in [CS00]. Another semantic choice is that between *early* and *late* transition systems. We have given an early system, with inputs instantiated immediately – see the overview of Quaglia [Qua99] for discussion of early, late and open semantics.

1.4 Language Implementation: the PICT experiment

The π-calculus is sufficiently expressive to be used as the basis for a programming language. The literature contains a number of encodings of λ-calculi and data structures into π. For some theoretical work one can therefore combine the benefits of a rather small calculus, having a simple semantics, with the flexibility of high-level constructs, provided by encodings. More practically, the language PICT has been developed since 1992, mostly by Pierce and Turner, to experiment with programming in the π-calculus, with rich type systems for communicating concurrent objects, and with efficient implementation techniques. Loosely, it has the same relationship to the π-calculus as functional languages do to the λ-calculus:

Sequential (functional)	Concurrent
λ-calculus	π-calculus
functions	processes
function application	parallel composition
beta reduction	communication
LISP, ML, Haskell, etc.	PICT

Documentation and an implementation are available electronically [PT98]; descriptions of the design and implementation are in [Tur96, PT00]. See also implementations and papers on the Join Language [Joi].

A number of programming idioms turn out to be useful – we will now touch on a couple (here we use a polyadic π-calculus, not precisely specified, with tuples $\langle x_1 .. x_n\rangle$, tuple patterns $\langle x_1 .. x_n\rangle$, replicated input $!c\langle x_1 .. x_n\rangle.P$, and basic arithmetic). One can define process abstractions, analogous to local function definitions, as follows:

$$\begin{array}{l} \mathbf{new}\ \mathit{plustwo}\ \mathbf{in} \\ \quad !\mathit{plustwo}\langle x\ r\rangle.\overline{r}\langle x+2\rangle \\ \quad |\ \mathbf{new}\ r\ \mathbf{in} \\ \qquad \overline{\mathit{plustwo}}\langle 56\ r\rangle \mid r\langle z\rangle.\overline{\mathit{printi}}\langle z\rangle \end{array}$$

Here one can send on the *plustwo* channel a number x and a result channel r; the server will send back $x+2$ on r. Channels can also be used to implement locking rather directly –

here is an approximation to a two method object implementation, with mutual exclusion between the bodies of the methods:

$$
\begin{array}{l}
\textbf{new}\ \mathit{lock}\ \textbf{in} \\
\quad \overline{\mathit{lock}}\langle\rangle \\
\quad |\ !\mathit{method1}\langle \mathit{arg}\rangle. \\
\qquad \mathit{lock}\langle\rangle. \\
\qquad\quad \ldots \\
\qquad\quad \overline{\mathit{lock}}\langle\rangle \\
\\
\quad |\ !\mathit{method2}\langle \mathit{arg}\rangle. \\
\qquad \mathit{lock}\langle\rangle. \\
\qquad\quad \ldots \\
\qquad\quad \overline{\mathit{lock}}\langle\rangle
\end{array}
$$

Access to the implementation could be passed around as a simple tuple of method (channel) names, e.g. as $\langle \mathit{method1}\ \ \mathit{method2}\rangle$.

The reduction semantics of the π-calculus is highly nondeterministic. There is therefore a basic design decision for any language implementation: how should that nondeterminism be resolved? There are several possible choices:

1. One could consult a pseudo-random number generator (or a true source of quantum-mechanical randomness) at every choice. This might be desirable for a simulation language, but is prohibitively expensive for a programming language.

2. One could fix an evaluation strategy (as one does in ML, say). This would highly constrain future compiler writers, who would always have to schedule π-processes in the same way. It would prevent optimisations and severely constrain distributed implementations.

3. One could allow a compiler to use any 'reasonable' evaluation strategy. This is delicate, as some fairness conditions are required, e.g. to ensure the process $\textbf{new}\ x\ \textbf{in}\ \big(\overline{x}\,|\,!\,x.\overline{x}\,|\,x.\overline{\mathrm{print}}\text{“ping”}\big)$ does eventually print the 'ping'.

PICT adopts the third approach. The current implementation has a scheduler as follows: it maintains a *state* consisting of a *run queue* of processes to be scheduled (round robin) together with *channel queues* of processes waiting to communicate. It executes in steps, in each of which the process at the front of the run queue is removed and processed. This internal behaviour of the implementation is described by Turner in [Tur96, Ch. 7] and incorporated into the abstract machine given in [Sew97]. When an output or input on a library channel reaches the front of the run queue some special processing takes place. For many library channels this consists of a single call to a corresponding Unix IO routine. The implementation is thus entirely deterministic.

Note that a minor syntactic change to a program in such a language, such as swapping two parallel components, may result in completely different behaviour. This can make debugging difficult, but it seems to be inescapable.

1.5 Operational Congruences

A reduction or labelled-transition semantics gives only a rather intensional notion of the behaviour of processes. One often needs a more extensional semantics, abstracting from the syntax of the calculus, for example to make precise any of the following questions:

- when do two processes have the same behaviour?
- when does a process meet a specification (itself expressed as a process)?
- when is a program transformation correct?
- when is an abstract machine correct?
- when are two calculi equally expressive?

An extensional semantics is often defined as a quotient of the syntax by some *operational equivalence* or *operational preorder*, itself defined using reductions or labelled transitions. Much of the π literature involves either a *bisimulation congruence* defined over labelled transitions or *barbed bisimulation congruence*, defined using reductions and *barbs* – vestigial labelled transitions. We will sketch definitions for the π-calculus given earlier.

Take *bisimulation* $\dot{\sim}$ to be the largest family of relations indexed by finite sets of names such that each $\dot{\sim}_A$ is a symmetric relation over $\{\, P \mid \mathrm{fn}(P) \subseteq A \,\}$ and for all $P \mathrel{\dot{\sim}_A} Q$,

- if $A \vdash P \xrightarrow{\ell} P'$ then $\exists Q' \,.\, A \vdash Q \xrightarrow{\ell} Q' \wedge P' \mathrel{\dot{\sim}_{A \cup \mathrm{fn}(\ell)}} Q'$

(one must check that $\dot{\sim}$ exists uniquely, but we omit a rigorous formulation here). Intuitively this says that P is equivalent to Q if any transition of one can be matched by a transition of the other, with the resulting states also equivalent. On occasion a finer relation, obtained by closing up under substitutions, is required. Define $\sim$ by $P \sim_A Q$ iff for all substitutions σ with $\mathrm{dom}(\sigma) \cup \mathrm{ran}(\sigma) \subseteq A$ we have $\sigma P \mathrel{\dot{\sim}_A} \sigma Q$.

On the other hand, to define *barbed bisimulation* first take barbs as follows: $P{\downarrow_x}$ if P can do an input on channel x and $P{\downarrow_{\overline{x}}}$ if P can do an output on x. Now $\dot{\sim}^b$ is the largest symmetric relation such that for all $P \mathrel{\dot{\sim}^b} Q$,

- if $P \longrightarrow P'$ then $\exists Q' \,.\, Q \longrightarrow Q' \wedge P' \mathrel{\dot{\sim}^b} Q'$, and moreover
- $P{\downarrow_\alpha}$ implies $Q{\downarrow_\alpha}$.

This requires only the reductions and immediate offers of communication to be matched. Barbed bisimulation congruence is defined by closing under all contexts – say $P \sim^b Q$ iff for contexts C we have $C[P] \dot{\sim}^b C[Q]$.

These are rather different styles of definition, yet they sometimes define the same equivalence, giving one confidence that one is dealing with a robust notion [San93, MS92]. Both have advantages. The definition of barbed bisimulation congruence does not depend on an LTS, and so may be readily given for novel calculi in which labelled transitions are not well-understood. On the other hand, the definition involves quantification over all contexts, making it harder to prove instances of the equivalence (but see techniques in the thesis of Fournet [Fou98]) and to have a clear intuition as to its significance.

For several of the questions at the beginning of this subsection it is important to have an equivalence or preorder that is a *congruence* – i.e. that is preserved by the constructors of the calculus. This enables (in)equational reasoning to be used freely. The fact that $\sim^b$ is a congruence is immediate from the definition; the fact that $\dot{\sim}$ or $\sim$ is may require some delicate proof. Indeed, working with an indexed relation requires a little care even to state the congruence property, to keep the A-indexing straight. Consider a family $\mathcal{S}$ of relations indexed by finite sets of names such that each $\mathcal{S}_A$ is a relation over $\{\, P \mid \mathrm{fn}(P) \subseteq A \,\}$. Say $\mathcal{S}$ is an indexed congruence if each $\mathcal{S}_A$ is an equivalence relation and the following hold.

$$\textsc{In}\,\frac{P\ \mathcal{S}_{A,w}\ P' \quad c \in A}{cw.P\ \mathcal{S}_A\ cw.P'} \qquad \textsc{Res}\,\frac{P\ \mathcal{S}_{A,c}\ P'}{\textbf{new}\ c\ \textbf{in}\ P\ \mathcal{S}_A\ \textbf{new}\ c\ \textbf{in}\ P'}$$

$$\textsc{Par}\,\frac{P\ \mathcal{S}_A\ P' \quad Q\ \mathcal{S}_A\ Q'}{P\,|\,Q\ \mathcal{S}_A\ P'\,|\,Q'}$$

Theorem 3 *Bisimulation $\dot\sim$ is an indexed congruence.*

This result depends on the exact calculus used; in many variants one must move to $\sim$ to obtain a congruence for input prefixing.

Work on operational congruences for process calculi without scope extrusion showed that there are many more-or-less plausible notions of equivalence, differing e.g. in their treatment of linear/branching time, of internal reductions, of termination and divergence, etc. Some of the space is illustrated in the surveys of van Glabbeek [Gla90, Gla93]. Much of this carries over to π-calculi. For example, we can define trace-based equivalences straightforwardly. For partial traces (aka prefix-closed traces) write

$$A_1 \vdash P_1 \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_n} P_{n+1}$$

to mean $\exists P_2, \ldots, P_n, A_2, \ldots, A_n\ .\ \forall i \in 1..n\ .\ A_{i+1} = A_i \cup \mathrm{fn}(\ell_i) \wedge A_i \vdash P_i \xrightarrow{\ell_i} P_{i+1}$. If $\mathrm{fn}(P, Q) \subseteq A$ then the partial A-traces of P and partial trace equivalence are defined by

$$\begin{array}{rcl} \mathrm{ptr}_A(P) & \stackrel{def}{=} & \{\, \ell_1 \mathrel{..} \ell_n \mid \exists P'\ .\ A \vdash P \xrightarrow{\ell_1} \ldots \xrightarrow{\ell_n} P' \,\} \\ P =^{\mathrm{ptr}}_A Q & \stackrel{def}{\Leftrightarrow} & \mathrm{ptr}_A(P) = \mathrm{ptr}_A(Q) \end{array}$$

Standard facts such as $P \mathrel{\dot\sim_A} Q \implies P =^{\mathrm{ptr}}_A Q$ go through as usual. For π-calculi there are also other choices, e.g. between *open*, *late* and *early* bisimulations.

All this diversity raises a problem: how, in some particular application of a π-calculus, should one choose an *appropriate* equivalence or congruence? This was studied for Pict-like programming languages in [Sew97]. From the discussion of the scheduling behaviour of implementations described in Section 1.4 above it is immediate that no realistic implementation will be bisimilar (in any sense) to the labelled transition semantics; several other choices are arguably determined by more subtle implementation properties.

1.6 Typing

In the monadic π-calculus of Section 1.1 the value sent on a channel is always of the form expected by a receiver – a single name. In calculi with more interesting values this no longer holds, for example a polyadic π process might contain an output of a pair in parallel with an input expecting a triple:

$$\overline{c}\langle a\ b\rangle \mid c\langle x\ y\ z\rangle.P$$

Intuitively this should be regarded as an execution error, just as an application to a pair of a function that takes a triple of arguments would be. In both cases such errors can be prevented by imposing a type system. The types for a simple calculus with tuples might be

$$\begin{array}{rcll} T & ::= & \textbf{chan}\, T & \text{type of channel names carrying } T \\ & & \langle T_1 \mathrel{..} T_n \rangle & \text{type of tuples of values of types } T_1 \mathrel{..} T_n \end{array}$$

with a typing judgement $\Gamma \vdash P$ **proc**, where Γ is a finite partial function from names to types, read as 'under the assumptions Γ on names the process P is well-typed'. If Γ takes c to be a channel carrying pairs

$$\Gamma = c : \mathbf{chan}\,\langle T\, U\rangle,\ a : T,\ b : U$$

then we would expect $\Gamma \vdash \overline{c}\langle a\ b\rangle$ **proc** to hold but $\Gamma \vdash c\langle x\ y\ z\rangle.P$ **proc** not to hold. Note that types are the types of values, not of processes – in contrast to λ-calculus type systems a process here does not have a type but is simply well-formed or not.

To make this precise requires the definition of the syntax and semantics (reduction and/or labelled transition) of a π-calculus with polyadic or tuple communication, a definition of execution error and a definition of the typing rules. One should then state and prove that well-typed processes do not have execution errors and that typing is preserved by reductions or (a stronger result) by labelled transitions. This is carried out in Chapter 3.

More sophisticated type systems are an active research area, addressing polymorphism, linearity, deadlock freedom, locality and security. We refer the reader to [Pie98] for details and pointers into the literature.

1.7 Further Reading

The interested reader is referred to the book *Communicating and Mobile Systems: the π-Calculus* by Milner [Mil99] and to the Mobility web page maintained by Uwe Nestmann [Nes], from which much of the literature is accessible electronically. The page includes pointers to the introductory papers by Milner, Parrow and Walker [MPW92], a tutorial by Parrow, a text on *Foundational Calculi for Programming Languages* by Pierce, an annotated bibliography by Honda, and many other useful pointers.

Chapter 2

Distributed π calculi

A body of recent research has applied techniques derived from π-calculi to address problems in distributed systems and programming languages. In some ways there is a good match between the asynchronous π-calculus and distributed systems:

- π gives a very clear treatment of concurrency, fundamental to distributed systems;
- π asynchronous message passing is close to reliable datagram communication, which lies not far above IP;
- the π treatment of naming is widely applicable. Most obviously, there are tight analogies between
  - communication channels (with read/write operations)
  - references (with deref/assign)
  - cryptographic keys (with decrypt/encrypt)

  The essential point is that *π-style semantics provide a tractable and compositional way of describing systems that can locally generate fresh names.* Note that π names are *pure*, in the sense of Needham [Nee89]; they are not assumed to contain any information about their creation.

On the other hand, there are many important issues that standard π-calculi do not address, such as:

- point-to-point and multicast communication
- failure (of machines and communication links), time and timeouts
- code and agent migration
- security (secrecy, integrity, trust, cryptography)
- the distinction between local and non-local performance
- quality of service

These cannot be abstracted away – there are many interesting language design/semantics problems, and in some cases delicate protocol design problems. It has proved fruitful to study these problems in the context of particular calculi, designed for the purpose. In the remainder of this chapter we highlight some of the choices in the (rather large) design space of such calculi, focussing on the possible grouping and interaction primitives. This discussion is only a starting point – we touch on some example calculi but cannot here do justice even to these, let alone to the many other works in the field. Our examples are taken from:

- The π_l calculus of Amadio and Prasad [AP94], for modelling the failure semantics of Facile [TLK96].
- The Distributed Join Calculus of Fournet et al [FGL$^+$96], intended as the basis for a mobile agent language.
- The Spi calculus of Abadi and Gordon [AG97], for reasoning about security protocols.
- The Dπ calculus of Riely and Hennessy [RH99], used to study typing for open systems of mobile agents.
- The dpi calculus of Sewell [Sew98], used to study locality enforcement of capabilities with a subtyping system.
- The Ambient calculus of Cardelli and Gordon [CG98], used for modelling security domains.
- The Agent and Nomadic π calculi of Sewell, Wojciechowski and Pierce [SWP98], introduced to study communication infrastructures for mobile agents.
- The Seal calculus of Vitek and Castagna [VC98], focussing on protection mechanisms including revocable capabilities.
- The Box-π calculus of Sewell and Vitek [SV99, SV00], used to study secure encapsulation of untrusted components and causality typing.

Grouping The first point is that standard π-calculi do not have any notion of the identity of processes; the syntax describes only collections of atomic processes (outputs, inputs etc.) in parallel. For example, suppose we have two processes P and Q. In π we might have

$$P \,|\, Q \quad \longrightarrow \ldots \longrightarrow \quad (R_1 \,|\, .. \,|\, R_n)$$

for some R_i; the calculus does not have any association between these R_i and the original P and Q, so the identity of the components is lost. For many purposes, therefore, one must add primitives for grouping process terms, into units of:

- failure (e.g. machines or runtime system instances);
- migration (e.g. mobile agents);
- trust (e.g. large administrative domains or small secure critical regions);
- synchronisation (i.e. regions within which an output and an input on the same channel name can interact).

The π-calculus is often referred to as a calculus of mobile processes, but it is perhaps more accurate to view it as a calculus in which the scopes of names are mobile — processes can move only in the sense that their interaction possibilities, represented as the set of channel names they know, can change.

Hierarchy Grouping primitives can either be given a flat structure, so a whole system is simply a set of groups, or some hierarchy – either two-level or an arbitrary tree. To simply model a flat set of named machines, each of which is running some π-style process code, one might take a new syntactic category of *configuration*, e.g. defined by

$$\begin{array}{lcll} C, D & ::= & m[\![P]\!] & \text{machine } m \text{ running process } P \\ & & 0 & \text{nil} \\ & & C \mid D & \text{parallel composition of } C \text{ and } D \\ & & \textbf{new } c \textbf{ in } C & \text{new name binder} \end{array}$$

This would support a semantics with machine failure, or (as in Dπ) systems with code mobility. If one wishes to consider migration of part of the process running at a machine then a more elaborate hierarchy is required. A two-level hierarchy suffices: a system of named agents, each containing a π-style process and running on a named machine, can be described by configurations

$$\begin{array}{lcll} C, D & ::= & a@m[P] & \text{agent } a \text{ on machine } m \text{ running process } P \\ & & 0 & \text{nil} \\ & & C \mid D & \text{parallel composition of } C \text{ and } D \\ & & \textbf{new } c \textbf{ in } C & \text{new name binder} \end{array}$$

This is roughly the approach of Nomadic π. There are primitives for agent creation and migration:

$$\begin{array}{lcll} P, Q, R & ::= & \textbf{agent } b = Q \textbf{ in } R & \text{create a new agent with body } Q \\ & & \textbf{migrate to } m.P & \text{migrate to machine } m \\ & & \ldots & \end{array}$$

with b binding in P and Q, and reductions such as

$$\begin{array}{rcl} a@m[P \mid \textbf{agent } b = Q \textbf{ in } R] & \longrightarrow & \textbf{new } b \textbf{ in } a@m[P \mid R] \mid b@m[Q] \\ a@m[P \mid \textbf{migrate to } n.R] & \longrightarrow & a@n[P \mid R] \end{array}$$

(where $b \notin \text{fn}(P, a, m)$) for creation and migration respectively. In the first, the new agent b is created on the same machine as the creating agent a; in the second note that the whole of agent a migrates to machine n. In both reductions the continuation process R can execute only after the creation/migration.

A two-level hierarchy provides a simple setting for considering inter-agent communication (the goal of the Nomadic π work), but for several purposes an arbitrary tree-shaped hierarchy is preferable. One might wish to represent larger units than machines, e.g. intranets delimited by firewalls; to model smaller units of software, e.g. untrusted components of an application that must be securely encapsulated; or to support a smooth programming style, in which applications automatically take their subcomponents with them on migration. The Distributed Join, dpi, Ambient, Seal and Box-π calculi all take tree-shaped hierarchies. The latter three add a named-group primitive to the syntax of processes

$$\begin{array}{lcll} P & ::= & \ldots & \\ & & a[P] & \text{ambient/seal/box named } a \text{ containing } P \end{array}$$

– they do not require a separate notion of configuration. The hierarchy is determined by the nesting structure of terms. In the Distributed Join and dpi calculi groups (there called locations) have unique names and the hierarchy is determined by the binding structure; we omit the details here.

Group Naming Grouped entities can be anonymous or named; if named, the names may be unique or non-unique (this, as many other things, might be enforced either by the design of the calculus syntax or by some additional well-formedness or typing condition). Unique naming simplifies some programming and so was adopted in the Distributed Join, dpi and Nomadic π calculi. One should note, however, that in a network with potentially malicious components a machine (or larger administrative unit) may not have control of the namespace used by incoming entities. In this case non-unique names are appropriate and were adopted in the Ambient, Seal and Box π calculi.

Interaction There is a vast range of possible primitives for the movement of groups and for interaction between them. We consider three aspects below.

Interaction across the group hierarchy Calculi differ in the extent to which communication, migration or other interaction is allowed *across* the group hierarchy. There are three main alternatives, which we discuss in the context of π-style communication.

- *Location-independent.* An output $\overline{c}v$ can interact with a corresponding input on c irrespective of their relative position in the hierarchy. This was adopted in the Distributed Join and High-level Nomadic π calculi, for the ease of programming that it supports. Its implementation requires complex distributed infrastructure, however, and the high level of abstraction makes failure and attack semantics problematic.
- *Local.* An output can interact only with a 'nearby' input. One might take a subset of the following primitives.

$\overline{c}^{\star}v$	output v on channel c within this group
$\overline{c}^{\uparrow}v$	output v on channel c to parent
$\overline{c}^{\downarrow n}v$	output v on channel c to child n
$\overline{c}^{\rightarrow n}v$	output v on channel c to sibling n
$\overline{c}^{\uparrow n}v$	output v on channel c to parent if a child of n
$\overline{c}^{a@m}v$	output v on channel c to agent a on machine m

  Local primitives can be implemented more simply (the last was adopted in Low-level Nomadic π for this reason). They also support *encapsulation* – constraining the interaction possibilities of untrusted code by containing it within a group. This is important in the Ambient, Seal and Box-π calculi (Box-π adopts the first three local output primitives).
- *Path-based.* An intermediate possibility allows non-local output but requires the sender to specify an explicit *path* to the receiver.

$\overline{c}^{path}v$	follow path p then output v on channel c

  where paths are sequences of local movements:

path	::=	$\star$	output within this group
		$\uparrow.path$	go to parent and then follow *path*
		$\downarrow n.path$	go to child n and then follow *path*

  This is a variant of the Ambient calculus mobility primitives, in which an ambient must acquire a path of capabilities in order to migrate; restricting the spread of such capabilities is a basic mechanism for secure programming.

Inputs At the receiver end, one might allow inputs from any source or from a specified source:

$cp.P$	input on channel c from any sender
$c^{\downarrow n}p.P$	input on channel c from child n

The guarantee of authenticity implied by the latter is used in the Seal and Box-π work on encapsulation. Most flexibly, one can input from a set of sources but bind the actual source to a variable, e.g. with the primitive below in which n binds in P and is replaced by the actual source child name in a communication.

$c^{\downarrow(n)}p.P$	input on channel c from any child

Communicated values Finally, we turn to the values which may be involved in a communication or migration. Most simply, one might communicate basic values as in a polyadic π-calculus – names and tuples of names. Generalising to higher-order communication would allow code – processes and abstractions – to be communicated. A rather different generalisation allows an executing process to migrate with all of its state. This is sometimes referred to as *strong* migration, with code mobility as *weak* migration. The two can be unified by introducing a *grab* primitive for capturing the state of a group into a first-class value.

Semantics The detailed design of a syntax and reduction semantics for a distributed π-calculus can be delicate, depending on the choice of primitives (indeed, the desire for a clean reduction semantics may affect that choice). We do not give a full discussion here, but refer the reader to the papers introducing particular calculi cited above. For each, it may be interesting to consider whether the syntax keeps the parts of a group syntactically adjacent or not (and why!), whether a grouping hierarchy is maintained in the term or binding structure, how uniqueness of naming is enforced, and the locality of the interactions that are permitted.

Chapter 3

Simple Types

To discuss typing we must first extend the calculus to allow communication of richer values than single names. In the subsequent sections we define a simple type system and state its soundness properties with respect to the reduction semantics, and then with respect to labelled transition semantics.

3.1 Polyadicity and Tuples

The calculus of Section 1.1 is *monadic*, i.e. outputs $\overline{x}v$ and inputs $xy.P$ send and receive only single names – here the value v and binder y are single names. Much work uses *polyadic* calculi, allowing outputs $\overline{x}\langle v_1 \mathinner{..} v_n\rangle$ of n-tuples of names and corresponding inputs $x\langle y_1 \mathinner{..} y_n\rangle.P$ that have n binders. Reductions

$$\overline{x}\langle v_1 \mathinner{..} v_n\rangle \mid x\langle y_1 \mathinner{..} y_n\rangle.P \longrightarrow \{v_1/y_1,...,v_n/y_n\}P$$

involve simultaneous substitution of names for names. A mild generalisation allows communication not just of flat tuples of names, but also of nested tuples. It seems that this eases programming enough to outweigh the slight notational complexity above polyadic calculi, and supports cleaner type systems. We introduce syntactic classes of *values* and corresponding *patterns*:

v	$::=$	x	name x
		$\langle v_1 \mathinner{..} v_n\rangle$	tuple of $v_1 \mathinner{..} v_n$, with $n \geq 0$
p	$::=$	x	name x
		$\langle p_1 \mathinner{..} p_n\rangle$	tuple pattern, with $n \geq 0$ and distinct names

Henceforth v and p will range over arbitrary values and patterns respectively, not just over names. Processes are as before except that outputs and inputs involve values and patterns, and new-binders are annotated by a type T (defined in the next section):

P, Q	$::=$	0	nil
		$P \mid Q$	parallel composition of P and Q
		$\overline{c}v$	output v on channel c
		$cp.P$	input from channel c
		$\textbf{new}\ c : T\ \textbf{in}\ P$	new channel name creation

We write $\mathrm{bn}(p)$ for the names of p. Note that the channel used in an output or input must still be a name, not an arbitrary value. Applying a substitution $\{v/x\}$ of a value for a name to a process P may therefore be undefined, e.g. in the expression $\{\langle w\,w\rangle/x\}\overline{x}y$, as $\overline{\langle w\ w\rangle}y$ is not in the syntax. There is a technical choice here – one could instead allow outputs and inputs on any value in the syntax, perhaps defining the semantics to allow communication actually take place only on names. Here we prefer to syntactically exclude such pathological processes.

A substitution $\{v/p\}$ of a value for a pattern may also be undefined if v does not match the shape of p, for example $\{\langle x\rangle/\langle y\ z\rangle\}$. We define a partial function $\{{}^{-}\!/_{-}\}$, taking a pattern and a value and giving, where it is defined, a partial function from names to values.

$$\begin{array}{rcll}
\{{}^{v}\!/_{x}\} & = & \{x \mapsto v\} & \\
\{{}^{\langle v_1 \,..\, v_{k'}\rangle}\!/_{\langle p_1 \,..\, p_k\rangle}\} & = & \{{}^{v_1}\!/_{p_1}\} \cup \ldots \cup \{{}^{v_k}\!/_{p_k}\} & \text{if } k = k' \\
 & & \text{undefined, otherwise} &
\end{array}$$

If σ is a finite partial function from names to values we define the application of σ to a value in the obvious way, and to process as follows.

$$\begin{array}{rcll}
\sigma(0) & = & 0 & \\
\sigma(P \,|\, Q) & = & \sigma(P) \,|\, \sigma(Q) & \\
\sigma(\overline{c}v) & = & \overline{\sigma(c)}\sigma(v) & \text{if } \sigma(c) \text{ a name} \\
\sigma(cp.P) & = & \sigma(c)\hat{p}.\sigma(\hat{P}) & \text{if } \sigma(c) \text{ a name} \\
\sigma(\mathbf{new}\ c : T\ \mathbf{in}\ P) & = & \mathbf{new}\ \hat{c} : T\ \mathbf{in}\ \sigma(\hat{P}) & \\
 & & \text{undefined, otherwise} &
\end{array}$$

where in the input case $cp.P = c\hat{p}.\hat{P}$ and $\mathrm{bn}(\hat{p}) \cap (\mathrm{dom}(\sigma) \cup \mathrm{fn}(\mathrm{ran}(\sigma))) = \emptyset$, and in the new case $\mathbf{new}\ c : T\ \mathbf{in}\ P = \mathbf{new}\ \hat{c} : T\ \mathbf{in}\ \hat{P}$ and $\hat{c} \notin (\mathrm{dom}(\sigma) \cup \mathrm{fn}(\mathrm{ran}(\sigma)))$.

Semantics – Definition of Reduction The reduction relation can now be defined as before except that the structural congruence axioms must carry types:

$$\begin{array}{rcll}
\mathbf{new}\ x : T\ \mathbf{in}\ \mathbf{new}\ y : U\ \mathbf{in}\ P & \equiv & \mathbf{new}\ y : U\ \mathbf{in}\ \mathbf{new}\ x : T\ \mathbf{in}\ P & x \neq y \\
P \,|\, \mathbf{new}\ x : T\ \mathbf{in}\ Q & \equiv & \mathbf{new}\ x : T\ \mathbf{in}\ (P \,|\, Q) & x \notin \mathrm{fn}(P)
\end{array}$$

and the (COM) and (RES) rules, which become

$$\textsc{Com}\quad \frac{}{\overline{c}v \,|\, cp.P \longrightarrow \{v/p\}P \quad \text{if } \{v/p\}P \text{ defined}}$$

$$\textsc{Res}\quad \frac{P \longrightarrow P'}{\mathbf{new}\ x : T\ \mathbf{in}\ P \longrightarrow \mathbf{new}\ x : T\ \mathbf{in}\ P'}$$

Semantics – Definition of Runtime Error Processes that contain execution errors, such as the $\overline{c}\langle a\ b\rangle \,|\, c\langle x\ y\ z\rangle.P$ of §1.6, can now be identified. We write P err to mean that P contains an execution error, as defined by the rules below.

$$\textsc{Err Com}\ \frac{\{v/p\}P \text{ not defined}}{\overline{c}v \,|\, cp.P \text{ err}} \qquad \textsc{Err Par}\ \frac{P \text{ err}}{P \,|\, Q \text{ err}}$$

$$\textsc{Err Res}\ \frac{P \text{ err}}{\mathbf{new}\ x : T\ \mathbf{in}\ P \text{ err}} \qquad \textsc{Err Struct}\ \frac{P \text{ err} \quad P \equiv P'}{P' \text{ err}}$$

3.2 Typing

As in §1.6, we take *types*

T	$::=$	$\mathbf{chan}\, T$	type of channel names carrying T
		$\langle T_1 \mathinner{..} T_n \rangle$	type of tuples of values of types $T_1 \mathinner{..} T_n$

We take *type environments*, ranged over by Γ and Δ, to be finite partial functions from names to types. We write Γ, Δ for the union of type environments that have disjoint domains. The type system has three judgments:

$\Gamma \vdash v : T$	value v has type T in environment Γ
$\vdash p : T \rhd \Delta$	pattern p matches type T, giving bindings Δ
$\Gamma \vdash P \ \mathbf{proc}$	process P is well-typed in environment Γ

These are defined by the rules below.

Values:

$$\frac{}{\Gamma, x : T \vdash x : T} \qquad \frac{\Gamma \vdash v_1 : T_1 \ \mathinner{..}\ \Gamma \vdash v_k : T_k}{\Gamma \vdash \langle v_1 \mathinner{..} v_k \rangle : \langle T_1 \mathinner{..} T_k \rangle}$$

Patterns:

$$\frac{}{\vdash x : T \rhd x : T} \qquad \frac{\vdash p_1 : T_1 \rhd \Delta_1 \ \mathinner{..}\ \vdash p_k : T_k \rhd \Delta_k}{\vdash \langle p_1 \mathinner{..} p_k \rangle : \langle T_1 \mathinner{..} T_k \rangle \rhd \Delta_1, \mathinner{..}, \Delta_k}$$

Processes:

$$\textsc{Out}\ \frac{\Gamma \vdash x : \mathbf{chan}\, T \quad \Gamma \vdash v : T}{\Gamma \vdash \overline{x} v \ \mathbf{proc}} \qquad \textsc{In}\ \frac{\Gamma \vdash x : \mathbf{chan}\, T \quad \vdash p : T \rhd \Delta \quad \Gamma, \Delta \vdash P \ \mathbf{proc}}{\Gamma \vdash x p . P \ \mathbf{proc}} \qquad \textsc{Par}\ \frac{\Gamma \vdash P \ \mathbf{proc} \quad \Gamma \vdash Q \ \mathbf{proc}}{\Gamma \vdash P \mid Q \ \mathbf{proc}}$$

$$\textsc{Nil}\ \frac{}{\Gamma \vdash 0 \ \mathbf{proc}} \qquad \textsc{Res}\ \frac{\Gamma, x : \mathbf{chan}\, T \vdash P \ \mathbf{proc}}{\Gamma \vdash \mathbf{new}\ x : \mathbf{chan}\ T\ \mathbf{in}\ P \ \mathbf{proc}}$$

The properties of a well-typed process with respect to the reduction semantics are as one would expect. Firstly, well-typing is preserved by reduction.

Theorem 4 (Subject Reduction) *If* $\Gamma \vdash P$ **proc** *and* Γ atomic *and* $P \rightarrow Q$ *then* $\Gamma \vdash Q$ **proc**.

Moreover, a well-typed process cannot contain a runtime error.

Theorem 5 (Absence of Runtime Errors) *If* $\Gamma \vdash P$ **proc** *and* Γ atomic *then* $\neg(P \text{ err})$.

Here our use of arbitrary tuple patterns requires a further constraint – observe that

$$c : \mathbf{chan}\, \langle \langle\rangle\, \langle\rangle \rangle,\ x : \langle \langle\rangle\, \langle\rangle \rangle \vdash \overline{c} x \mid c\langle y_1\ y_2 \rangle . 0 \ \mathbf{proc}$$

holds but the substitution $\{x / \langle y_1\, y_2 \rangle\}$ is not defined. To exclude such free names with non-empty tuple types, we say T atomic iff $T = \langle\rangle$ or $T = \mathbf{chan}\, T'$ for some T', and Γ atomic iff all the types in the range of Γ are atomic.

Note that the (Res) rule allows new-binding only at types of the form $\mathbf{chan}\, T$ – it would be counter-intuitive to allow new values of tuple types to be dynamically generated. In general, we say a type T is *extensible* if it allows new-binding, so for the calculus above T extensible iff $\exists T' \,.\, T = \mathbf{chan}\, T'$. Extending the calculus and type system with base types such as **Int** or **Bool** is straightforward. These are not extensible.

3.3 Typing and Labelled Transitions

The properties of well-typed processes with respect to labelled transition semantics are more delicate. Broadly, one can either define a typed LTS, allowing only well-typed inputs and outputs with the environment, or an untyped LTS, allowing badly-typed inputs and outputs. In either case the definition must support communication of tuples, not just of names. An untyped LTS may be required where one wishes to consider interaction with badly-typed processes, e.g. a malicious attacker. Otherwise, a typed LTS gives a tighter notion of behaviour, definitionally excluding pathological inputs and outputs. Note that for an untyped LTS it is best to take a syntax without type annotations on new-binders, i.e. with **new** x **in** P instead of **new** $x : T$ **in** P, whereas for a typed LTS the converse is preferable.

For either LTS the labels ℓ now allow arbitrary values:

ℓ	$::=$	τ	internal action
		$\overline{x}v$	output of v on x
		xv	input of v on x

Typed LTS The typed labelled transition relation has the form

$$\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$$

which should be contrasted with untyped relation $A \vdash P \xrightarrow{\ell} Q$ of §1.3. The typed transitions are with respect to a type environment Γ for P, not merely a set of free names A that includes those of P – we should expect $\Gamma \vdash P$ **proc**. Moreover, the labelling includes a type environment Δ for the new names extruded or intruded by this transition – we should expect, e.g. if $\ell = \overline{x}v$, that x has a channel type $\mathbf{chan}\, T$ with respect to Γ and that $\Gamma, \Delta \vdash v : T$. For example, there will be transitions

$$x : \mathbf{chan}\,\mathbf{chan}\,\langle\rangle,\ y : \mathbf{chan}\,\langle\rangle \vdash \overline{x}y \xrightarrow[\emptyset]{\overline{x}y} 0$$

$$x : \mathbf{chan}\,\mathbf{chan}\,\langle\rangle \vdash (\mathbf{new}\ y : \mathbf{chan}\,\langle\rangle\ \mathbf{in}\ \overline{x}y) \xrightarrow[y\,:\,\mathbf{chan}\,\langle\rangle]{\overline{x}y} 0$$

For the calculus of §3.1 the typed transitions are defined as the smallest relation satisfying the rules below.

$$\textsc{Out}\ \frac{}{\Gamma \vdash \overline{x}v \xrightarrow[\emptyset]{\overline{x}v} 0}
\qquad
\textsc{In}\ \frac{\begin{array}{l}\Gamma \vdash x : \mathbf{chan}\, T\\ \Gamma, \Delta \vdash v : T\\ \mathrm{dom}(\Delta) \subseteq \mathrm{fn}(v)\\ \Delta \text{ extensible}\end{array}}{\Gamma \vdash xp.P \xrightarrow[\Delta]{xv} \{{}^{v}/_{p}\}P}$$

$$\textsc{Par}\ \frac{\Gamma \vdash P \xrightarrow[\Delta]{\ell} P'}{\Gamma \vdash P \mid Q \xrightarrow[\Delta]{\ell} P' \mid Q}
\qquad
\textsc{Com}\ \frac{\Gamma \vdash P \xrightarrow[\Delta]{\overline{x}v} P' \quad \Gamma \vdash Q \xrightarrow[\Delta]{xv} Q'}{\Gamma \vdash P \mid Q \xrightarrow[\emptyset]{\tau} \mathbf{new}\ \Delta\ \mathbf{in}\ (P' \mid Q')}$$

$$\textsc{Res}\ \frac{\Gamma, x : T \vdash P \xrightarrow[\Delta]{\ell} P' \quad x \notin \mathrm{fn}(\ell)}{\Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ P \xrightarrow[\Delta]{\ell} \mathbf{new}\ x : T\ \mathbf{in}\ P'}
\ \textsc{Open}\ \frac{\Gamma, x : T \vdash P \xrightarrow[\Delta]{\overline{y}v} P' \quad x \in \mathrm{fn}(v) \setminus \{y\}}{\Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ P \xrightarrow[\Delta, x : T]{\overline{y}v} P'}$$

$$\textsc{Struct Right}\ \frac{\Gamma \vdash P \xrightarrow[\Delta]{\ell} P' \quad P' \equiv P''}{\Gamma \vdash P \xrightarrow[\Delta]{\ell} P''}$$

In all rules with conclusion of the form $\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$ there are implicit side conditions Γ atomic and $\Gamma \vdash P$ **proc**. The only other changes are that (COM) involves the free names of v and in (OPEN), which allows x to be extruded if it occurs anywhere in v. Note that in the (IN) axiom the expression $\{^v/_p\}P$ is guaranteed to be well-defined.

Note that (just as in the statement of subject reduction above) we must consider transitions only with respect to atomic type environments to rule out all run-time errors. Note also that the (IN) rule requires Δ to be extensible, so all intruded new names are of channel types. For this calculus T extensible implies T atomic, and we expect this to be true in general – extensible types often have values which are simply names, perhaps not just of channels but also of boxes or agents (c.f. Nomadic and Box π).

Theorem 6 (Subject Reduction – Typed LTS) *If $\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$ then*

1. $\Gamma, \Delta \vdash Q$ **proc**
2. *if $\ell = \overline{x}v$ or $\ell = xv$ then there is T such that $\Gamma \vdash x : \mathbf{chan}\, T$ and $\Gamma, \Delta \vdash v : T$.*

In fact, in this calculus the Δ of a transition $\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$ is uniquely determined – it is either empty for a τ transition or determined by the type of x for an output $\overline{x}v$ or input xv. The semantics can therefore be simplified by omitting all type environment labels and replacing **new** Δ **in** $P' \mid Q'$ in the (COM) rule by **new** $tc(\Gamma, v, T)$ **in** $P' \mid Q'$, where $\Gamma \vdash x : \mathbf{chan}\, T$ and the type environment $tc(\Gamma, v, T)$ is defined as follows.

$$\begin{array}{lcl}
tc(\Gamma, x, T) & = & x : T \quad \text{if } x \notin \text{dom}(\Gamma) \text{ and } T \text{ atomic} \\
tc(\Gamma, x, T) & = & \emptyset \qquad \text{if } \Gamma \vdash x : T \\
tc(\Gamma, \langle v_1 \mathinner{..} v_k \rangle, \langle T_1 \mathinner{..} T_k \rangle) & = & \bigsqcup_{1..k} tc(\Gamma, v_i, T_i) \\
tc(\Gamma, v, T) & & \text{undefined elsewhere}
\end{array}$$

Here $\bigsqcup_{i \in 1..k} \Gamma_i$ is the union of the Γ_i if these agree on all common names (i.e. if for all $x \in \text{dom}(\Gamma_i) \cap \text{dom}(\Gamma_j)$ we have $\Gamma_i(x) = \Gamma_j(x)$) and is undefined otherwise.

In systems with subtyping, where a process may be able to extrude a name at different types, the Δs may be essential. We do not pursue this here.

Untyped LTS The alternative, of untyped transitions, has a simpler definition of the transition relations but more complex statement of subject reduction. For the calculus of §3.1 (but now without type annotations on new-binders) transitions can be defined as the smallest relation satisfying the rules below. We leave the statement of subject reduction as an exercise for the reader.

$$\text{OUT}\ \frac{}{A \vdash \overline{x}v \xrightarrow{\overline{x}v} 0} \qquad\qquad \text{IN}\ \frac{}{A \vdash xp.P \xrightarrow{xv} \{^v/_p\}P}$$

$$\text{PAR}\ \frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q} \qquad \text{COM}\ \frac{A \vdash P \xrightarrow{\overline{x}v} P' \quad A \vdash Q \xrightarrow{xv} Q'}{A \vdash P \mid Q \xrightarrow{\tau} \mathbf{new}\ \text{fn}(v) - A\ \mathbf{in}\ (P' \mid Q')}$$

$$\text{RES}\ \frac{A, x \vdash P \xrightarrow{\ell} P' \quad x \notin \text{fn}(\ell)}{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} \mathbf{new}\ x\ \mathbf{in}\ P'} \qquad \text{OPEN}\ \frac{A, x \vdash P \xrightarrow{\overline{y}v} P' \quad x \neq y \quad x \in \text{fn}(v)}{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\overline{y}v} P'}$$

$$\text{STRUCT RIGHT}\ \frac{A \vdash P \xrightarrow{\ell} P' \quad P' \equiv P''}{A \vdash P \xrightarrow{\ell} P''}$$

In all rules with conclusion of the form $A \vdash P \xrightarrow{\ell} Q$ there is an implicit side condition $\text{fn}(P) \subseteq A$. In the (IN) axiom there is now an implicit side condition that $\{^v/_p\}P$ is well-defined. The only other changes from §1.3 are that (COM) involves the free names of v and in (OPEN), which allows x to be extruded if it occurs anywhere in v.

Chapter 4

Metatheory: Overview

We now develop the basic metatheory for the π-calculi of the previous chapters. We first give basic properties of the labelled transition system, then show the τ-transitions of the labelled transition semantics coincide with the reduction semantics, and show that bisimulation is a congruence. We go on to show subject reduction and type-soundness results for the calculus and simple type system of Chapter 3.

The aim is to provide a model development of this theory for simple π-calculi, so that it can be understood clearly and generalized as required. We do not discuss variations of calculus or semantics in any detail – for that, the reader should refer to the literature cited earlier, and to the forthcoming monograph of Sangiorgi and Walker. In particular, we work mostly with a monadic calculus without choice, replication, or matching, and use only explicitly-indexed early labelled transition semantics.

The exposition is divided into two parts. The first gives an overview, with statements of lemmas, sketch proofs and discussion. The second, in the following chapter, repeats the statements and gives full proofs.

4.1 Basic Properties of the LTS

Structural congruence preserves free name sets and is closed under substitutions.

Lemma 7 *If* $P \equiv Q$ *then* $\mathrm{fn}(P) = \mathrm{fn}(Q)$.

Lemma 8 *If* $P \equiv Q$ *then for any substitution* f *we have* $fP \equiv fQ$.

Lemma 9 (Naming) *If* $A \vdash P \xrightarrow{\ell} Q$ *then*

1. $\mathrm{fn}(P) \subseteq A$ *and* $\mathrm{fn}(Q) \subseteq \mathrm{fn}(P, \ell)$
2. *if* $\ell = \overline{x}v$ *then* $x \in \mathrm{fn}(P)$ *and* $\mathrm{fn}(\ell) \cap A \subseteq \mathrm{fn}(P)$
3. *if* $\ell = xv$ *then* $x \in \mathrm{fn}(P)$.

Proof Sketch By induction on the derivation of $A \vdash P \xrightarrow{\ell} Q$. $\square$

The calculus has no mismatching – it does not allow inequality testing of names – so transitions are preserved by arbitrary substitutions as follows. Note that only *injective* substitution, as stated in Lemma 11, is required for the correspondence between reduction and labelled transition semantics (all of the section leading to Theorem 2), and for the congruence of bisimulation with respect to parallel and new (Lemma 31).

Lemma 10 (Substitution) *If $A \vdash P \xrightarrow{\ell} Q$, $f : A \to B$, and $g : (\mathrm{fn}(\ell) - A) \to \mathcal{N}$, and further if ℓ is an output then g is injective and $B \cap \mathrm{ran}(g) = \emptyset$, then $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)Q$.*

Proof Sketch Induction on derivations of transitions. □

Lemma 11 (Injective Substitution) *If $A \vdash P \xrightarrow{\ell} P'$, and $f : A \to B$ and $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - B)$ are injective, then $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)P'$.*

Proof Sketch One can either prove this as a corollary of Lemma 10 or give a direct proof (by induction on derivations of transitions) if in a calculus where Lemma 10 does not hold. □

Lemma 12 (Shifting) *$(A \vdash P \xrightarrow{zv} P' \wedge v \notin A)$ iff $(A, v \vdash P \xrightarrow{zv} P' \wedge v \notin \mathrm{fn}(P))$.*

Proof Sketch By two inductions on derivations of transitions. □

The transitions of an injectively-substituted process fP are determined by the transitions of P as follows. This Lemma is used to show that bisimulation is preserved by injective substitution (Lemma 30). It is not required for the correspondence between reduction and labelled transition semantics (all of the section leading to Theorem 2), although the corollary Lemma 14 (Strengthening) is.

Lemma 13 (Converse to Injective Substitution) *For $f : A \to B$ injective, if $B \vdash fP \xrightarrow{\ell'} Q'$ then (at least) one of the following two cases applies*

1. *there exist ℓ, Q, $g : (\mathrm{fn}(\ell) - A) \to_{\mathrm{bij}} \hat{B}$ such that $\hat{B} \cap B = \emptyset$ and $\ell' = (f+g)(\ell)$ and $A \vdash P \xrightarrow{\ell} Q$ and $Q' = (f+g)Q$.*
2. *there exist $x \in A$, $y \notin A$, $z \in B - \mathrm{ran}(f)$ and Q such that $\ell' = f(x)z$ and $A \vdash P \xrightarrow{xy} Q$ and $Q' = (f + \{z/y\})Q$.*

Proof Sketch Induction on derivations of transitions, using Lemmas 8, 9 and 11. □

Lemma 14 (Strengthening) *If $A, B \vdash P \xrightarrow{\ell} P'$ and $B \cap \mathrm{fn}(P, \ell) = \emptyset$ then $A \vdash P \xrightarrow{\ell} P'$.*

Proof Sketch This is a simple corollary of Lemmas 13 and 11. We also give a direct proof by induction on derivations of transitions. □

Lemma 15 (Weakening and Strengthening) *$(A \vdash P \xrightarrow{\ell} P' \wedge x \notin A \cup \mathrm{fn}(\ell))$ iff $(A, x \vdash P \xrightarrow{\ell} P' \wedge x \notin \mathrm{fn}(P, \ell))$.*

PROOF SKETCH This is a corollary of Lemmas 14 and 11. □

As we are working up to alpha conversion some care is required when analysing transitions. We need the following lemma (of which only the input and restriction cases are really interesting).

Lemma 16 (Transition Analysis)

1. *$A \vdash \overline{x}v \xrightarrow{\ell} Q$ iff $\mathrm{fn}(\overline{x}v) \subseteq A$, $\ell = \overline{x}v$ and $Q \equiv 0$.*
2. *$A \vdash xp.P \xrightarrow{\ell} Q$ iff there exists v such that $\mathrm{fn}(xp.P) \subseteq A$, $\ell = xv$, and $Q \equiv \{^v/_p\}P$.*
3. *$A \vdash P \mid Q \xrightarrow{\ell} R$ iff either*
   (a) *there exists $\hat{P}$ such that $\mathrm{fn}(Q) \subseteq A$, $A \vdash P \xrightarrow{\ell} \hat{P}$ and $R \equiv \hat{P} \mid Q$.*
   (b) *there exist x, v, $\hat{P}$ and $\hat{Q}$ such that $\ell = \tau$, $A \vdash P \xrightarrow{\overline{x}v} \hat{P}$, $A \vdash Q \xrightarrow{xv} \hat{Q}$, and $R \equiv \textbf{new}\ \{v\} - A\ \textbf{in}\ (\hat{P} \mid \hat{Q})$.*

   or symmetric cases.
4. *$A \vdash \textbf{new}\ x\ \textbf{in}\ P \xrightarrow{\ell} Q$ iff either*
   (a) *there exist $\hat{x} \notin A \cup \mathrm{fn}(\ell) \cup (\mathrm{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{^{\hat{x}}/_x\}P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \textbf{new}\ \hat{x}\ \textbf{in}\ \hat{Q}$.*
   (b) *there exist y, $\hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{^{\hat{x}}/_x\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q \equiv \hat{Q}$.*

PROOF SKETCH The right-to-left implications are all shown using a single transition rule together with (TRANS STRUCT RIGHT). The left-to-right implications are shown by induction on derivations of transitions. □

Note that the alpha conversion in the restriction case 4 is essential. To see why, we give examples to show why $A \vdash \textbf{new}\ x\ \textbf{in}\ P \xrightarrow{\ell} Q$ does not imply that one of the following hold.

1. there exists $\hat{Q}$ such that $A, x \vdash P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \textbf{new}\ x\ \textbf{in}\ \hat{Q}$.
2. there exists y and $\hat{Q}$ such that $\ell = \overline{y}x$, $A, x \vdash P \xrightarrow{\overline{y}x} \hat{Q}$, $y \neq x$, and $\hat{Q} \equiv Q$.

For the first, there is a transition

$$\{x, y, z\} \vdash \textbf{new}\ x\ \textbf{in}\ \overline{y}z \xrightarrow{\overline{y}z} \textbf{new}\ x\ \textbf{in}\ 0$$

but $\{x, y, z\}, x$ is not well-formed. For the second, if $y \in A$ then there is a transition

$$A \vdash \textbf{new}\ x\ \textbf{in}\ \overline{y}x \xrightarrow{\overline{y}z} 0$$

for any $z \notin A$. Here we may have $x \in A$ or $x \notin A$. In constrast, alpha conversion in the input case 2 is not essential.

It will be convenient to have more particular results for analysing transitions $A \vdash \textbf{new}\ B\ \textbf{in}\ P \xrightarrow{\ell} Q$, under the assumptions $B \cap A = \emptyset$ and $B \cap (A \cup \mathrm{fn}(\ell)) = \emptyset$. These rename the label and resulting process as required, in contrast to Lemma 16 which keeps the label fixed but renames the source process. For completeness we also give results making only the first assumption.

Lemma 17 (Transition Analysis – Unary Disjoint New) *If* $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ *and* $x \notin A \cup \mathrm{fn}(\ell)$ *then either*

1. *there exists* Q' *such that* $A, x \vdash P \xrightarrow{\ell} Q'$ *and* $Q \equiv \mathbf{new}\ x\ \mathbf{in}\ Q'$
2. *there exist* $y \in A$, $\hat{x} \notin A$ *and* Q' *such that* $\ell = \overline{y}\hat{x}$, $A, x \vdash P \xrightarrow{\overline{y}x} Q'$, *and* $Q \equiv \{\hat{x}/x\}Q'$.

Proof Sketch The transition can be analysed by Lemma 16, then Lemma 11 (injective substitution) and Lemma 9 used. □

Lemma 18 (Transition Analysis – n-ary Disjoint New) *If* $A \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$, $A \cap B = \emptyset$ *and* $\mathrm{fn}(\ell) \cap B = \emptyset$ *then either*

1. *there exists* Q' *such that* $A, B \vdash P \xrightarrow{\ell} Q'$ *and* $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ Q'$
2. *there exist* $y \in A$, $\hat{x} \notin A$, $x \in B$ *and* Q' *such that* $\ell = \overline{y}\hat{x}$, $A, B \vdash P \xrightarrow{\overline{y}x} Q'$, *and* $Q \equiv \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q'$.

Proof Sketch Induction on the size of B using Lemma 17. □

The following two lemmas, which assume the restricted names are disjoint from the name context A but may occur in the label, are included for completeness – they are not used later.

Lemma 19 (Transition Analysis – Unary A-Disjoint New) *If* $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ *and* $x \notin A$ *then either*

1. *For any* $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, x$ *there exists* $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, x, C)$ *injective and* Q' *such that* $A, x \vdash P \xrightarrow{(id_A + g)\ell} Q'$ *and* $Q \equiv (id_A + g^{-1})\mathbf{new}\ x\ \mathbf{in}\ Q'$.
2. *There exists* $y \in A$, $\hat{x} \notin A$ *and* Q' *such that* $\ell = \overline{y}\hat{x}$, $A, x \vdash P \xrightarrow{\overline{y}x} Q'$ *and* $Q \equiv \{{}^{\hat{x}}/_{x}\}Q'$.

Proof Sketch The transition can be analysed by Lemma 16, then injective substitution (Lemma 11) used. □

Lemma 20 (Transition Analysis – n-ary A-Disjoint New) *If* $A \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$ *and* $A \cap B = \emptyset$ *then either*

1. *there exist* $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, B)$ *injective and* Q' *such that* $A, B \vdash P \xrightarrow{(id_A + g)\ell} Q'$ *and* $Q \equiv (id_A + g^{-1})\mathbf{new}\ B\ \mathbf{in}\ Q'$.
2. *there exist* $y \in A$, $\hat{x} \notin A$, $x \in B$ *and* Q' *such that* $\ell = \overline{y}\hat{x}$, $A, B \vdash P \xrightarrow{\overline{y}x} Q'$, *and* $Q \equiv \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q'$.

Proof Sketch We prove a slightly stronger result, with $\mathrm{ran}(g)$ disjoint from an arbitrary $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, B$, by induction on the size of B using Lemma 19. □

4.2 Coincidence of the Two Semantics

Reductions Imply Transitions This direction of the equivalence has two main parts: we must show that transitions are invariant under structural congruence and construct τ-transitions for each reduction axiom.

Take the *size* of a derivation of a structural congruence to be number of instances of inference rules contained in it.

Lemma 21 *For any derivation of $P' \equiv P$ there is a derivation of the same size of $\{^v/_p\}P' \equiv \{^v/_p\}P$.*

The following lemma is stated as Theorem 1 in §1.

Lemma 22 *If $P' \equiv P$ then $A \vdash P' \xrightarrow{\ell} Q$ iff $A \vdash P \xrightarrow{\ell} Q$.*

Proof Sketch Induction on the size of derivation of $P' \equiv P$, with case analysis using the transition analysis Lemmas 16 and 17 for the axioms. □

Lemma 23 *If $\mathrm{fn}(P) \subseteq A$ and $P \rightarrow Q$ then $A \vdash P \xrightarrow{\tau} Q$.*

Proof Sketch Induction on derivations of $P \rightarrow Q$, constructing derivations of τ-transitions for the reduction axiom (Com), and using Lemma 22 for the (Struct) case. □

Transitions Imply Reductions For the converse direction we first show that if a process has an output or input transition then it contains a corresponding output or input.

Lemma 24 (Term Structure – Output Transition) *If $A \vdash P \xrightarrow{\overline{z}v} P'$ then we have $P \equiv \textbf{new } \{v\} - A \textbf{ in } (\overline{z}v \mid P')$*

Lemma 25 (Term Structure – Input Transition) *If $A \vdash Q \xrightarrow{xv} Q'$ then there exist B, p, Q_1 and Q_2 such that $B \cap (A \cup \mathrm{fn}(xv)) = \{\}$, $Q \equiv \textbf{new } B \textbf{ in } (xp.Q_1 \mid Q_2)$ and $Q' \equiv \textbf{new } B \textbf{ in } (\{^v/_p\}Q_1 \mid Q_2)$.*

Lemma 26 *If $A \vdash P \xrightarrow{\tau} Q$ then $P \rightarrow Q$.*

Proof Sketch Induction on derivations of $A \vdash P \xrightarrow{\tau} Q$, using the preceding Lemmas 24 and 25 for the (Com) case. □

We also show the following lemma here, giving the term structure of a τ-transition, for use in Lemma 33.

Lemma 27 (Term Structure – Tau Transition) *If $A \vdash P \xrightarrow{\tau} Q$ and $B \subseteq_{\mathrm{fin}} \mathcal{N}$ then there exist C, x, v, p, Q_1, Q_2 such that $C \cap (A \cup B) = \emptyset$, $p \notin A \cup B \cup C$, $P \equiv \textbf{new } C \textbf{ in } (\overline{x}v \mid xp.Q_1 \mid Q_2)$ and $Q \equiv \textbf{new } C \textbf{ in } (\{^v/_p\}Q_1 \mid Q_2)$.*

Proof By Lemma 26 $P \longrightarrow Q$. One can show the result for empty B by induction on derivations of reductions, and then use alpha-renaming. □

Theorem 2 *If* $\mathrm{fn}(P) \subseteq A$ *then* $P \longrightarrow Q$ *iff* $A \vdash P \xrightarrow{\tau} Q$.

Proof This is immediate from Lemmas 23 and 26 above. □

4.3 Strong Bisimulation and Congruence

We first give a rigorous definition of bisimulation as a greatest fixed point. Say an *indexed relation* $\mathcal{R}$ is a family of relations $\mathcal{R}_A$ indexed by finite sets of names with each $\mathcal{R}_A$ a binary relation over $\{ P \mid \mathrm{fn}(P) \subseteq A \}$. The indexed relations are clearly closed under pointwise unions. Define a endofunction Φ over indexed relations by $P \; \Phi(\mathcal{R})_A \; Q$ if

- $\mathrm{fn}(P, Q) \subseteq A$,
- if $A \vdash P \xrightarrow{\ell} P'$ then $\exists Q' \,.\, A \vdash Q \xrightarrow{\ell} Q' \wedge P' \; \mathcal{R}_{A \cup \mathrm{fn}(\ell)} \; Q'$, and
- if $A \vdash Q \xrightarrow{\ell} Q'$ then $\exists P' \,.\, A \vdash P \xrightarrow{\ell} P' \wedge P' \; \mathcal{R}_{A \cup \mathrm{fn}(\ell)} \; Q'$.

Say an indexed relation $\mathcal{R}$ is a *bisimulation* if $\mathcal{R} \subseteq \Phi(\mathcal{R})$.

Take *bisimulation* $\dot{\sim}$ to be $\bigcup \{ \mathcal{R} \mid \mathcal{R} \subseteq \Phi(\mathcal{R}) \}$. As usual, this is a fixed point and an equivalence relation:

Lemma 28 $\Phi(\dot{\sim}) = \dot{\sim}$ *and* $\dot{\sim}$ *is an equivalence.*

Proof Sketch Standard reasoning, albeit here for indexed relations. □

The rest of this section is devoted to showing that $\dot{\sim}$ is an indexed congruence. We begin by proving the soundness of an 'up-to' proof technique. We then show that bisimulation is preserved by injective substitution, and hence by weakening. Both of these results are used in a proof that bisimulation is preserved by parallel and new contexts. That result in turn is used in a proof that bisimulation is preserved by arbitrary substitutions, and hence by input-prefix contexts. This depends critically on the asynchrony of the calculus, as manifested in Lemma 32. In a richer calculus, e.g. with output prefixing, one would expect to have to close $\dot{\sim}$ up under substitution to obtain a congruence.

If R is a relation over process terms such that $P \; R \; Q \implies \mathrm{fn}(P) = \mathrm{fn}(Q)$, for example $\equiv$, then we can regard R as an indexed relation with $P \; R_A \; Q$ iff $\mathrm{fn}(P, Q) \subseteq A \wedge P \; R \; Q$.

To check bisimulation we need consider extrusion and intrusion only of some new names. Say an indexed relation $\mathcal{R}$ is a *loose bisimulation up to structural congruence and bijective renaming* (or just *loose bisimulation* for short) if whenever $P \; \mathcal{R}_A \; Q$ there exists $D \subseteq_{\mathrm{fin}} (\mathcal{N} - A)$ such that

- $\mathrm{fn}(P, Q) \subseteq A$,
- if $A \vdash P \xrightarrow{\ell} P'$ and $\mathrm{fn}(\ell) \cap D = \emptyset$ then there exists Q', C, and a bijection $f : A \cup \mathrm{fn}(\ell) \to C$ such that $A \vdash Q \xrightarrow{\ell} Q'$ and $fP' \equiv \mathcal{R}_C \equiv fQ'$, and
- if $A \vdash Q \xrightarrow{\ell} Q'$ then the symmetric condition holds.

Lemma 29 *If $\mathcal{R}$ is a loose bisimulation then $\mathcal{R} \subseteq \dot{\sim}$.*

PROOF SKETCH We check $\mathcal{S}_A \stackrel{def}{=} \{ f^{-1}P, f^{-1}Q \mid f : A \rightarrow_{\mathrm{bij}} B \wedge P \equiv\mathcal{R}_B\equiv Q \}$ is a bisimulation. □

Lemma 30 (Injective Substitution – Bisimulation) *If $P \dot{\sim}_A Q$ and $f : A \rightarrow B$ is injective then $fP \dot{\sim}_B fQ$.*

PROOF SKETCH By checking $\mathcal{R}_B = \{ fP_1, fP_2 \mid f : A \rightarrow_{\mathrm{inj}} B \wedge P_1 \dot{\sim}_A P_2 \}$ is a bisimulation, using Lemma 13 to go from transitions of fP_1 to transitions of P_1, and then using Lemmas 11 and 12. □

Lemma 31 (Congruence – Par and New) *If $P \dot{\sim}_{A,B} P'$ and $Q \dot{\sim}_{A,B} Q'$ then* **new** B **in** $(P \mid Q) \dot{\sim}_A$ **new** B **in** $(P' \mid Q')$.

PROOF SKETCH By checking $\mathcal{R}_A = \{$ **new** B **in** $(P \mid Q)$, **new** B **in** $(P' \mid Q') \mid P \dot{\sim}_{A,B} P' \wedge Q \dot{\sim}_{A,B} Q' \}$ is a loose bisimulation, using Lemmas 18 and 16 to analyse transitions of **new** B **in** $(P \mid Q)$, and also using Lemmas 30 and 11. □

Lemma 32 (Asynchrony) *If $A \vdash P \xrightarrow{\overline{z}v} Q \xrightarrow{zv} R$ then $A \vdash P \xrightarrow{\tau}$* **new** $\{v\} - A$ **in** R.

PROOF SKETCH By Lemmas 24 and 25. □

Lemma 33 (Substitution – Bisimulation) *If $P \dot{\sim}_A Q$ and $\sigma : A \rightarrow B$ then $\sigma P \dot{\sim}_B \sigma Q$.*

PROOF SKETCH We check $\mathcal{R}_B = \{ \sigma P, \sigma P' \mid \exists A \,.\, \sigma : A \rightarrow B \wedge P \dot{\sim}_A P' \}$ is a bisimulation up to $\equiv$. We use Lemmas 24, 25 and 27 to analyse transitions of σP (one could alternatively develop an explicit converse to the substitution lemma for transitions). The interesting case is that in which σ creates a τ-transition by identifying two names. Here the asynchrony property stated in Lemma 32 is crucial, and in one subcase the new-congruence part of Lemma 31 is also required. □

Lemma 34 (Congruence – Input Prefix) *If $P \dot{\sim}_{A,p} Q$ and $x \in A$ then $xp.P \dot{\sim}_A xp.Q$.*

PROOF SKETCH We check the union of $\dot{\sim}$ and the pair $(xp.P,\ xp.Q)$ at A is a bisimulation up to $\equiv$. □

Theorem 3 *Bisimulation $\dot{\sim}$ is an indexed congruence.*

PROOF Immediate from Lemmas 31 and 34. □

4.4 Type Soundness and Subject Reduction

We now prove the type soundness results stated in Chapter 3, namely subject reduction (for the reduction semantics), absence of runtime errors, and subject reduction for the typed labelled transition semantics. These results are for the calculus with tuples defined in that chapter. We do not develop analogues of the reduction/LTS and bisimulation congruence results for this calculus – they would be fairly straightforward adaptations of the results of §4.1–4.3. For simplicity we have chosen to work with atomic type environments everywhere.

The first 4 lemmas, for weakening and strengthening of typing judgements, are all proved by routine inductions on type derivations.

Lemma 35 (Weakening – Values) *If* $\Gamma \vdash v : T'$ *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, x : T \vdash v : T'$.

Lemma 36 (Weakening – Processes) *If* $\Gamma \vdash P$ **proc** *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, x : T \vdash P$ **proc**.

Lemma 37 (Strengthening – Values) *If* $\Gamma, x : T \vdash v : T'$ *and* $x \notin \mathrm{fn}(v)$ *then* $\Gamma \vdash v : T'$.

Lemma 38 (Strengthening – Processes) *If* $\Gamma, x : T \vdash P$ **proc** *and* $x \notin \mathrm{fn}(P)$ *then* $\Gamma \vdash P$ **proc**.

Lemma 39 (Type Soundness of Structural Congruence) *If* $P \equiv Q$ *then* $\Gamma \vdash P$ **proc** *iff* $\Gamma \vdash Q$ **proc**.

Proof Sketch By induction on the derivation of $P \equiv Q$, using Lemmas 36,38 for the scope extrusion axiom. □

Lemma 40 (Injective Substitution – Reductions) *If* $P \to Q$ *and* $f : \mathrm{fn}(P) \to \mathcal{N}$ *is injective then* $fP \to fQ$.

Proof Sketch This can be proved either directly or as a corollary of the results for labelled transitions (Theorem 2 and Lemma 11), if those are adapted for the calculus with tuples. □

We define well-typed substitutions as follows. Say $\Gamma \vdash \sigma : \Delta$ iff $\mathrm{dom}(\sigma) = \mathrm{dom}(\Delta)$ and $\forall x \in \mathrm{dom}(\Delta) \,.\, \Gamma \vdash \sigma(x) : \Delta(x)$.

Lemma 41 (Good Substitutions) *If* Γ atomic *and* $\Gamma \vdash v : T$ *and* $\vdash p : T \triangleright \Delta$ *then* $\{{}^{v}/_{p}\}$ *is defined and* $\Gamma \vdash \{{}^{v}/_{p}\} : \Delta$.

Proof Sketch By induction on the two typing derivations. □

Lemma 42 (Substitution – Values) *If* $\Gamma, \Delta \vdash v : T$ *and* $\Gamma \vdash \sigma : \Delta$ *then* $\Gamma \vdash \sigma v : T$.

Proof Sketch By induction on the value typing derivation. □

Lemma 43 (Substitution – Processes) *If* $\Gamma, \Delta \vdash P$ **proc** *and* $\Gamma \vdash \sigma : \Delta$ *then* σP *is defined and* $\Gamma \vdash \sigma P$ **proc**.

Proof Sketch By induction on the type derivation for P, using Lemma 42. □

Theorem 4 (Subject Reduction) *If* $\Gamma \vdash P$ **proc** *and* Γ atomic *and* $P \to Q$ *then* $\Gamma \vdash Q$ **proc**.

Proof Sketch By induction on the derivation of $P \to Q$, using Lemmas 41, 43 and Lemma 39 in the (Com) and (Struct) cases. □

Theorem 5 (Absence of Runtime Errors) *If* $\Gamma \vdash P$ **proc** *and* Γ atomic *then* $\neg(P \text{ err})$.

Proof Sketch By induction on derivations of P err. □

Theorem 6 (Subject Reduction – Typed LTS) *If* $\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$ *then*

1. $\Gamma, \Delta \vdash Q$ **proc**
2. *if* $\ell = \overline{x}v$ *or* $\ell = xv$ *then there is* T *such that* $\Gamma \vdash x : \mathbf{chan}\, T$ *and* $\Gamma, \Delta \vdash v : T$.

Proof Sketch By induction on derivations of transitions, using Lemmas 41, 36 and 43 in the (In) case, Lemma 36 in the (Par) case, Lemma 35 in the (Res) and (Open) cases, and Lemma 39 in the (Struct) case. □

Chapter 5

Metatheory: Detailed Proofs

5.1 Basic Properties of the LTS

Lemma 7 *If $P \equiv Q$ then* $\mathrm{fn}(P) = \mathrm{fn}(Q)$*.*

PROOF Routine induction on derivation of $P \equiv Q$. □

Lemma 8 *If $P \equiv Q$ then for any substitution f we have $fP \equiv fQ$.*

PROOF Routine induction on derivation of $P \equiv Q$. □

Lemma 9 (Naming) *If $A \vdash P \xrightarrow{\ell} Q$ then*

1. $\mathrm{fn}(P) \subseteq A$ *and* $\mathrm{fn}(Q) \subseteq \mathrm{fn}(P, \ell)$
2. *if* $\ell = \overline{x}v$ *then* $x \in \mathrm{fn}(P)$ *and* $\mathrm{fn}(\ell) \cap A \subseteq \mathrm{fn}(P)$
3. *if* $\ell = xv$ *then* $x \in \mathrm{fn}(P)$*.*

PROOF By induction on the derivation of $A \vdash P \xrightarrow{\ell} Q$. The first clause of Part 1 is immediate in all cases by the implicit condition on the transition rules. For the other parts:

(OUT) By the condition $\mathrm{fn}(\overline{x}v) \subseteq A$.

(IN) For Part 1, $\mathrm{fn}(\{^v/_p\}P) \subseteq (\mathrm{fn}(P) - \{p\}) \cup \{v\} \subseteq \mathrm{fn}(xp.P) \cup \mathrm{fn}(xv)$. For Part 3, $x \in \mathrm{fn}(xp.P)$.

(PAR) By the induction hypothesis.

(COM) Part 1 is by parts 1, 2 and 3 of the induction hypothesis.

(RES) By the induction hypothesis.

(OPEN) For Part 1, by Part 1 of the induction hypothesis $\mathrm{fn}(P') \subseteq \mathrm{fn}(P) \cup \mathrm{fn}(\overline{y}x)$. As $x \in \mathrm{fn}(\overline{y}x)$ we have $\mathrm{fn}(P') \subseteq \mathrm{fn}(\textbf{new}\ x\ \textbf{in}\ P) \cup \mathrm{fn}(\overline{y}x)$. For the first clause of Part 2, by the induction hypothesis $y \in \mathrm{fn}(P)$ so by the side condition $y \neq x$ we have $y \in \mathrm{fn}(\textbf{new}\ x\ \textbf{in}\ P)$. For the second clause, by the induction hypothesis $\mathrm{fn}(\overline{y}x) \cap (A, x) \subseteq \mathrm{fn}(P)$ so $\mathrm{fn}(\overline{y}x) \cap A \subseteq \mathrm{fn}(\textbf{new}\ x\ \textbf{in}\ P)$.

(Struct Right) By the induction hypothesis and Lemma 7.

□

Lemma 10 (Substitution) *If $A \vdash P \xrightarrow{\ell} Q$, $f : A \to B$, and $g : (\mathrm{fn}(\ell) - A) \to \mathcal{N}$, and further if ℓ is an output then g is injective and $B \cap \mathrm{ran}(g) = \emptyset$, then $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)Q$.*

Proof Induction on derivations of transitions.

(Out) immediate.

(Par),(Struct Right) Straightforward uses of the induction hypothesis.

(In) Consider $A \vdash xp.P \xrightarrow{xv} \{^v/_p\}P$. We have $\mathrm{fn}(xp.P) \subseteq A$. Take some $\hat{p}$ and $\hat{P}$ such that $xp.P = x\hat{p}.\hat{P}$ and $\hat{p} \notin (A \cup B \cup (\mathrm{fn}(\ell) - A) \cup \mathrm{ran}(g))$, then $f(xp.P) = f(x\hat{p}.\hat{P}) = f(x)\hat{p}.f(\hat{P})$ and $\mathrm{fn}(f(x)\hat{p}.f(\hat{P})) \subseteq B$.

By (In) $B \vdash f(x)\hat{p}.f(\hat{P}) \xrightarrow{f(x)(f+g)v} \{^{(f+g)v}/_{\hat{p}}\}f(\hat{P})$.

Now $\mathrm{fn}(\hat{P}) \subseteq A, \hat{p}$ so $\mathrm{fn}(\hat{P}) \cap \mathrm{dom}(g) = \emptyset$, so $f\hat{P} = (f+g)\hat{P}$. Hence $\{^{(f+g)v}/_{\hat{p}}\}f\hat{P} = \{^{(f+g)v}/_{\hat{p}}\}(f+g)\hat{P} = (f+g)(\{^v/_{\hat{p}}\}\hat{P}) = (f+g)(\{^v/_p\}P)$, so we have the transition $B \vdash f(xp.P) \xrightarrow{(f+g)xv} (f+g)(\{^v/_p\}P)$.

(Com) $\mathrm{fn}(\tau) = \emptyset$, so we have $f : A \to B$ and $g : \emptyset \to \mathcal{N}$. Take some $\hat{g} : (\mathrm{fn}(\overline{x}v) - A) \to (\mathcal{N} - B)$ injective. By the induction hypothesis and (Com) we have

$$\textsc{Com}\ \frac{B \vdash fP \xrightarrow{(f+\hat{g})(\overline{x}v)} (f+\hat{g})P' \quad B \vdash fQ \xrightarrow{(f+\hat{g})(xv)} (f+\hat{g})Q'}{B \vdash f(P \mid Q) \xrightarrow{\tau} \mathbf{new}\ \{(f+\hat{g})v\} - B\ \mathbf{in}\ ((f+\hat{g})(P' \mid Q'))}$$

Now $\{(f+\hat{g})v\} - B = \mathrm{ran}(\hat{g})$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} \mathbf{new}\ \mathrm{ran}(\hat{g})\ \mathbf{in}\ ((f+\hat{g})(P' \mid Q'))$. We have $f(\mathbf{new}\ \mathrm{dom}(\hat{g})\ \mathbf{in}\ (P' \mid Q')) = (\mathbf{new}\ \mathrm{ran}(\hat{g})\ \mathbf{in}\ (f+\hat{g})(P' \mid Q'))$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} f(\mathbf{new}\ \{v\} - A\ \mathbf{in}\ (P' \mid Q'))$.

(Res) Take some $\hat{x} \notin B \cup \mathrm{ran}(g)$ and define $\hat{f} : (A, x) \to (B, \hat{x})$ by

$$\begin{array}{rcl} \hat{f}(x) & = & \hat{x} \\ \hat{f}(z) & = & f(z), \text{ for } z \in A. \end{array}$$

By the induction hypothesis $B, \hat{x} \vdash \hat{f}P \xrightarrow{(\hat{f}+g)\ell} (\hat{f}+g)P'$. By (Res) we have $B \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{f}P \xrightarrow{(\hat{f}+g)\ell} \mathbf{new}\ \hat{x}\ \mathbf{in}\ (\hat{f}+g)P'$, so $B \vdash f(\mathbf{new}\ x\ \mathbf{in}\ P) \xrightarrow{(f+g)\ell} (f+g)\mathbf{new}\ x\ \mathbf{in}\ P'$.

(Open) Define $\hat{f} : (A, x) \to (B, g(x))$ and $\hat{g}$ as $f + (x \mapsto g(x))$ and $g \restriction (\mathrm{fn}(\overline{y}x) - (A, x)) = \emptyset$ respectively. By the induction hypothesis $B, g(x) \vdash \hat{f}P \xrightarrow{(\hat{f}+\hat{g})\overline{y}x} (\hat{f}+\hat{g})P'$, so by (Open) $B \vdash \mathbf{new}\ g(x)\ \mathbf{in}\ \hat{f}P \xrightarrow{(\hat{f}+\hat{g})\overline{y}x} (\hat{f}+\hat{g})P'$, so as $f + g = \hat{f} + \hat{g}$ we have $B \vdash f(\mathbf{new}\ x\ \mathbf{in}\ P) \xrightarrow{(f+g)\overline{y}x} (f+g)P'$.

□

Lemma 11 (Injective Substitution) *If $A \vdash P \xrightarrow{\ell} P'$, and $f : A \to B$ and $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - B)$ are injective, then $B \vdash fP \xrightarrow{(f+g)\ell} (f+g)P'$.*

Proof This is a simple corollary of Lemma 10, but we also give a direct proof by induction on derivations of transitions.

(Out) immediate.

(Par),(Struct Right) Straightforward uses of the induction hypothesis.

(In) Consider $A \vdash xp.P \xrightarrow{xv} \{^v/_p\}P$. We have $\mathrm{fn}(xp.P) \subseteq A$. Take some $\hat{p}$ and $\hat{P}$ such that $xp.P = x\hat{p}.\hat{P}$ and $\hat{p} \notin (A \cup B \cup (\mathrm{fn}(\ell) - A) \cup \mathrm{ran}(g))$, then $f(xp.P) = f(x\hat{p}.\hat{P}) = f(x)\hat{p}.f(\hat{P})$ and $\mathrm{fn}(f(x)\hat{p}.f(\hat{P})) \subseteq B$.

By (In) $B \vdash f(x)\hat{p}.f(\hat{P}) \xrightarrow{f(x)(f+g)v} \{^{(f+g)v}/_{\hat{p}}\}f(\hat{P})$.

Now $\mathrm{fn}(\hat{P}) \subseteq A, \hat{p}$ so $\mathrm{fn}(\hat{P}) \cap \mathrm{dom}(g) = \emptyset$, so $f\hat{P} = (f+g)\hat{P}$. Hence $\{^{(f+g)v}/_{\hat{p}}\}f\hat{P} = \{^{(f+g)v}/_{\hat{p}}\}(f+g)\hat{P} = (f+g)(\{^v/_{\hat{p}}\}\hat{P}) = (f+g)(\{^v/_p\}P)$, so we have the transition $B \vdash f(xp.P) \xrightarrow{(f+g)xv} (f+g)(\{^v/_p\}P)$.

(Com) $\mathrm{fn}(\tau) = \emptyset$, so we have $f : A \to B$ and $g : \emptyset \to (\mathcal{N} - B)$. Take some $\hat{g} : (\mathrm{fn}(\overline{x}v) - A) \to (\mathcal{N} - B)$ injective. By the induction hypothesis and (Com) we have

$$\text{Com}\ \frac{B \vdash fP \xrightarrow{(f+\hat{g})(\overline{x}v)} (f+\hat{g})P' \quad B \vdash fQ \xrightarrow{(f+\hat{g})(xv)} (f+\hat{g})Q'}{B \vdash f(P \mid Q) \xrightarrow{\tau} \mathbf{new}\ \{(f+\hat{g})v\} - B\ \mathbf{in}\ ((f+\hat{g})(P' \mid Q'))}$$

Now $\{(f+\hat{g})v\} - B = \mathrm{ran}(\hat{g})$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} \mathbf{new}\ \mathrm{ran}(\hat{g})\ \mathbf{in}\ ((f+\hat{g})(P' \mid Q'))$. We have $f(\mathbf{new}\ \mathrm{dom}(\hat{g})\ \mathbf{in}\ (P' \mid Q')) = (\mathbf{new}\ \mathrm{ran}(\hat{g})\ \mathbf{in}\ (f+\hat{g})(P' \mid Q'))$, so $B \vdash f(P \mid Q) \xrightarrow{\tau} f(\mathbf{new}\ \{v\} - A\ \mathbf{in}\ (P' \mid Q'))$.

(Res) Take some $\hat{x} \notin B \cup \mathrm{ran}(g)$ and define $\hat{f} : (A, x) \to (B, \hat{x})$ by

$$\begin{array}{rcl} \hat{f}(x) & = & \hat{x} \\ \hat{f}(z) & = & f(z), \text{ for } z \in A. \end{array}$$

By the induction hypothesis $B, \hat{x} \vdash \hat{f}P \xrightarrow{(\hat{f}+g)\ell} (\hat{f}+g)P'$. By (Res) we have $B \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{f}P \xrightarrow{(\hat{f}+g)\ell} \mathbf{new}\ \hat{x}\ \mathbf{in}\ (\hat{f}+g)P'$, so $B \vdash f(\mathbf{new}\ x\ \mathbf{in}\ P) \xrightarrow{(f+g)\ell} (f+g)\mathbf{new}\ x\ \mathbf{in}\ P'$.

(Open) Define $\hat{f} : (A, x) \to (B, g(x))$ and $\hat{g}$ as $f + (x \mapsto g(x))$ and $g \downharpoonleft (\mathrm{fn}(\overline{y}x) - (A, x)) = \emptyset$ respectively. By the induction hypothesis $B, g(x) \vdash \hat{f}P \xrightarrow{(\hat{f}+\hat{g})\overline{y}x} (\hat{f}+\hat{g})P'$, so by (Open) $B \vdash \mathbf{new}\ g(x)\ \mathbf{in}\ \hat{f}P \xrightarrow{(\hat{f}+\hat{g})\overline{y}x} (\hat{f}+\hat{g})P'$, so as $f + g = \hat{f} + \hat{g}$ we have $B \vdash f(\mathbf{new}\ x\ \mathbf{in}\ P) \xrightarrow{(f+g)\overline{y}x} (f+g)P'$.

□

Lemma 12 (Shifting) *$(A \vdash P \xrightarrow{zv} P' \wedge v \notin A)$ iff $(A, v \vdash P \xrightarrow{zv} P' \wedge v \notin \mathrm{fn}(P))$.*

Proof By two inductions on derivations of transitions.

(Out),(Com),(Open) vacuous.

(Par),(Struct Right) Straightforward uses of the induction hypothesis.

(In) Straightforward.

(Res) Consider

$$\frac{A, y \vdash P \xrightarrow{zv} P'}{A \vdash \mathbf{new}\ y\ \mathbf{in}\ P \xrightarrow{zv} \mathbf{new}\ y\ \mathbf{in}\ P'}\ (\text{Res}) \qquad \frac{A, x, y \vdash P \xrightarrow{zv} P'}{A, x \vdash \mathbf{new}\ y\ \mathbf{in}\ P \xrightarrow{zv} \mathbf{new}\ y\ \mathbf{in}\ P'}\ (\text{Res})$$

$y \notin \mathrm{fn}(zv)$	$y \notin \mathrm{fn}(zv)$
$x \in \mathrm{fn}(v) - A$	$x \in \mathrm{fn}(v) - \mathrm{fn}(\mathbf{new}\ y\ \mathbf{in}\ P))$

For the left-to-right implication, note that $x \in \mathrm{fn}(v) - (A, y)$, so by the induction hypothesis $A, y, x \vdash P \xrightarrow{zv} P'$ and $x \in \mathrm{fn}(v) - \mathrm{fn}(P)$. For the right-to-left implication, note that as A, x, y is well-formed we have $x \in \mathrm{fn}(v) - \mathrm{fn}(P)$, so by the induction hypothesis $A, y \vdash P \xrightarrow{zv} P'$ and $x \in \mathrm{fn}(v) - (A, y)$.

□

Lemma 13 (Converse to Injective Substitution) *For $f : A \to B$ injective, if $B \vdash fP \xrightarrow{\ell'} Q'$ then (at least) one of the following two cases applies*

1. *there exist ℓ, Q, $g : (\mathrm{fn}(\ell) - A) \to_{\mathrm{bij}} \hat{B}$ such that $\hat{B} \cap B = \emptyset$ and $\ell' = (f + g)(\ell)$ and $A \vdash P \xrightarrow{\ell} Q$ and $Q' = (f + g)Q$.*
2. *there exist $x \in A$, $y \notin A$, $z \in B - \mathrm{ran}(f)$ and Q such that $\ell' = f(x)z$ and $A \vdash P \xrightarrow{xy} Q$ and $Q' = (f + \{z/y\})Q$.*

Proof Induction on derivations of transitions.

(Out) We have $B \vdash f(\overline{x}v) \xrightarrow{\overline{f(x)}f(v)} 0$. Taking $\ell = \overline{x}v$, $Q = 0$ and $g : \emptyset \to \emptyset$ gives clause 1.

(In) Suppose wlg that $p \notin A \cup B$. We have $B \vdash f(xp.P) \xrightarrow{f(x)z} \{^z/_p\}fP$.

Case $z \in \mathrm{ran}(f)$**.** Take some $y \in A$ such that $f(y) = z$, then $A \vdash xp.P \xrightarrow{xy} \{^y/_p\}P$. Taking $\ell = xy$, $Q = \{^y/_p\}P$ and $g : \emptyset \to \emptyset$ gives clause 1, as $\{^z/_p\}fP = f(\{^y/_p\}P)$.

Case $z \notin B$**.** Take some $y \notin A$, then $A \vdash xp.P \xrightarrow{xy} \{^y/_p\}P$. Taking $\ell = xy$, $Q = \{^y/_p\}P$, $\hat{B} = \{z\}$ and $g : \{y\} \to \{z\}$ gives clause 1, as $\{^z/_p\}fP = (f + g)(\{^y/_p\}P)$.

Case $z \in B - \mathrm{ran}(f)$**.** Take some $y \notin A$, then $A \vdash xp.P \xrightarrow{xy} \{^y/_p\}P$. Taking $Q = \{^y/_p\}P$ gives clause 2, as $\{^z/_p\}fP = (f + \{z/y\})(\{^y/_p\}P)$.

(Par) By the induction hypothesis.

(Com) Consider an instance

$$\textsc{Com}\ \frac{B \vdash fP \xrightarrow{\overline{x}v} P' \quad B \vdash fQ \xrightarrow{xv} Q'}{B \vdash fP \mid fQ \xrightarrow{\tau} \mathbf{new}\ \{v\} - B\ \mathbf{in}\ (P' \mid Q')}$$

By Lemma 9 either $v \in \mathrm{ran}(f)$ or $v \notin B$, so clause 2 of the induction hypothesis cannot apply. The first case is by two routine uses of the induction hypothesis. For the second case, by the induction hypothesis for the output transition there exist x_1, v_1, $\hat{P}$, $g_1 : \{v_1\} \to \{v\}$ such that $f(x_1) = x$ and $A \vdash P \xrightarrow{\overline{x_1}v_1} \hat{P}$ and $P' = (f + g_1)\hat{P}$. By the induction hypothesis for the input transition there exist x_2, v_2, $\hat{Q}$, $g_2 : \{v_2\} \to \{v\}$ such that $f(x_2) = x$ and $A \vdash Q \xrightarrow{x_2 v_2} \hat{Q}$ and $Q' = (f + g_2)\hat{Q}$.

By f injective $x_1 = x_2$.

By Lemma 11 $A \vdash Q \xrightarrow{x_1 v_1} \{^{v_1}/_{v_2}\}\hat{Q}$.

By (Com) $A \vdash P \mid Q \xrightarrow{\tau} \mathbf{new}\ \{v_1\} - A\ \mathbf{in}\ (\hat{P} \mid \{^{v_1}/_{v_2}\}\hat{Q})$.

It remains only to note that

$$\mathbf{new}\ \{v\} - B\ \mathbf{in}\ (P' \mid Q') = f\mathbf{new}\ \{v_1\} - A\ \mathbf{in}\ (\hat{P} \mid \{^{v_1}/_{v_2}\}\hat{Q})$$

(Res) Consider an instance

$$\textsc{Res}\ \frac{B,x' \vdash P' \xrightarrow{\ell'} Q' \quad x' \notin \mathrm{fn}(\ell')}{B \vdash \mathbf{new}\ x'\ \mathbf{in}\ P' \xrightarrow{\ell'} \mathbf{new}\ x'\ \mathbf{in}\ Q'}$$

where $fP = \mathbf{new}\ x'\ \mathbf{in}\ P'$. There exist $x \notin A$ and P_0 such that $P = \mathbf{new}\ x\ \mathbf{in}\ P_0$ and $P' = (f + \{x'/x\})P_0$. By the induction hypothesis either

1. there exist ℓ, Q_0, $g : (\mathrm{fn}(\ell) - A, x) \rightarrow_{\mathrm{bij}} \hat{B}$ such that $\hat{B} \cap (B, x') = \emptyset$ and $\ell' = (f + \{x'/x\} + g)(\ell)$ and $A, x \vdash P_0 \xrightarrow{\ell} Q_0$ and $Q' = (f + \{x'/x\} + g)Q_0$. It follows that $x \notin \mathrm{fn}(\ell)$.

   By (Res) $A \vdash P \xrightarrow{\ell} \mathbf{new}\ x\ \mathbf{in}\ Q_0$.

   Taking $Q = \mathbf{new}\ x\ \mathbf{in}\ Q_0$ gives clause 1.
2. there exist $w \in A, x$, $y \notin A, x$, $z \in B, x' - \mathrm{ran}(f + \{x'/x\})$ and Q_0 such that $\ell' = ((f + \{x'/x\})w)z$ and $A, x \vdash P_0 \xrightarrow{wy} Q_0$ and $Q' = (f + \{x'/x\} + \{z/y\})Q_0$. It follows that $w \neq x$.

   By (Res) $A \vdash P \xrightarrow{wy} \mathbf{new}\ x\ \mathbf{in}\ Q_0$.

   Taking $Q = \mathbf{new}\ x\ \mathbf{in}\ Q_0$ gives clause 2.

(Open) Consider an instance

$$\textsc{Open}\ \frac{B,x' \vdash P' \xrightarrow{\overline{y}x'} Q' \quad y \neq x'}{B \vdash \mathbf{new}\ x'\ \mathbf{in}\ P' \xrightarrow{\overline{y}x'} Q'}$$

where $fP = \mathbf{new}\ x'\ \mathbf{in}\ P'$. There exist $x \notin A$ and P_0 such that $P = \mathbf{new}\ x\ \mathbf{in}\ P_0$ and $P' = (f + \{x'/x\})P_0$.

By the induction hypothesis there exist $y_0 \in A$ and Q such that $f(y_0) = y$ and $A, x \vdash P_0 \xrightarrow{\overline{y_0}x} Q$ and $Q' = (f + \{x'/x\} + \emptyset)Q$.

By (Open) $A \vdash P \xrightarrow{\overline{y_0}x} Q$.

Taking $\ell = \overline{y_0}x$ and $g : \{x\} \rightarrow \{x'\}$ gives clause 1.

(Struct Right) Consider an instance

$$\textsc{Struct Right}\ \frac{B \vdash fP \xrightarrow{\ell'} Q' \quad Q' \equiv Q''}{B \vdash fP \xrightarrow{\ell'} Q''}$$

By the induction hypothesis there exists ℓ and Q (and other data, depending on whether case 1 or 2 holds) such that $A \vdash P \xrightarrow{\ell} Q$ and $Q' = (f + g)Q$ (writing g for $\{z/y\}$ in 2). Regarding $(f + g)^{-1}$ as a bijection with domain $\mathrm{ran}(f) \cup \mathrm{ran}(g)$, we have $(f + g)^{-1}Q' = Q$ and by Lemma 8 $(f + g)^{-1}Q' \equiv (f + g)^{-1}Q''$, so by (Struct Right) $A \vdash P \xrightarrow{\ell} (f + g)^{-1}Q''$.

□

Lemma 14 (Strengthening) *If $A, B \vdash P \xrightarrow{\ell} P'$ and $B \cap \mathrm{fn}(P, \ell) = \emptyset$ then $A \vdash P \xrightarrow{\ell} P'$.*

Proof This is a simple corollary of Lemmas 13 and 11. We also give a direct proof by induction on derivations of transitions.

(OUT), (IN) All immediate.

(PAR),(STRUCT RIGHT) Straightforward use of the induction hypothesis.

(COM) We have a rule instance of the form

$$\text{Com}\,\frac{A, B \vdash P \xrightarrow{\overline{x}v} P' \quad A, B \vdash Q \xrightarrow{xv} Q'}{A, B \vdash P \mid Q \xrightarrow{\tau} \textbf{new } \{v\} - (A, B) \textbf{ in } (P' \mid Q')}$$

By Lemma 9.2 $\text{fn}(\overline{x}v) \cap (A, B) \subseteq \text{fn}(P)$ and by assumption $B \cap \text{fn}(P) = \emptyset$ so $\text{fn}(\overline{x}v) \cap B = \emptyset$. By the induction hypothesis and (COM) we then have $A \vdash P \mid Q \xrightarrow{\tau} \textbf{new } \{v\} - A \textbf{ in } (P' \mid Q')$, but $\{v\} - A = \{v\} - (A, B)$, so we have the transition $A \vdash P \mid Q \xrightarrow{\tau} \textbf{new } \{v\} - (A, B) \textbf{ in } (P' \mid Q')$ as required.

(RES) We have a rule instance of the form

$$\text{Res}\,\frac{A, B, x \vdash P \xrightarrow{\ell} P'}{A, B \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\ell} \textbf{new } x \textbf{ in } P'}$$

with $x \notin \text{fn}(\ell)$. By A, B, x well-formed we have $x \notin B$, so $B \cap \text{fn}(\textbf{new } x \textbf{ in } P) = \emptyset$ implies $B \cap \text{fn}(P) = \emptyset$. By the induction hypothesis $A, x \vdash P \xrightarrow{\ell} P'$ so by (RES) $A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\ell} \textbf{new } x \textbf{ in } P'$.

(OPEN) Similar to (RES), noting that the sidecondition is a predicate on x and the label only.

□

Lemma 15 (Weakening and Strengthening) *$(A \vdash P \xrightarrow{\ell} P' \wedge x \notin A \cup \text{fn}(\ell))$ iff $(A, x \vdash P \xrightarrow{\ell} P' \wedge x \notin \text{fn}(P, \ell))$.*

PROOF The right-to-left implication follows from the well-formedness of A, x and from Lemma 14. The left-to-right implication follows from the condition $\text{fn}(P) \subseteq A$ in the definition of the transition rules and from Lemma 11, taking f to be the inclusion from A to A, x and g the identity on $\text{fn}(\ell) - A$. □

Lemma 16 (Transition Analysis)

1. *$A \vdash \overline{x}v \xrightarrow{\ell} Q$ iff $\text{fn}(\overline{x}v) \subseteq A$, $\ell = \overline{x}v$ and $Q \equiv 0$.*

2. *$A \vdash xp.P \xrightarrow{\ell} Q$ iff there exists v such that $\text{fn}(xp.P) \subseteq A$, $\ell = xv$, and $Q \equiv \{^v/_p\}P$.*

3. *$A \vdash P \mid Q \xrightarrow{\ell} R$ iff either*

   (a) *there exists $\hat{P}$ such that $\text{fn}(Q) \subseteq A$, $A \vdash P \xrightarrow{\ell} \hat{P}$ and $R \equiv \hat{P} \mid Q$.*

   (b) *there exist x, v, $\hat{P}$ and $\hat{Q}$ such that $\ell = \tau$, $A \vdash P \xrightarrow{\overline{x}v} \hat{P}$, $A \vdash Q \xrightarrow{xv} \hat{Q}$, and $R \equiv \textbf{new } \{v\} - A \textbf{ in } (\hat{P} \mid \hat{Q})$.*

   or symmetric cases.

4. *$A \vdash \textbf{new } x \textbf{ in } P \xrightarrow{\ell} Q$ iff either*

   (a) *there exist $\hat{x} \notin A \cup \text{fn}(\ell) \cup (\text{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{^{\hat{x}}/_x\}P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \textbf{new } \hat{x} \textbf{ in } \hat{Q}$.*

(b) there exist y, $\hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q \equiv \hat{Q}$.

PROOF The right-to-left implications are all shown using a single transition rule together with (TRANS STRUCT RIGHT). The left-to-right implications are shown by induction on derivations of transitions. Only the input and restriction cases are at all interesting; we give just the restriction case.

Case 4a, ($\Leftarrow$) By Lemma 9, $\mathrm{fn}(\{{}^{\hat{x}}/_{x}\}P) \subseteq A, \hat{x}$, so we have $\mathrm{fn}(\mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P) \subseteq A$. By (TRANS RES), $A \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. By $\hat{x} \notin \mathrm{fn}(P) - x$ we have $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P = \mathbf{new}\ x\ \mathbf{in}\ P$. By (TRANS STRUCT RIGHT), $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$.

Case 4b, ($\Leftarrow$) Again by Lemma 9, $\mathrm{fn}(\{{}^{\hat{x}}/_{x}\}P) \subseteq A, \hat{x}$, so we have $\mathrm{fn}(\mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P) \subseteq A$. By (TRANS OPEN), $A \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$. Again by $\hat{x} \notin \mathrm{fn}(P) - x$, we have $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \{{}^{\hat{x}}/_{x}\}P = \mathbf{new}\ x\ \mathbf{in}\ P$ so by (TRANS STRUCT RIGHT) $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$.

Case 4, ($\Rightarrow$) Suppose $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$. Let

$$\Phi(A, R, \ell, Q) \overset{def}{\Leftrightarrow} R = \mathbf{new}\ x\ \mathbf{in}\ P \implies (4a \vee 4b)$$

We show Φ is closed under the rules defining labelled transitions.

(TRANS RES) An instance of (TRANS RES) with conclusion $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ must be of the form

$$\textsc{Res}\ \frac{A, \hat{x} \vdash \hat{P} \xrightarrow{\ell} \hat{Q}}{A \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P} \xrightarrow{\ell} \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}}\ \hat{x} \notin \mathrm{fn}(\ell)$$

for some $\hat{x}$, $\hat{P}$, $\hat{Q}$ with $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P} = \mathbf{new}\ x\ \mathbf{in}\ P$, $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q} = Q$ and $\mathrm{fn}(\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P}) \subseteq A$. By $A, \hat{x}$ defined and $\hat{x} \notin \mathrm{fn}(\ell)$ we have $\hat{x} \notin A \cup \mathrm{fn}(\ell)$. By $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P} = \mathbf{new}\ x\ \mathbf{in}\ P$ we have $\hat{x} \notin \mathrm{fn}(P) - x$ and $\hat{P} = \{{}^{\hat{x}}/_{x}\}P$, so $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \hat{Q}$. By reflexivity of $\equiv$, we have $Q \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. So clause 4a holds.

(TRANS OPEN) An instance of (TRANS OPEN) for $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ must be of the form

$$\textsc{Open}\ \frac{A, \hat{x} \vdash \hat{P} \xrightarrow{\overline{y}\hat{x}} Q}{A \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P} \xrightarrow{\overline{y}\hat{x}} Q}\ y \neq \hat{x}$$

for some y, $\hat{x}$, and $\hat{P}$ with $\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P} = \mathbf{new}\ x\ \mathbf{in}\ P$, $\overline{y}\hat{x} = \ell$ and also $\mathrm{fn}(\mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{P}) \subseteq A$. As before $\hat{x} \notin A \cup (\mathrm{fn}(P) - x)$ and $\hat{P} = \{{}^{\hat{x}}/_{x}\}P$, so taking $\hat{Q} = Q$ clause 4b holds.

(TRANS STRUCT RIGHT) An instance of (TRANS STRUCT RIGHT) with conclusion $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ must be of the form

$$\textsc{Struct Right}\ \frac{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q' \quad Q' \equiv Q}{A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q}$$

for some Q' with $\mathrm{fn}(\mathbf{new}\ x\ \mathbf{in}\ P) \subseteq A$. By $\Phi(A, \mathbf{new}\ x\ \mathbf{in}\ P, \ell, Q')$ either

Case 4a there exist $\hat{x} \notin A \cup \mathrm{fn}(\ell) \cup (\mathrm{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \hat{Q}$ and $Q' \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. By $\equiv$ an equivalence we have $Q \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$, so clause 4a holds.

Case 4b there exist $y, \hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q' \equiv \hat{Q}$. By $\equiv$ an equivalence we have $Q \equiv \hat{Q}$, so clause 4b holds.

The cases for all other rules are vacuous.

□

Lemma 17 (Transition Analysis – Unary Disjoint New) *If $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ and $x \notin A \cup \mathrm{fn}(\ell)$ then either*

1. *there exists Q' such that $A, x \vdash P \xrightarrow{\ell} Q'$ and $Q \equiv \mathbf{new}\ x\ \mathbf{in}\ Q'$*

2. *there exist $y \in A$, $\hat{x} \notin A$ and Q' such that $\ell = \overline{y}\hat{x}$, $A, x \vdash P \xrightarrow{\overline{y}x} Q'$, and $Q \equiv \{\hat{x}/x\}Q'$.*

Proof By Lemma 16.4 either

1. there exist $\hat{x} \notin A \cup \mathrm{fn}(\ell) \cup (\mathrm{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. By Lemma 11 with $(id_A + \{x/\hat{x}\}) : A, \hat{x} \to A, x$ and $id_{\mathrm{fn}(\ell)-A}$ we have $A, x \vdash P \xrightarrow{\ell} \{x/\hat{x}\}\hat{Q}$. Taking $Q' = \{x/\hat{x}\}\hat{Q}$ gives clause 1.

2. there exist $y, \hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q \equiv \hat{Q}$. By Lemma 9 $y \in A, \hat{x}$ so $y \in A$. By Lemma 11 $A, x \vdash P \xrightarrow{\overline{y}x} (id_A + \{x/\hat{x}\})\hat{Q}$. Taking Q' to be the target of this transition gives clause 2.

□

Lemma 18 (Transition Analysis – n-ary Disjoint New) *If $A \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$, $A \cap B = \emptyset$ and $\mathrm{fn}(\ell) \cap B = \emptyset$ then either*

1. *there exists Q' such that $A, B \vdash P \xrightarrow{\ell} Q'$ and $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ Q'$*

2. *there exist $y \in A$, $\hat{x} \notin A$, $x \in B$ and Q' such that $\ell = \overline{y}\hat{x}$, $A, B \vdash P \xrightarrow{\overline{y}x} Q'$, and $Q \equiv \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q'$.*

Proof Induction on the size of B. The empty case is immediate, giving clause 1. Otherwise, consider $A \vdash \mathbf{new}\ b\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$. By Lemma 17 either

1. there exists Q'' such that $A, b \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q''$ and $Q \equiv \mathbf{new}\ b\ \mathbf{in}\ Q''$

   By the induction hypothesis either

   (a) there exists Q such that $A, b, B \vdash P \xrightarrow{\ell} Q$ and $Q'' \equiv \mathbf{new}\ B\ \mathbf{in}\ Q$. Clause 1 holds.

   (b) there exist $y \in A, b$, $\hat{x} \notin A, b$, $x \in B$ and Q' such that $\ell = \overline{y}\hat{x}$, $A, b, B \vdash P \xrightarrow{\overline{y}x} Q'$, and $Q'' \equiv \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q'$. By the assumption $\mathrm{fn}(\ell) \cap b, B = \emptyset$ we have $y \in A$. Now, $Q \equiv \mathbf{new}\ b\ \mathbf{in}\ Q'' \equiv \mathbf{new}\ b\ \mathbf{in}\ \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q' \equiv \{\hat{x}/x\}\mathbf{new}\ (b, B) - x\ \mathbf{in}\ Q'$, giving clause 2.

2. there exist $y \in A$, $\hat{x} \notin A$ and Q'' such that $\ell = \overline{y}\hat{x}$, $A, b \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\overline{y}b} Q''$, and $Q \equiv \{\hat{x}/b\}Q''$.

   By the induction hypothesis there exists Q' such that $A, b, B \vdash P \xrightarrow{\overline{y}b} Q'$ and $Q'' \equiv \mathbf{new}\ B\ \mathbf{in}\ Q'$ (case 2 leads to a contradiction). Taking $x = b$ we have clause 2.

□

Lemma 19 (Transition Analysis – Unary A-Disjoint New) *If $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$ and $x \notin A$ then either*

1. *For any $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, x$ there exists $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, x, C)$ injective and Q' such that $A, x \vdash P \xrightarrow{(id_A + g)\ell} Q'$ and $Q \equiv (id_A + g^{-1})\mathbf{new}\ x\ \mathbf{in}\ Q'$.*

2. *There exists $y \in A$, $\hat{x} \notin A$ and Q' such that $\ell = \overline{y}\hat{x}$, $A, x \vdash P \xrightarrow{\overline{y}x} Q'$ and $Q \equiv \{{}^{\hat{x}}/_{x}\}Q'$.*

PROOF By Lemma 16.4 either

1. there exist $\hat{x} \notin A \cup \mathrm{fn}(\ell) \cup (\mathrm{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. Consider $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, x$ and an arbitrary $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, x, C)$ injective.

   As $\hat{x} \notin \mathrm{fn}(\ell)$ we can regard g as a function $g : (\mathrm{fn}(\ell) - A, \hat{x}) \to (\mathcal{N} - A, x, C)$ and moreover $(id_A + \{x/\hat{x}\} + g)\ell = (id_A + g)\ell$, so by Lemma 11 $A, x \vdash P \xrightarrow{(id_A+g)\ell} (id_A + \{x/\hat{x}\} + g)\hat{Q}$. Take Q' to be the target of this transition. Now $(id_A + g^{-1})\mathbf{new}\ x\ \mathbf{in}\ Q' = (id_A + g^{-1})\mathbf{new}\ x\ \mathbf{in}\ (id_A + \{x/\hat{x}\} + g)\hat{Q} = \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q} \equiv Q$.

2. there exist y, $\hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{{}^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q \equiv \hat{Q}$. By Lemma 9 $y \in A, \hat{x}$ so $y \in A$. By Lemma 11 $A, x \vdash P \xrightarrow{\overline{y}x} (id_A + \{x/\hat{x}\})\hat{Q}$. Take Q' to be the target of this transition.

□

Lemma 20 (Transition Analysis – n-ary A-Disjoint New) *If $A \vdash \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$ and $A \cap B = \emptyset$ then either*

1. *there exist $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, B)$ injective and Q' such that $A, B \vdash P \xrightarrow{(id_A+g)\ell} Q'$ and $Q \equiv (id_A + g^{-1})\mathbf{new}\ B\ \mathbf{in}\ Q'$.*

2. *there exist $y \in A$, $\hat{x} \notin A$, $x \in B$ and Q' such that $\ell = \overline{y}\hat{x}$, $A, B \vdash P \xrightarrow{\overline{y}x} Q'$, and $Q \equiv \{\hat{x}/x\}\mathbf{new}\ B - x\ \mathbf{in}\ Q'$.*

PROOF We prove a stronger result, with clause 1 replaced by

1. for any $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, B$ there exist $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - A, B, C)$ injective and Q' such that $A, B \vdash P \xrightarrow{(id_A+g)\ell} Q'$ and $Q \equiv (id_A + g^{-1})\mathbf{new}\ B\ \mathbf{in}\ Q'$.

by induction on the size of B. The empty case is immediate from Lemma 11 (giving clause 1). Otherwise, consider $A \vdash \mathbf{new}\ b\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ P \xrightarrow{\ell} Q$ and $C \subseteq_{\mathrm{fin}} \mathcal{N} - A, (b, B)$. By Lemma 19 either

1. (as $C \subseteq_{\text{fin}} \mathcal{N} - A, b$) there exists $\hat{g} : (\text{fn}(\ell) - A) \to (\mathcal{N} - A, b, C)$ injective and $\hat{Q}'$ such that $A, b \vdash \textbf{new } B \textbf{ in } P \xrightarrow{(id_A + \hat{g})\ell} \hat{Q}'$ and $Q \equiv (id_A + \hat{g}^{-1})\textbf{new } b \textbf{ in } \hat{Q}'$. By the induction hypothesis either
   (a) for any $\hat{C} \subseteq_{\text{fin}} \mathcal{N} - (A, b), B$ there exist $\hat{\hat{g}} : (\text{fn}((id_A + \hat{g})\ell) - (A, b)) \to (\mathcal{N} - (A, b), B, \hat{C})$ injective and Q' such that $(A, b), B \vdash P \xrightarrow{(id_{(A,b)} + \hat{\hat{g}})(id_A + \hat{g})\ell} Q'$ and $\hat{Q}' \equiv (id_{(A,b)} + \hat{\hat{g}}^{-1})\textbf{new } B \textbf{ in } Q'$. Using this for $\hat{C} = C$ and taking $g = \hat{\hat{g}} \circ \hat{g}$ gives clause 1.
   (b) there exist $y \in (A, b)$, $\hat{\hat{x}} \notin (A, b)$, $x \in B$ and Q' such that $(id_A + \hat{g})\ell = \overline{y}\hat{\hat{x}}$, $(A, b), B \vdash P \xrightarrow{\overline{y}x} Q'$, and $\hat{Q}' \equiv \{\hat{\hat{x}}/x\}\textbf{new } B - x \textbf{ in } Q'$.
   We have $y \in A$ by Lemma 9. Taking $\hat{x} = \hat{g}^{-1}(\hat{\hat{x}})$ gives clause 2.

2. There exists $y \in A$, $\hat{x} \notin A$ and $\hat{Q}'$ such that $\ell = \overline{y}\hat{x}$, $A, b \vdash \textbf{new } B \textbf{ in } P \xrightarrow{\overline{y}b} \hat{Q}'$ and $Q \equiv \{^{\hat{x}}/_b\}\hat{Q}'$. Trivially $b \in b, B$. By the induction hypothesis (clause 1 must hold, so taking C empty and g the empty function) there exists Q' such that $A, b, B \vdash P \xrightarrow{\overline{y}b} Q'$ and $\hat{Q}' \equiv \textbf{new } B \textbf{ in } Q'$. It remains only to note that $Q \equiv \{^{\hat{x}}/_b\}\hat{Q}' = \{^{\hat{x}}/_b\}\textbf{new } (b, B) - b \textbf{ in } Q'$ to show clause 2.

□

5.2 Coincidence of the Two Semantics

Take the *size* of a derivation of a structural congruence to be number of instances of inference rules contained in it.

Lemma 21 *For any derivation of $P' \equiv P$ there is a derivation of the same size of $\{^v/_p\}P' \equiv \{^v/_p\}P$.*

Proof Routine induction. □

Lemma 22 *If $P' \equiv P$ then $A \vdash P' \xrightarrow{\ell} Q$ iff $A \vdash P \xrightarrow{\ell} Q$.*

Proof Induction on the size of derivation of $P' \equiv P$. In symmetric cases we show only the right-to-left direction of the conclusion.

(Struct Cong Refl) By the reflexivity of iff.

(Struct Cong Sym) By the symmetry of iff.

(Struct Cong Tran) By the induction hypothesis and transitivity of iff.

(Struct Cong Input) Consider $P' \equiv P$ and $A \vdash xp.P \xrightarrow{\ell} Q$. By Lemma 16.2, there exists v such that $\text{fn}(xp.P) \subseteq A$, $\ell = xv$, and $Q \equiv \{^v/_p\}P$. Using Lemma 7, $\text{fn}(xp.P') = \text{fn}(xp.P)$. By Lemma 21 $\{^v/_p\}P' \equiv \{^v/_p\}P$, so $Q \equiv \{^v/_p\}P'$. Finally by Lemma 16.2, $A \vdash xp.P' \xrightarrow{\ell} Q$.

(Struct Cong Par) Consider $P' \equiv P$, $Q' \equiv Q$ and $A \vdash P \mid Q \xrightarrow{\ell} R$. By Lemma 16.3 one of the following holds.

Case 16.3a there exists $\hat{P}$ such that $\text{fn}(Q) \subseteq A$, $A \vdash P \xrightarrow{\ell} \hat{P}$ and $R \equiv \hat{P} \mid Q$. By Lemma 7, $\text{fn}(Q') = \text{fn}(Q)$. By the inductive hypothesis $A \vdash P' \xrightarrow{\ell} \hat{P}$ and clearly $\hat{P} \mid Q \equiv \hat{P} \mid Q'$, so by Lemma 16.3, $A \vdash P' \mid Q' \xrightarrow{\ell} R$.

Case 16.3b there exist x, v, $\hat{P}$ and $\hat{Q}$ such that $\ell = \tau$, $A \vdash P \xrightarrow{\overline{x}v} \hat{P}$, $A \vdash Q \xrightarrow{xv} \hat{Q}$, and $R \equiv \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\hat{P} \mid \hat{Q})$. By the induction hypothesis $A \vdash P' \xrightarrow{\overline{x}v} \hat{P}$ and $A \vdash Q' \xrightarrow{xv} \hat{Q}$. By Lemma 16.3, $A \vdash P' \mid Q' \xrightarrow{\ell} R$.

or symmetric cases.

(Struct Cong Res) Consider $P' \equiv P$ and $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P \xrightarrow{\ell} Q$. By Lemma 16.4 one of the following holds.

Case 16.4a there exist $\hat{x} \notin A \cup \mathrm{fn}(\ell) \cup (\mathrm{fn}(P) - x)$ and $\hat{Q}$ such that $A, \hat{x} \vdash \{^{\hat{x}}/_{x}\}P \xrightarrow{\ell} \hat{Q}$ and $Q \equiv \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$. By Lemma 21 $\{^{\hat{x}}/_{x}\}P' \equiv \{^{\hat{x}}/_{x}\}P$ (with a derivation of the same size). By the induction hypothesis $A, \hat{x} \vdash \{^{\hat{x}}/_{x}\}P' \xrightarrow{\ell} \hat{Q}$. By Lemma 16.4 $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P' \xrightarrow{\ell} Q$.

Case 16.4b there exist y, $\hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P) - x)$, and $\hat{Q}$ such that $\ell = \overline{y}\hat{x}$, $A, \hat{x} \vdash \{^{\hat{x}}/_{x}\}P \xrightarrow{\overline{y}\hat{x}} \hat{Q}$, and $Q \equiv \hat{Q}$. By Lemma 21 $\{^{\hat{x}}/_{x}\}P' \equiv \{^{\hat{x}}/_{x}\}P$, with a derivation of the same size. By the induction hypothesis $A, \hat{x} \vdash \{^{\hat{x}}/_{x}\}P' \xrightarrow{\overline{y}v} \hat{Q}$. By Lemma 7 $\mathrm{fn}(P') = \mathrm{fn}(P)$, so $\hat{x} \notin A \cup \{y\} \cup (\mathrm{fn}(P') - x)$. By Lemma 16.4, $A \vdash \mathbf{new}\ x\ \mathbf{in}\ P' \xrightarrow{\ell} Q$.

(Struct Par Nil), (Struct Par Com), (Struct Par Assoc), (Struct Res Res) These are straightforward. We check the other axiom in detail.

(Struct Res Par) Consider $\mathbf{new}\ \hat{x}\ \mathbf{in}\ (P \mid \hat{Q}) \equiv P \mid \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$ where $\hat{x} \notin \mathrm{fn}(P)$.

For the left-to-right direction, suppose $A \vdash \mathbf{new}\ \hat{x}\ \mathbf{in}\ (P \mid \hat{Q}) \xrightarrow{\ell} R$.

Take Q and $x \notin A \cup \mathrm{fn}(\ell)$ such that $\mathbf{new}\ x\ \mathbf{in}\ (P \mid Q) = \mathbf{new}\ \hat{x}\ \mathbf{in}\ (P \mid \hat{Q})$ and $P \mid \mathbf{new}\ x\ \mathbf{in}\ Q = P \mid \mathbf{new}\ \hat{x}\ \mathbf{in}\ \hat{Q}$.

We therefore have $A \vdash \mathbf{new}\ x\ \mathbf{in}\ (P \mid Q) \xrightarrow{\ell} R$.

By Lemma 17 either

1. there exists R' such that $A, x \vdash P \mid Q \xrightarrow{\ell} R'$ and $R \equiv \mathbf{new}\ x\ \mathbf{in}\ R'$. By Lemma 16.4 this transition holds iff one of the following holds:
   (a) (Par)[Left] there exists P' such that $\mathrm{fn}(Q) \subseteq A, x$, $A, x \vdash P \xrightarrow{\ell} P'$ and $R' \equiv P' \mid Q$.
   (b) (Par)[Right] there exists Q' such that $\mathrm{fn}(P) \subseteq A, x$, $A, x \vdash Q \xrightarrow{\ell} Q'$ and $R' \equiv Q' \mid P$.
   (c) (Com)[Left] there exist z, v, P' and Q' such that $\ell = \tau$, $A, x \vdash P \xrightarrow{\overline{z}v} P'$, $A, x \vdash Q \xrightarrow{zv} Q'$, and $R' \equiv \mathbf{new}\ \{v\} - A, x\ \mathbf{in}\ (P' \mid Q')$. By Lemma 9 $v \neq x$.
   (d) (Com)[Right] there exist z, v, Q' and P' such that $\ell = \tau$, $A, x \vdash Q \xrightarrow{\overline{z}v} Q'$, $A, x \vdash P \xrightarrow{zv} P'$, and $R' \equiv \mathbf{new}\ \{v\} - A, x\ \mathbf{in}\ (Q' \mid P')$.
2. there exist $y \in A$, $\hat{x} \notin A$ and R' such that $\ell = \overline{y}\hat{x}$, $A, x \vdash P \mid Q \xrightarrow{\overline{y}x} R'$, and $R \equiv \{\hat{x}/x\}R'$. By Lemma 16.4 this transition holds iff one of the following holds:
   (a) (Par)[Left] there exists P' such that $\mathrm{fn}(Q) \subseteq A, x$, $A, x \vdash P \xrightarrow{\overline{y}x} P'$ and $R' \equiv P' \mid Q$. By Lemma 9 this leads to a contradiction.

(b) (Par)[Right] there exists Q' such that $\text{fn}(P) \subseteq A, x$, $A, x \vdash Q \xrightarrow{\overline{y}x} Q'$ and $R' \equiv Q' \mid P$.

For the right-to-left direction, suppose $A \vdash P \mid \textbf{new } \hat{x} \textbf{ in } \hat{Q} \xrightarrow{\ell} R$.

Take Q and $x \notin A \cup \text{fn}(\ell)$ such that $\textbf{new } x \textbf{ in } (P \mid Q) = \textbf{new } \hat{x} \textbf{ in } (P \mid \hat{Q})$ and $P \mid \textbf{new } x \textbf{ in } Q = P \mid \textbf{new } \hat{x} \textbf{ in } \hat{Q}$.

We therefore have $A \vdash P \mid \textbf{new } x \textbf{ in } Q \xrightarrow{\ell} R$.

By Lemma 16.4 this transition holds iff one of the following holds:

1. (Par)[Left] there exists P' such that $\text{fn}(\textbf{new } x \textbf{ in } Q) \subseteq A$, $A \vdash P \xrightarrow{\ell} P'$ and $R' \equiv P' \mid \textbf{new } x \textbf{ in } Q$.
2. (Par)[Right] there exists Q' such that $\text{fn}(P) \subseteq A$, $A \vdash \textbf{new } x \textbf{ in } Q \xrightarrow{\ell} Q'$ and $R' \equiv Q' \mid P$.
   By Lemma 17 either
   (a) there exists Q'' such that $A, x \vdash Q \xrightarrow{\ell} Q''$ and $Q' \equiv \textbf{new } x \textbf{ in } Q''$
   (b) there exist $y \in A$, $\hat{x} \notin A$ and Q'' such that $\ell = \overline{y}\hat{x}$, $A, x \vdash Q \xrightarrow{\overline{y}x} Q''$, and $Q' \equiv \{\hat{x}/x\}Q''$.
3. (Com)[Left] there exist z, v, P' and Q'' such that $\ell = \tau$, $A \vdash P \xrightarrow{\overline{z}v} P'$, $A \vdash \textbf{new } x \textbf{ in } Q \xrightarrow{zv} Q''$, and $R' \equiv \textbf{new } \{v\} - A \textbf{ in } (P' \mid Q'')$.
   Without loss of generality $v \neq x$ (otherwise the transitions can be renamed by Lemma 11). By Lemma 17 there exists Q' such that $A, x \vdash Q \xrightarrow{zv} Q'$ and $Q'' \equiv \textbf{new } x \textbf{ in } Q'$
4. (Com)[Right] there exist z, w, Q'' and P' such that $\ell = \tau$, $A \vdash \textbf{new } x \textbf{ in } Q \xrightarrow{\overline{z}w} Q''$, $A \vdash P \xrightarrow{zw} P'$, and $R' \equiv \textbf{new } \{w\} - A \textbf{ in } (Q'' \mid P')$. Without loss of generality $w \neq x$ (otherwise the transitions can be renamed by Lemma 11).
   By Lemma 17 either
   (a) there exists Q' such that $A, x \vdash Q \xrightarrow{\overline{z}w} Q'$ and $Q'' \equiv \textbf{new } x \textbf{ in } Q'$
   (b) there exist $z \in A$, $w \notin A$ and Q' such that $\overline{z}w = \overline{z}w$, $A, x \vdash Q \xrightarrow{\overline{z}x} Q'$, and $Q'' \equiv \{w/x\}Q'$.

The cases correspond as follows – one can now check that in each there are matching transitions.

$\textbf{new } x \textbf{ in } (P \mid Q)$	$P \mid \textbf{new } x \textbf{ in } Q$
1a	1
1b	2a
1c	3
1d $v \neq x$	4a
1d $v = x$	4b
2b	2b

□

Lemma 23 *If* $\text{fn}(P) \subseteq A$ *and* $P \rightarrow Q$ *then* $A \vdash P \xrightarrow{\tau} Q$.

Proof Induction on the derivation of $P \rightarrow Q$.

(RED COM) For the base case we construct derivations of τ transitions:

$$\textsc{Com}\ \dfrac{\textsc{Out}\ \dfrac{}{A \vdash \overline{x}v \xrightarrow{\overline{x}v} 0} \qquad \textsc{In}\ \dfrac{}{A \vdash xp.P \xrightarrow{xv} \{^v/_p\}P}}{A \vdash \overline{x}v \,|\, xp.P \xrightarrow{\tau} \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (0 \,|\, \{^v/_p\}P)}$$

By the premise $\mathrm{fn}(\overline{x}v \,|\, xp.P) \subseteq A$ we have $v \in A$, so using (TRANS STRUCT RIGHT) we have $A \vdash \overline{x}v \,|\, xp.P \xrightarrow{\tau} \{^v/_p\}P$, the right hand side of which is exactly the right hand side of (RED COM).

(RED PAR), (RED RES) require straightforward uses of induction hypothesis, using the (TRANS PAR) and (TRANS RES) rules.

(RED STRUCT) By Lemma 7, $\mathrm{fn}(P') \subseteq A$. By the inductive hypothesis, $A \vdash P' \xrightarrow{\tau} Q'$. By Lemma 22, $A \vdash P \xrightarrow{\tau} Q'$. By (TRANS STRUCT RIGHT), $A \vdash P \xrightarrow{\tau} Q$.

□

Lemma 24 (Term Structure – Output Transition) *If $A \vdash P \xrightarrow{\overline{z}v} P'$ then we have $P \equiv \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, P')$*

PROOF Induction on the derivation of $A \vdash P \xrightarrow{\overline{z}v} P'$.

(TRANS OUT) Obvious.

(TRANS PAR) By the induction hypothesis, $P \equiv \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, P')$, so

$$\begin{array}{rcl} P \,|\, Q & \equiv & (\mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, P')) \,|\, Q \\ & \equiv & \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, P' \,|\, Q)\ \text{(as by } \mathrm{fn}(P \,|\, Q) \subseteq A \text{ we have } \mathrm{fn}(Q) \subseteq A) \end{array}$$

(TRANS RES) By the induction hypothesis $P \equiv \mathbf{new}\ \{v\} - (A, x)\ \mathbf{in}\ (\overline{z}v \,|\, P')$, so

$$\begin{array}{rcl} \mathbf{new}\ x\ \mathbf{in}\ P & \equiv & \mathbf{new}\ x\ \mathbf{in}\ \mathbf{new}\ \{v\} - (A, x)\ \mathbf{in}\ (\overline{z}v \,|\, P') \\ & \equiv & \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, \mathbf{new}\ x\ \mathbf{in}\ P')\ (\text{as } x \notin \mathrm{fn}(\overline{z}v)) \end{array}$$

(TRANS OPEN) By the induction hypothesis $P \equiv \mathbf{new}\ \{v\} - (A, x)\ \mathbf{in}\ (\overline{z}v \,|\, P')$, so

$$\begin{array}{rcl} \mathbf{new}\ x\ \mathbf{in}\ P & \equiv & \mathbf{new}\ x\ \mathbf{in}\ \mathbf{new}\ \{v\} - (A, x)\ \mathbf{in}\ (\overline{z}v \,|\, P') \\ & \equiv & \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{z}v \,|\, P')\ (\text{as } x = v \wedge x \neq z) \end{array}$$

(TRANS STRUCT-RIGHT) By the induction hypothesis.

All other cases are vacuous.

□

Lemma 25 (Term Structure – Input Transition) *If $A \vdash Q \xrightarrow{xv} Q'$ then there exist B, p, Q_1 and Q_2 such that $B \cap (A \cup \mathrm{fn}(xv)) = \{\}$, $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2)$ and $Q' \equiv \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2)$.*

PROOF Induction on derivation of $A \vdash Q \xrightarrow{xv} Q'$.

(TRANS IN) Obvious.

(Trans Par) Consider $A \vdash Q \,|\, P \xrightarrow{xv} Q' \,|\, P$. By the induction hypothesis there exist B, p, Q_1 and Q_2 such that $B \cap (A \cup \mathrm{fn}(xv)) = \{\}$, $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2)$ and $Q' \equiv \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2)$. Taking $\hat{Q}_2 = Q_2 \,|\, P$ we have

$$\begin{array}{rcl} Q \,|\, P & \equiv & \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2) \,|\, P \\ & \equiv & \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, \hat{Q}_2) \\ Q' \,|\, P & \equiv & \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2) \,|\, P \\ & \equiv & \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, \hat{Q}_2) \end{array}$$

(Trans Res) Consider $A \vdash \mathbf{new}\ z\ \mathbf{in}\ Q \xrightarrow{xv} \mathbf{new}\ z\ \mathbf{in}\ Q'$ with $z \notin A \cup \mathrm{fn}(xv)$. By the induction hypothesis there exist B, p, Q_1 and Q_2 such that $B \cap (A, z \cup \mathrm{fn}(xv)) = \{\}$, $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2)$ and $Q' \equiv \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2)$. Taking $\hat{B} = B, z$ we have

$$\begin{array}{rcl} \mathbf{new}\ z\ \mathbf{in}\ Q & \equiv & \mathbf{new}\ z\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2) \\ & \equiv & \mathbf{new}\ \hat{B}\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2) \\ \mathbf{new}\ z\ \mathbf{in}\ Q' & \equiv & \mathbf{new}\ z\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2) \\ & \equiv & \mathbf{new}\ \hat{B}\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2) \end{array}$$

(Trans Struct Right) By the induction hypothesis.

All other cases are vacuous.

□

Lemma 26 *If $A \vdash P \xrightarrow{\tau} Q$ then $P \rightarrow Q$.*

Proof Induction on derivations of $A \vdash P \xrightarrow{\tau} Q$

(Trans Par) By the induction hypothesis and (Red Par).

(Trans Com) By Lemma 24 $P \equiv \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{x}v \,|\, P')$. By Lemma 9 $x \in A$ so $P \equiv \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{x}v \,|\, P')$.

By Lemma 25 there exist B, p, Q_1 and Q_2 such that $B \cap (A \cup \mathrm{fn}(xv)) = \{\}$, $Q \equiv \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2)$ and $Q' \equiv \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2)$. We have

$$\begin{array}{rl} P \,|\, Q & \equiv\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{x}v \,|\, P') \,|\, \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2) \\ & \equiv\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (\overline{x}v \,|\, P' \,|\, \mathbf{new}\ B\ \mathbf{in}\ (xp.Q_1 \,|\, Q_2))\ (\text{as } \mathrm{fn}(Q) \subseteq A) \\ & \equiv\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ (\overline{x}v \,|\, P' \,|\, xp.Q_1 \,|\, Q_2)\ (\text{as } (A, v) \cap B = \{\}) \\ & \rightarrow\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, P' \,|\, Q_2)\ (\text{by Red Com}) \\ & \equiv\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (P' \,|\, \mathbf{new}\ B\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2))\ (\text{as } (A, v) \cap B = \{\}) \\ & \equiv\ \ \mathbf{new}\ \{v\} - A\ \mathbf{in}\ (P' \,|\, Q') \end{array}$$

(Trans Res) By the induction hypothesis and (Red Res).

(Trans Struct Right) By the induction hypothesis and (Red Struct).

All other cases are vacuous.

□

Lemma 27 (Term Structure – Tau Transition) *If $A \vdash P \xrightarrow{\tau} Q$ and $B \subseteq_{\mathrm{fin}} \mathcal{N}$ then there exist C, x, v, p, Q_1, Q_2 such that $C \cap (A \cup B) = \emptyset$, $p \notin A \cup B \cup C$, $P \equiv$ $\mathbf{new}\ C\ \mathbf{in}\ (\overline{x}v \,|\, xp.Q_1 \,|\, Q_2)$ and $Q \equiv \mathbf{new}\ C\ \mathbf{in}\ (\{^v/_p\}Q_1 \,|\, Q_2)$.*

PROOF By Lemma 26 $P \longrightarrow Q$. One can show the result for empty B by induction on derivations of reductions, and then use alpha-renaming. □

Theorem 2 *If* $\mathrm{fn}(P) \subseteq A$ *then* $P \longrightarrow Q$ *iff* $A \vdash P \xrightarrow{\tau} Q$.

PROOF This is immediate from Lemmas 23 and 26 above. □

5.3 Strong Bisimulation and Congruence

Lemma 28 $\Phi(\dot\sim) = \dot\sim$ *and* $\dot\sim$ *is an equivalence.*

PROOF We check a sequence of simple facts:

1. Φ is monotone, i.e. if $\mathcal{R} \subseteq \mathcal{R}'$ then $\Phi(\mathcal{R}) \subseteq \Phi(\mathcal{R}')$
2. $1 \subseteq \Phi(1)$
3. If $\mathcal{R} \subseteq \Phi(\mathcal{R})$ then $\mathcal{R}^{-1} \subseteq \Phi(\mathcal{R}^{-1})$.
4. If $\mathcal{R} \subseteq \Phi(\mathcal{R})$ then $\mathcal{R};\mathcal{R} \subseteq \Phi(\mathcal{R};\mathcal{R})$.
5. If $\forall i \in I \,.\, \mathcal{R}_i \subseteq \Phi(\mathcal{R}_i)$ then $\cup_{i\in I}\mathcal{R}_i \subseteq \Phi(\cup_{i\in I}\mathcal{R}_i)$.
6. $\dot\sim$ is a fixed point of Φ, i.e. $\dot\sim = \Phi(\dot\sim)$.
7. $\dot\sim$ is an equivalence.

Items 1–4 are immediate from the definition of Φ; the remainder follow from these as below.

5: Suppose $\forall i \in I \,.\, \mathcal{R}_i \subseteq \Phi(\mathcal{R}_i)$. By 1, as $\mathcal{R}_i \subseteq \cup_{i\in I}\mathcal{R}_i$, we have $\Phi(\mathcal{R}_i) \subseteq \Phi(\cup_{i\in I}\mathcal{R}_i)$. By transitivity of $\subseteq$ we have $\mathcal{R}_i \subseteq \Phi(\cup_{i\in I}\mathcal{R}_i)$. Hence $\cup_{i\in I}\mathcal{R}_i \subseteq \Phi(\cup_{i\in I}\mathcal{R}_i)$.

6: By 5 $\dot\sim \subseteq \Phi(\dot\sim)$. By 1 $\Phi(\dot\sim) \subseteq \Phi(\Phi(\dot\sim))$. By the definition of $\dot\sim$ $\Phi(\dot\sim) \subseteq \dot\sim$.

7: By 2 $1 \subseteq \dot\sim$ so have reflexivity; by 6 and 3 $\dot\sim^{-1} \subseteq \dot\sim$ so have symmetry; by 6 and 4 $\dot\sim;\dot\sim \subseteq \dot\sim$ so have transitivity.

□

Lemma 29 *If* $\mathcal{R}$ *is a loose bisimulation then* $\mathcal{R} \subseteq \dot\sim$.

PROOF We check $\mathcal{S}$ is a bisimulation, where

$$\mathcal{S}_A \stackrel{def}{=} \{\, f^{-1}P, f^{-1}Q \mid f : A \underset{\text{bij}}{\to} B \wedge P \equiv\mathcal{R}_B\equiv Q \,\}$$

Consider $P \equiv P_0\ \mathcal{R}_B\ Q_0 \equiv Q$, $f : A \to_{\text{bij}} B$ and the D provided by the loose bisimulation.

Suppose $A \vdash f^{-1}P \xrightarrow{\ell} P'$.

Take some $g : (\mathrm{fn}(\ell) - A) \to (\mathcal{N} - B, D)$ injective.

By Lemma 11 $B \vdash P \xrightarrow{(f+g)\ell} (f+g)P'$.

By Lemma 22 $B \vdash P_0 \xrightarrow{(f+g)\ell} (f+g)P'$.

By the loose bisimulation property and Lemma 22 there exist $\hat{Q}'$, C, and a bijection $h : B \cup \mathrm{fn}((f+g)\ell) \to C$ such that $B \vdash Q \xrightarrow{(f+g)\ell} \hat{Q}'$ and $h(f+g)P' \equiv\mathcal{R}_C\equiv h\hat{Q}'$.

By Lemma 11 $A \vdash f^{-1}Q \xrightarrow{\ell} (f+g)^{-1}\hat{Q}'$.
Now $h \circ (f+g) : A \cup \mathrm{fn}(\ell) \to_{\mathrm{bij}} C$, so $P' \; \mathcal{S}_{A\cup\mathrm{fn}(\ell)} \; (f+g)^{-1}\hat{Q}'$. □

Lemma 30 (Injective Substitution – Bisimulation) *If $P \stackrel{\cdot}{\sim}_A Q$ and $f : A \to B$ is injective then $fP \stackrel{\cdot}{\sim}_B fQ$.*

Proof We check

$$\mathcal{R}_B = \{\, fP_1, fP_2 \mid f : A \underset{\mathrm{inj}}{\to} B \wedge P_1 \stackrel{\cdot}{\sim}_A P_2 \,\}$$

is a bisimulation.

Suppose $B \vdash fP_1 \xrightarrow{\ell'} Q_1'$. By Lemma 13 one of the following holds.

1. there exist ℓ, Q_1, $g : (\mathrm{fn}(\ell) - A) \to_{\mathrm{bij}} \hat{B}$ such that $\hat{B} \cap B = \emptyset$ and $\ell' = (f+g)(\ell)$ and $A \vdash P_1 \xrightarrow{\ell} Q_1$ and $Q_1' = (f+g)Q_1$.

   By bisimulation $A \vdash P_2 \xrightarrow{\ell} Q_2$ and $Q_1 \stackrel{\cdot}{\sim}_{A\cup\mathrm{fn}(\ell)} Q_2$.

   By Lemma 11 $B \vdash fP_2 \xrightarrow{\ell'} (f+g)Q_2$.

   Finally $(f+g)Q_1 \; \mathcal{R}_{B\cup\mathrm{fn}(\ell')} \; (f+g)Q_2$.

2. there exist $x \in A$, $y \notin A$, $z \in B - \mathrm{ran}(f)$ and Q_1 such that $\ell' = f(x)z$ and $A \vdash P_1 \xrightarrow{xy} Q_1$ and $Q_1' = (f + \{z/y\})Q_1$.

   By bisimulation $A \vdash P_2 \xrightarrow{xy} Q_2$ and $Q_1 \stackrel{\cdot}{\sim}_{A,y} Q_2$.

   By Lemma 11 (for the pair of functions $f : A \to B - z$, $g : \{y\} \to \{z\}$) we have $B - z \vdash fP_2 \xrightarrow{\ell'} (f + \{z/y\})Q_2$.

   By Lemma 12 $B \vdash fP_2 \xrightarrow{\ell'} (f + \{z/y\})Q_2$.

   Finally $(f + \{z/y\})Q_1 \; \mathcal{R}_B \; (f + \{z/y\})Q_2$.

□

Lemma 31 (Congruence – Par and New) *If $P \stackrel{\cdot}{\sim}_{A,B} P'$ and $Q \stackrel{\cdot}{\sim}_{A,B} Q'$ then* **new** B **in** $(P \mid Q) \stackrel{\cdot}{\sim}_A$ **new** B **in** $(P' \mid Q')$.

Proof Let $\mathcal{R}_A = \{\, \textbf{new } B \textbf{ in } (P \mid Q), \textbf{new } B \textbf{ in } (P' \mid Q') \mid P \stackrel{\cdot}{\sim}_{A,B} P' \wedge Q \stackrel{\cdot}{\sim}_{A,B} Q' \,\}$. We check $\mathcal{R}$ is a loose bisimulation (omitting symmetric cases).

Suppose $A \vdash \textbf{new } B \textbf{ in } (P \mid Q) \xrightarrow{\ell} R$ and $\mathrm{fn}(\ell) \cap B = \emptyset$.
By Lemma 18 one of the following holds.

1. There exists $\hat{R}$ such that $A, B \vdash P \mid Q \xrightarrow{\ell} \hat{R}$ and $R \equiv \textbf{new } B \textbf{ in } \hat{R}$ By Lemma 16.3 either

   3a (Par) there exists $\hat{P}$ such that $\mathrm{fn}(Q) \subseteq (A, B)$, $(A, B) \vdash P \xrightarrow{\ell} \hat{P}$ and $\hat{R} \equiv \hat{P} \mid Q$.

   By the definition of bisimulation we have $\hat{P}'$ such that $A, B \vdash P' \xrightarrow{\ell} \hat{P}'$ and $\hat{P} \stackrel{\cdot}{\sim}_{(A,B)\cup\mathrm{fn}(\ell)} \hat{P}'$.

   By (Par) $A, B \vdash P' \mid Q' \xrightarrow{\ell} \hat{P}' \mid Q'$.

   By (Res) $A \vdash \textbf{new } B \textbf{ in } (P' \mid Q') \xrightarrow{\ell} \textbf{new } B \textbf{ in } (\hat{P}' \mid Q')$.

   Now, weakening by Lemma 30 gives $Q \stackrel{\cdot}{\sim}_{(A,B)\cup\mathrm{fn}(\ell)} Q'$, so
   $R \equiv \textbf{new } B \textbf{ in } \hat{P} \mid Q \; \mathcal{R}_{A\cup\mathrm{fn}(\ell)} \; \textbf{new } B \textbf{ in } (\hat{P}' \mid Q')$.

3b (Com) there exist x, v, $\hat{P}$ and $\hat{Q}$ such that $\ell = \tau$, $(A,B) \vdash P \xrightarrow{\overline{x}v} \hat{P}$, $(A,B) \vdash Q \xrightarrow{xv} \hat{Q}$, and $\hat{R} \equiv \textbf{new } \{v\} - (A,B) \textbf{ in } (\hat{P} \mid \hat{Q})$.

By the definition of bisimulation we have $\hat{P}'$ and $\hat{Q}'$ such that $A,B \vdash P' \xrightarrow{\overline{x}v} \hat{P}'$, $A,B \vdash Q' \xrightarrow{xv} \hat{Q}'$, $\hat{P} \mathrel{\dot{\sim}}_{(A,B)\cup\{x,v\}} \hat{P}'$, and $\hat{Q} \mathrel{\dot{\sim}}_{(A,B)\cup\{x,v\}} \hat{Q}'$.

By (Com) and (Res) $A \vdash \textbf{new } B \textbf{ in } P' \mid Q' \xrightarrow{\tau} R'$ where $R' = \textbf{new } B \textbf{ in new } \{v\} - A, B \textbf{ in } \hat{P}' \mid \hat{Q}'$.

Finally, we have $R \equiv \mathcal{R}_A\ R'$.

2. there exist $y \in A$, $\hat{x} \notin A$, $x \in B$ and $\hat{R}$ such that $\ell = \overline{y}\hat{x}$, $A,B \vdash P \mid Q \xrightarrow{\overline{y}x} \hat{R}$, and $R \equiv \{\hat{x}/x\}\textbf{new } B - x \textbf{ in } \hat{R}$.

   We have $\hat{P}$ such that $A,B \vdash P \xrightarrow{\overline{y}x} \hat{P}$ and $\hat{R} \equiv \hat{P} \mid Q$.

   By the definition of bisimulation we have $\hat{P}'$ such that $A,B \vdash P' \xrightarrow{\overline{y}x} \hat{P}'$ and $\hat{P} \mathrel{\dot{\sim}}_{A,B} \hat{P}'$.

   By (Par), (Open) and (Res) $A \vdash \textbf{new } B \textbf{ in } P' \mid Q' \xrightarrow{\overline{y}x} R''$ where $R'' = \textbf{new } B - x \textbf{ in } \hat{P}' \mid \hat{Q}'$.

   By Lemma 11 $A \vdash \textbf{new } B \textbf{ in } P' \mid Q' \xrightarrow{\overline{y}\hat{x}} \{{}^{\hat{x}}\!/\!_{x}\}R''$.

   Finally, we have $R \equiv \{{}^{\hat{x}}\!/\!_{x}\}\textbf{new } B - x \textbf{ in } \hat{P} \mid Q$ and $(\textbf{new } B - x \textbf{ in } \hat{P} \mid Q)\ \mathcal{R}_{A,x}\ (\textbf{new } B - x \textbf{ in } \hat{P}' \mid Q')$.

□

Lemma 32 (Asynchrony) *If $A \vdash P \xrightarrow{\overline{z}v} Q \xrightarrow{zv} R$ then $A \vdash P \xrightarrow{\tau} \textbf{new } \{v\} - A \textbf{ in } R$.*

Proof By Lemma 24 $P \equiv \textbf{new } \{v\} - A \textbf{ in } (\overline{z}v \mid Q)$. By Lemma 25 there exist B, p, Q_1 and Q_2 such that $B \cap (A \cup \text{fn}(zv) \cup \text{fn}(zv)) = \{\}$, $Q \equiv \textbf{new } B \textbf{ in } (zp.Q_1 \mid Q_2)$ and $R \equiv \textbf{new } B \textbf{ in } (\{{}^{v}\!/\!_{p}\}Q_1 \mid Q_2)$. Combining these $P \equiv \textbf{new } \{v\} - A \textbf{ in new } B \textbf{ in } (\overline{z}v \mid zp.Q_1 \mid Q_2)$, so by the transition rules $A \vdash P \xrightarrow{\tau} \textbf{new } \{v\} - A \textbf{ in } R$. □

Lemma 33 (Substitution – Bisimulation) *If $P \mathrel{\dot{\sim}}_A Q$ and $\sigma : A \to B$ then $\sigma P \mathrel{\dot{\sim}}_B \sigma Q$.*

Proof Define $\mathcal{R}$ by

$$\mathcal{R}_B \quad = \quad \{\, \sigma P, \sigma P' \mid \exists A \,.\, \sigma : A \to B \wedge P \mathrel{\dot{\sim}}_A P' \,\}$$

We check $\mathcal{R}$ is a bisimulation up to $\equiv$. Suppose $B \vdash \sigma P \xrightarrow{\ell} Q$.

Case $\ell = \overline{x}v$ with $v \in B$.

By Lemma 24 $\sigma P \equiv \overline{x}v \mid Q$.

There exist $\hat{x}$, $\hat{v}$ and $\hat{Q}$ (all above A) such that $P \equiv \overline{\hat{x}}\hat{v} \mid \hat{Q}$, $\sigma\hat{x} = x$, $\sigma\hat{v} = v$, and $\sigma\hat{Q} = Q$.

Note that $A \cup \{\hat{x}, \hat{v}\} = A$ and $B \cup \{x, v\} = B$.

By the transition rules $A \vdash P \xrightarrow{\overline{\hat{x}}\hat{v}} \hat{Q}$.

By $P \mathrel{\dot{\sim}}_A P'$ we have $\hat{Q}'$ such that $A \vdash P' \xrightarrow{\overline{\hat{x}}\hat{v}} \hat{Q}'$ and $\hat{Q} \mathrel{\dot{\sim}}_A \hat{Q}'$.

By Lemma 10 $B \vdash \sigma P' \xrightarrow{\overline{x}v} \sigma\hat{Q}'$.

It remains only to note $Q = \sigma\hat{Q}\ \mathcal{R}_B\ \sigma\hat{Q}'$.

Case $\ell = \overline{x}v$ with $v \notin B$.

By Lemma 24 $\sigma P \equiv \textbf{new } v \textbf{ in } \overline{x}v \mid Q$.

There exist $\hat{x} \in A$, $\hat{v} \notin A$ and $\hat{Q}$ above $A, \hat{v}$ such that $P \equiv \textbf{new } \hat{v} \textbf{ in } \overline{\hat{x}}\hat{v} \mid \hat{Q}$, $\sigma\hat{x} = x$, and $\hat{\sigma}\hat{Q} = Q$, where $\hat{\sigma} = \sigma + (\hat{v} \mapsto v) : A, \hat{v} \to B, v$.

By the transition rules $A \vdash P \xrightarrow{\overline{\hat{x}}\hat{v}} \hat{Q}$.

By $P \dot{\sim}_A P'$ we have $\hat{Q}'$ such that $A \vdash P' \xrightarrow{\overline{\hat{x}}\hat{v}} \hat{Q}'$ and $\hat{Q} \dot{\sim}_{A \cup \{\hat{x}, \hat{v}\}} \hat{Q}'$.

By Lemma 10 $B \vdash \sigma P' \xrightarrow{\overline{x}v} \hat{\sigma}\hat{Q}'$.

It remains only to note $Q = \hat{\sigma}\hat{Q} \; \mathcal{R}_{B \cup \{x,v\}} \; \hat{\sigma}\hat{Q}'$.

Case $\ell = xv$.

By Lemma 25 there exist C, p, Q_1 and Q_2 such that $C \cap (B \cup \text{fn}(xv)) = \{\}$, $\sigma P \equiv \textbf{new } C \textbf{ in } (xp.Q_1 \mid Q_2)$ and $Q \equiv \textbf{new } C \textbf{ in } (\{^v/_p\}Q_1 \mid Q_2)$.

There exist $\hat{x} \in A$, a set $\hat{C}$ disjoint from A, $\phi : \hat{C} \to C$ a bijection, $\hat{p} \notin A, \hat{C}$, $\hat{Q}_1$ above $A, \hat{C}, \hat{p}$, and $\hat{Q}_2$ above $A, \hat{C}$ such that $P \equiv \textbf{new } \hat{C} \textbf{ in } (\hat{x}\hat{p}.\hat{Q}_1 \mid \hat{Q}_2)$, $\sigma\hat{x} = x$, $(\sigma + \phi + (\hat{p} \mapsto p))\hat{Q}_1 = Q_1$, $(\sigma + \phi)\hat{Q}_2 = Q_2$.

If $v \in \text{ran}(\sigma)$ then take some $\hat{v} \in A$ such that $\sigma\hat{v} = v$ and let $\hat{\sigma} = \sigma : A \to B$, otherwise take some $\hat{v} \notin A, \hat{C}$ and let $\hat{\sigma} = \sigma + (\hat{v} \mapsto v) : A, \hat{v} \to B \cup \{v\}$.

By the transition rules $A \vdash P \xrightarrow{\hat{x}\hat{v}} \textbf{new } \hat{C} \textbf{ in } (\{^{\hat{v}}/_{\hat{p}}\}\hat{Q}_1 \mid \hat{Q}_2)$.

By $P \dot{\sim}_A P'$ we have $\hat{Q}'$ such that $A \vdash P' \xrightarrow{\hat{x}\hat{v}} \hat{Q}'$ and

$$\textbf{new } \hat{C} \textbf{ in } (\{^{\hat{v}}/_{\hat{p}}\}\hat{Q}_1 \mid \hat{Q}_2) \dot{\sim}_{A \cup \{\hat{x}, \hat{v}\}} \hat{Q}'$$

By Lemma 10 $B \vdash \sigma P' \xrightarrow{xv} \hat{\sigma}\hat{Q}'$.

Now

$$\begin{array}{rcl} Q & \equiv & \textbf{new } C \textbf{ in } (\{^v/_p\}Q_1 \mid Q_2) \\ & = & \hat{\sigma}\textbf{new } \hat{C} \textbf{ in } (\{^{\hat{v}}/_{\hat{p}}\}\hat{Q}_1 \mid \hat{Q}_2) \\ & \mathcal{R}_{B \cup \{x,v\}} & \hat{\sigma}\hat{Q}' \end{array}$$

Case $\ell = \tau$.

By Lemma 27 there exist C, x, v, p, Q_1, Q_2 such that $C \cap (B \cup A) = \emptyset$, $p \notin B \cup A \cup C$, $\sigma P \equiv \textbf{new } C \textbf{ in } (\overline{x}v \mid xp.Q_1 \mid Q_2)$ and $Q \equiv \textbf{new } C \textbf{ in } (\{^v/_p\}Q_1 \mid Q_2)$.

There exist $\hat{x}$, $\hat{\hat{x}}$ and $\hat{v}$ in A, C, $\hat{Q}_1$ above A, C, p, and $\hat{Q}_2$ above A, C such that $P \equiv \textbf{new } C \textbf{ in } (\overline{\hat{x}}\hat{v} \mid \hat{\hat{x}}p.\hat{Q}_1 \mid \hat{Q}_2)$, $(\sigma + id_C)\hat{x} = x$, $(\sigma + id_C)\hat{\hat{x}} = x$, $(\sigma + id_C)\hat{v} = v$, $(\sigma + id_{C,p})\hat{Q}_1 = Q_1$, $(\sigma + id_C)\hat{Q}_2 = Q_2$. Note that either $v = \hat{v} \in C$ or else $v \in B$, $\hat{v} \in A$ and $\sigma(\hat{v}) = v$.

Case $\hat{x} = \hat{\hat{x}}$ By the transition rules $A \vdash P \xrightarrow{\tau} \hat{Q}$ where $\hat{Q} = \textbf{new } C \textbf{ in } (\{^{\hat{v}}/_p\}\hat{Q}_1 \mid \hat{Q}_2)$.

Note $\sigma\hat{Q} \equiv Q$.

By $P \dot{\sim}_A P'$ we have $\hat{Q}'$ such that $A \vdash P' \xrightarrow{\tau} \hat{Q}'$ and $\hat{Q} \dot{\sim}_A \hat{Q}'$.

By Lemma 10 $B \vdash \sigma P' \xrightarrow{\tau} \sigma\hat{Q}'$.

It remains only to note $Q \equiv \sigma\hat{Q} \; \mathcal{R}_B \; \sigma\hat{Q}'$.

Case $\hat{x} \neq \hat{\hat{x}}$ As the ranges of σ and id_C are disjoint we have $\hat{x} \in A$ and $\hat{\hat{x}} \in A$.
By the transition rules

$$\begin{array}{rcl} A \vdash P & \xrightarrow{\overline{\hat{\hat{x}}}\hat{v}} & \mathbf{new}\ C - \{\hat{v}\}\ \mathbf{in}\ (\hat{x}p.\hat{Q}_1 \mid \hat{Q}_2) \\ & \xrightarrow{\hat{x}\hat{v}} & \mathbf{new}\ C - \{\hat{v}\}\ \mathbf{in}\ (\{^{\hat{v}}/_{p}\}\hat{Q}_1 \mid \hat{Q}_2) \end{array}$$

By $P \dot{\sim}_A P'$ we have $\hat{Q}'$ such that

$$\begin{array}{rcl} A \vdash P' & \xrightarrow{\overline{\hat{\hat{x}}}\hat{v}} & \\ & \xrightarrow{\hat{x}\hat{v}} & \hat{Q}' \end{array}$$

and

$$\mathbf{new}\ C - \{\hat{v}\}\ \mathbf{in}\ (\{^{\hat{v}}/_{p}\}\hat{Q}_1 \mid \hat{Q}_2) \dot{\sim}_{A \cup \{\hat{v}\}} \hat{Q}'$$

By Lemma 10, taking $g = \{v/\hat{v}\}$ if $v = \hat{v} \in C$ and $g = \emptyset$ if $\hat{v} \in A$,

$$\begin{array}{rcl} B \vdash \sigma P' & \xrightarrow{\overline{x}v} & \\ & \xrightarrow{xv} & (\sigma + g)\hat{Q}' \end{array}$$

By Lemma 32 $B \vdash \sigma P' \xrightarrow{\tau} \mathbf{new}\ \{v\} - B\ \mathbf{in}\ (\sigma + g)\hat{Q}'$.
Case $v = \hat{v} \in C$**.** We have

$$\begin{array}{rcll} Q & \equiv & \mathbf{new}\ C\ \mathbf{in}\ \{^{v}/_{p}\}Q_1 \mid Q_2 & \\ & \equiv & \mathbf{new}\ C\ \mathbf{in}\ \{^{v}/_{p}\}(\sigma + id_{C,p})\hat{Q}_1 \mid (\sigma + id_C)\hat{Q}_2 & \\ & \equiv & \sigma\mathbf{new}\ C\ \mathbf{in}\ \{^{v}/_{p}\}\hat{Q}_1 \mid \hat{Q}_2 & \\ & \equiv & \sigma\mathbf{new}\ v\ \mathbf{in}\ \mathbf{new}\ C - v\ \mathbf{in}\ \{^{v}/_{p}\}\hat{Q}_1 \mid \hat{Q}_2 & \\ & \mathcal{R}_B & \sigma\mathbf{new}\ v\ \mathbf{in}\ \hat{Q}' & \text{Using Lemma 31 for } \mathbf{new}\ v\ \mathbf{in}\ \text{ cong} \\ & \equiv & \mathbf{new}\ v - B\ \mathbf{in}\ (\sigma + g)\hat{Q}' & \end{array}$$

Case $\hat{v} \in A$**.** We have

$$\begin{array}{rcl} Q & \equiv & \mathbf{new}\ C\ \mathbf{in}\ \{^{v}/_{p}\}Q_1 \mid Q_2 \\ & \equiv & \mathbf{new}\ C\ \mathbf{in}\ \{^{v}/_{p}\}(\sigma + id_{C,p})\hat{Q}_1 \mid (\sigma + id_C)\hat{Q}_2 \\ & \equiv & \mathbf{new}\ C\ \mathbf{in}\ (\sigma + id_C)\{^{\hat{v}}/_{p}\}\hat{Q}_1 \mid (\sigma + id_C)\hat{Q}_2 \\ & \equiv & \sigma\mathbf{new}\ C\ \mathbf{in}\ \{^{\hat{v}}/_{p}\}\hat{Q}_1 \mid \hat{Q}_2 \\ & \mathcal{R}_B & \sigma\hat{Q}' \\ & \equiv & \mathbf{new}\ v - B\ \mathbf{in}\ (\sigma + g)\hat{Q}' \end{array}$$

□

Lemma 34 (Congruence – Input Prefix) *If $P \dot{\sim}_{A,p} Q$ and $x \in A$ then $xp.P \dot{\sim}_A xp.Q$.*

PROOF We check the union of $\dot{\sim}$ and the pair $(xp.P, xp.Q)$ at A is a bisimulation up to $\equiv$.

The cases $\hat{P} \dot{\sim}_B \hat{Q}$ are routine.

Suppose $A \vdash xp.P \xrightarrow{\ell} P'$.

By Lemma 16 there exist v such that $\ell = xv$ and $P' \equiv \{^{v}/_{p}\}P$.

By (INP) $A \vdash xp.Q \xrightarrow{xv} \{^{v}/_{p}\}Q$.

By Lemma 33 $\{^{v}/_{p}\}P \dot{\sim}_{A \cup \{v\}} \{^{v}/_{p}\}Q$. □

Theorem 3 *Bisimulation $\dot{\sim}$ is an indexed congruence.*

PROOF Immediate from Lemmas 31 and 34. □

5.4 Type Soundness and Subject Reduction

Lemma 35 (Weakening – Values) *If* $\Gamma \vdash v : T'$ *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, x : T \vdash v : T'$.

PROOF By induction on type derivations. □

Lemma 36 (Weakening – Processes) *If* $\Gamma \vdash P$ **proc** *and* $x \notin \mathrm{dom}(\Gamma)$ *then* $\Gamma, x : T \vdash P$ **proc**.

PROOF By induction on type derivations, using Lemma 35. □

Lemma 37 (Strengthening – Values) *If* $\Gamma, x : T \vdash v : T'$ *and* $x \notin \mathrm{fn}(v)$ *then* $\Gamma \vdash v : T'$.

PROOF By induction on type derivations. □

Lemma 38 (Strengthening – Processes) *If* $\Gamma, x : T \vdash P$ **proc** *and* $x \notin \mathrm{fn}(P)$ *then* $\Gamma \vdash P$ **proc**.

PROOF By induction on type derivations , using Lemma 37. □

Lemma 39 (Type Soundness of Structural Congruence) *If* $P \equiv Q$ *then* $\Gamma \vdash P$ **proc** *iff* $\Gamma \vdash Q$ **proc**.

PROOF By induction on the derivation of $P \equiv Q$.

(ParId) $\Gamma \vdash P \,|\, 0$ **proc** iff $\Gamma \vdash P$ **proc** $\wedge\, \Gamma \vdash 0$ **proc** iff $\Gamma \vdash P$ **proc**.

(ParComm),(ParAssoc) Similar.

(NewExtrude) $P \,|\, \mathbf{new}\ \hat{x} : T\ \mathbf{in}\ \hat{Q} \equiv \mathbf{new}\ \hat{x} : T\ \mathbf{in}\ (P \,|\, \hat{Q})$ with $\hat{x} \notin \mathrm{fn}(P)$.

Take Q and $x \notin \mathrm{dom}(\Gamma) \cup \mathrm{fn}(P)$ such that $P \,|\, \mathbf{new}\ x : T\ \mathbf{in}\ Q = P \,|\, \mathbf{new}\ \hat{x} : T\ \mathbf{in}\ \hat{Q}$ and $\mathbf{new}\ x : T\ \mathbf{in}\ (P \,|\, Q) = \mathbf{new}\ \hat{x} : T\ \mathbf{in}\ (P \,|\, \hat{Q})$. We have

$$\begin{array}{rcll}
\Gamma \vdash P \,|\, \mathbf{new}\ x : T\ \mathbf{in}\ Q\ \mathbf{proc} & \iff & \Gamma \vdash P\ \mathbf{proc} \wedge \Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ Q\ \mathbf{proc} & \\
& \iff & \Gamma \vdash P\ \mathbf{proc} \wedge \Gamma, x : T \vdash Q\ \mathbf{proc} & \\
& \iff & \Gamma, x : T \vdash P\ \mathbf{proc} \wedge \Gamma, x : T \vdash Q\ \mathbf{proc} & (*) \\
& \iff & \Gamma, x : T \vdash P \,|\, Q\ \mathbf{proc} & \\
& \iff & \Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ P \,|\, Q\ \mathbf{proc} &
\end{array}$$

Here $(*)$ is by Lemmas 36 and 38.

(NewComm) Straightforward.

(CongIn),(CongPar),(CongNew) By the induction hypothesis and the corresponding typing rule.

(Refl),(Sym),(Tran) By the induction hypothesis and the corresponding properties of iff.

□

Lemma 40 (Injective Substitution – Reductions) *If* $P \to Q$ *and* $f : \mathrm{fn}(P) \to \mathcal{N}$ *is injective then* $fP \to fQ$.

We define well-typed substitutions as follows. Say $\Gamma \vdash \sigma : \Delta$ iff $\mathrm{dom}(\sigma) = \mathrm{dom}(\Delta)$ and $\forall x \in \mathrm{dom}(\Delta)\ .\ \Gamma \vdash \sigma(x) : \Delta(x)$.

Lemma 41 (Good Substitutions) *If* Γ atomic *and* $\Gamma \vdash v : T$ *and* $\vdash p : T \rhd \Delta$ *then* $\{{}^{v}/_{p}\}$ *is defined and* $\Gamma \vdash \{{}^{v}/_{p}\} : \Delta$.

PROOF By induction on the two typing derivations. Consider the last rule of the pattern judgement.

(VAR) $\vdash x : T \rhd x : T$. Clearly $\{{}^{v}/_{x}\} = \{x \mapsto v\}$, $\mathrm{dom}(\{x \mapsto v\}) = \mathrm{dom}(x : T)$, and $\Gamma \vdash \{{}^{v}/_{x}\} x : T$.

(TUPLE) $\vdash \langle p_1 \mathinner{..} p_k \rangle : \langle T_1 \mathinner{..} T_k \rangle \rhd \Delta_1, .., \Delta_k$. As Γ atomic the last rule of the value judgement must either be the value (TUPLE) rule, with conclusion $\Gamma \vdash \langle v_1 \mathinner{..} v_k \rangle : \langle T_1 \mathinner{..} T_k \rangle$, or the value (VAR) rule, with $k = 0$. In the first case, by the induction hypothesis for each i we have $\{{}^{v_i}/_{p_i}\}$ defined and $\Gamma \vdash \{{}^{v_i}/_{p_i}\} : \Delta_i$. By the definition of substitution $\{{}^{\langle v_1 \mathinner{..} v_k \rangle}/_{\langle p_1 \mathinner{..} p_k \rangle}\}$ is defined and equal to $\bigcup_{i \in 1..k} \{{}^{v_i}/_{p_i}\}$. As for each i $\mathrm{dom}(\{{}^{v_i}/_{p_i}\}) = \mathrm{dom}(\Delta_i)$, we have $\mathrm{dom}(\bigcup_{i \in 1..k} \{{}^{v_i}/_{p_i}\}) = \mathrm{dom}(\Delta_1, .., \Delta_k)$. To check $\Gamma \vdash \{{}^{\langle v_1 \mathinner{..} v_k \rangle}/_{\langle p_1 \mathinner{..} p_k \rangle}\} : \Delta_1, .., \Delta_k$ it remains only to observe that for each $z : T' \in \Delta_1, .., \Delta_k$ there is an i with $z : T' \in \Delta_i$, hence $\Gamma \vdash \{{}^{v_i}/_{p_i}\} z : T'$, hence $\Gamma \vdash (\bigcup_{i \in 1..k} \{{}^{v_i}/_{p_i}\}) z : T'$.

In the second case, by the definition of substitution $\{{}^{\langle\rangle}/_{\langle\rangle}\}$ is defined and equal to $\emptyset$, and $\Gamma \vdash \emptyset : \emptyset$.

Note that this result requires that the range of Γ contains no non-unit tuple types. $\square$

Lemma 42 (Substitution – Values) *If* $\Gamma, \Delta \vdash v : T$ *and* $\Gamma \vdash \sigma : \Delta$ *then* $\Gamma \vdash \sigma v : T$.

PROOF By induction on the value typing derivation.

(VAR) $\Gamma, \Delta \vdash x : T$.

Case $x : T \in \Gamma$. By the second premise the domain of σ coincides with that of Δ and hence is disjoint from Γ, so $\sigma(x) = x$, so $\Gamma \vdash \sigma(x) : T$.

Case $x : T \in \Delta$. By the second premise $\Gamma \vdash \sigma(x) : T$.

(TUPLE) $\Gamma, \Delta \vdash \langle v_1 \mathinner{..} v_k \rangle : \langle T_1 \mathinner{..} T_k \rangle$. By the induction hypothesis for each i we have $\Gamma \vdash \sigma(v_i) : T_i$ hence $\Gamma \vdash \sigma(\langle v_1 \mathinner{..} v_k \rangle) : \langle T_1 \mathinner{..} T_k \rangle$

$\square$

Lemma 43 (Substitution – Processes) *If* $\Gamma, \Delta \vdash P$ **proc** *and* $\Gamma \vdash \sigma : \Delta$ *then* σP *is defined and* $\Gamma \vdash \sigma P$ **proc**.

PROOF We prove both parts simultaneously by induction on the type derivation for P. For the second part we give two instances of each typing rule; in each case showing that the premises of the right-hand instance follow from those of the left-hand instance.

(OUT)

$$\textsc{Out}\ \frac{\begin{array}{c}\Gamma, \Delta \vdash x : \mathbf{chan}\, T \\ \Gamma, \Delta \vdash v : T\end{array}}{\Gamma, \Delta \vdash \overline{x} v \ \mathbf{proc}} \qquad \textsc{Out}\ \frac{\begin{array}{c}\Gamma \vdash \sigma(x) : \mathbf{chan}\, T \\ \Gamma \vdash \sigma(v) : T\end{array}}{\Gamma \vdash \overline{\sigma(x)} \sigma(v) \ \mathbf{proc}}$$

By Lemma 42 both premises of the right-hand-side instance hold. As $\Gamma \vdash \sigma(x) : \mathbf{chan}\, T$ we have that $\sigma(x)$ is a name, so $\sigma(\overline{x} v)$ is defined.

(In)

$$\textsc{In}\ \frac{\Gamma,\Delta \vdash x : \mathbf{chan}\, T \quad \vdash p : T \rhd \Theta \quad \Gamma,\Delta,\Theta \vdash P\ \mathbf{proc}}{\Gamma,\Delta \vdash xp.P\ \mathbf{proc}} \qquad \textsc{In}\ \frac{\Gamma \vdash \sigma(x) : \mathbf{chan}\, T \quad \vdash p : T \rhd \Theta \quad \Gamma,\Theta \vdash \sigma(P)\ \mathbf{proc}}{\Gamma \vdash \sigma(x)p.\sigma(P)\ \mathbf{proc}}$$

By Lemma 42 the first premise of the right-hand-side instance holds. The second is immediate, and clearly $\Gamma,\Theta \vdash \sigma : \Delta$, so the third follows from the induction hypothesis, which also tells us that σP is defined. As $\Gamma \vdash \sigma(x) : \mathbf{chan}\, T$ we have that $\sigma(x)$ is a name, so $\sigma(xp.P)$ is defined.

(Par)

$$\textsc{Par}\ \frac{\Gamma,\Delta \vdash P\ \mathbf{proc} \quad \Gamma,\Delta \vdash Q\ \mathbf{proc}}{\Gamma,\Delta \vdash P \mid Q\ \mathbf{proc}} \qquad \textsc{Par}\ \frac{\Gamma \vdash \sigma P\ \mathbf{proc} \quad \Gamma \vdash \sigma Q\ \mathbf{proc}}{\Gamma \vdash \sigma P \mid \sigma Q\ \mathbf{proc}}$$

The two premises of the right-hand-side instance are immediate from the induction hypothesis, and $\sigma(P \mid Q)$ is defined as σP and σQ are defined.

(Nil)

$$\textsc{Nil}\ \frac{}{\Gamma,\Delta \vdash 0\ \mathbf{proc}} \qquad \textsc{Nil}\ \frac{}{\Gamma \vdash 0\ \mathbf{proc}}$$

Trivial.

(Res)

$$\textsc{Res}\ \frac{\Gamma,\Delta,x : \mathbf{chan}\, T \vdash P\ \mathbf{proc}}{\Gamma,\Delta \vdash \mathbf{new}\ x : \mathbf{chan}\ T\ \mathbf{in}\ P\ \mathbf{proc}} \qquad \textsc{Res}\ \frac{\Gamma, x : \mathbf{chan}\, T \vdash \sigma P\ \mathbf{proc}}{\Gamma \vdash \mathbf{new}\ x : \mathbf{chan}\ T\ \mathbf{in}\ \sigma P\ \mathbf{proc}}$$

Clearly $\Gamma, x : \mathbf{chan}\, T \vdash \sigma : \Delta$, so the premise follows from the induction hypothesis, which also tells us that σP is defined and hence that $\sigma(\mathbf{new}\ x : \mathbf{chan}\ T\ \mathbf{in}\ P)$ is.

□

Theorem 4 (Subject Reduction) *If* $\Gamma \vdash P\ \mathbf{proc}$ *and* Γ atomic *and* $P \to Q$ *then* $\Gamma \vdash Q\ \mathbf{proc}$.

Proof By induction on the derivation of $P \to Q$.

(Com) Suppose (being cavalier about alpha equivalence) that the names of p are disjoint from dom(Γ). By the typing rules we have T such that $\Gamma \vdash c : \mathbf{chan}\, T$, $\Gamma \vdash v : T$, $\vdash p : T \rhd \Delta$, and $\Gamma,\Delta \vdash P\ \mathbf{proc}$. By Lemma 41 $\{^v/_p\}$ is defined and $\Gamma \vdash \{^v/_p\} : \Delta$. By Lemma 43 $\{^v/_p\}P$ is defined and $\Gamma \vdash \{^v/_p\}P\ \mathbf{proc}$.

(Par) By the typing rules we have $\Gamma \vdash P\ \mathbf{proc}$ and $\Gamma \vdash Q\ \mathbf{proc}$. By the induction hypothesis $\Gamma \vdash P'\ \mathbf{proc}$. Using the (Par) typing rule $\Gamma \vdash P' \mid Q\ \mathbf{proc}$.

(Res) By the typing rules we have $\hat{P}$ and $\hat{x}$ and T_0 such that $T = \mathbf{chan}\, T_0$ and $\mathbf{new}\ \hat{x} : T\ \mathbf{in}\ \hat{P} = \mathbf{new}\ x : T\ \mathbf{in}\ P$ and $\hat{x} \notin$ dom(Γ) and $\Gamma, \hat{x} : T \vdash \hat{P}\ \mathbf{proc}$. By Lemma 40 $\hat{P} \to \{^{\hat{x}}/_x\}P'$. By the induction hypothesis (noting that $\Gamma, \hat{x} : T$ atomic) we have $\Gamma, \hat{x} : T \vdash \{^{\hat{x}}/_x\}P'\ \mathbf{proc}$. By (Res) $\Gamma \vdash \mathbf{new}\ \hat{x} : T\ \mathbf{in}\ \{^{\hat{x}}/_x\}P'\ \mathbf{proc}$. But then $\Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ P'\ \mathbf{proc}$.

(STRUCT) By Lemma 39 $\Gamma \vdash P'$ **proc**. By the induction hypothesis $\Gamma \vdash P''$ **proc**. By Lemma 39 again $\Gamma \vdash P'''$ **proc**.

□

Theorem 5 (Absence of Runtime Errors) *If* $\Gamma \vdash P$ **proc** *and* Γ atomic *then* $\neg(P\ \mathrm{err})$.

PROOF We show $(\Gamma\ \text{atomic} \wedge \Gamma \vdash P\ \mathbf{proc} \wedge P\ \mathrm{err}) \implies \mathsf{ff}$, by induction on derivations of P err.

(ERR COM) Suppose the names of p are disjoint from $\mathrm{dom}(\Gamma)$. By the typing rules we have T such that $\Gamma \vdash c : \mathbf{chan}\, T$, $\Gamma \vdash v : T$, $\vdash p : T \rhd \Delta$, and $\Gamma, \Delta \vdash P$ **proc**. By Lemma 41 $\{^v/_p\}$ is defined and $\Gamma \vdash \{^v/_p\} : \Delta$. By Lemma 43 $\{^v/_p\}P$ is defined.

(ERR PAR) By the typing rules we have $\Gamma \vdash P$ **proc** and $\Gamma \vdash Q$ **proc**. By the induction hypothesis we have ff.

(ERR RES) Suppose $x \notin \mathrm{dom}(\Gamma)$. By the typing rules we have T_0 such that $T = \mathbf{chan}\, T_0$ and $\Gamma, x : T \vdash P$ **proc**. We have $\Gamma, x : T$ atomic so by the induction hypothesis we have ff.

(ERR STRUCT) By Lemma 39 $\Gamma \vdash P$ **proc** so by the induction hypothesis we have ff.

□

Theorem 6 (Subject Reduction – Typed LTS) *If* $\Gamma \vdash P \xrightarrow[\Delta]{\ell} Q$ *then*

1. $\Gamma, \Delta \vdash Q$ **proc**
2. *if* $\ell = \overline{x}v$ *or* $\ell = xv$ *then there is* T *such that* $\Gamma \vdash x : \mathbf{chan}\, T$ *and* $\Gamma, \Delta \vdash v : T$.

PROOF By induction on derivations of transitions.

(OUT) Immediate.

(IN) Using the fact that typing is unique, and being cavalier about alpha equivalence, we have the following with Γ, Δ and Θ disjoint.

$$\textsc{In}\,\frac{\begin{array}{l}\Gamma \vdash x : \mathbf{chan}\, T \\ \Gamma, \Delta \vdash v : T \\ \mathrm{dom}(\Delta) \subseteq \mathrm{fn}(v) \\ \Delta\ \text{extensible}\end{array}}{\Gamma \vdash xp.P \xrightarrow[\Delta]{xv} \{^v/_p\}P} \qquad\qquad \textsc{In}\,\frac{\begin{array}{l}\Gamma \vdash x : \mathbf{chan}\, T \\ \vdash p : T \rhd \Theta \\ \Gamma, \Theta \vdash P\ \mathbf{proc}\end{array}}{\Gamma \vdash xp.P\ \mathbf{proc}}$$

For Part (1), by Δ extensible we have Γ, Δ atomic. By Lemma 41 $\{^v/_p\}$ is defined and $\Gamma, \Delta \vdash \{^v/_p\} : \Theta$. By Lemma 36 $\Gamma, \Delta, \Theta \vdash P$ **proc**. By Lemma 43 $\{^v/_p\}P$ is defined and $\Gamma, \Delta \vdash \{^v/_p\}P$ **proc**. Part (2) is immediate.

(PAR) By the induction hypothesis $\Gamma, \Delta \vdash P'$ **proc**. By the typing rules $\Gamma \vdash Q$ **proc**. By Lemma 36 $\Gamma, \Delta \vdash Q$ **proc**. By (PAR) $\Gamma, \Delta \vdash P' \mid Q$ **proc**. Part (2) is immediate from the induction hypothesis.

(COM) By the induction hypothesis $\Gamma, \Delta \vdash P'$ **proc** and $\Gamma, \Delta \vdash Q'$ **proc**. By (PAR) and (RES) $\Gamma \vdash$ **new** Δ **in** $P' \mid Q'$ **proc**. Part (2) is vacuous.

(Res) By the typing rules $T = \mathbf{chan}\,T'$ for some T'. By the induction hypothesis and (Res) $\Gamma \vdash \mathbf{new}\ x : T\ \mathbf{in}\ P'\ \mathbf{proc}$. Also by the induction hypothesis, if $\ell = \overline{y}v$ or $\ell = yv$ then there is T'' such that $\Gamma, x : T \vdash y : \mathbf{chan}\,T''$ and $\Gamma, x : T, \Delta \vdash v : T''$. By Lemma 35 $\Gamma \vdash y : \mathbf{chan}\,T''$ and $\Gamma, \Delta \vdash v : T''$.

(Open) By the induction hypothesis $\Gamma, x : T, \Delta \vdash P'\ \mathbf{proc}$ so $\Gamma, \Delta, x : T \vdash P'\ \mathbf{proc}$. Also by the induction hypothesis, there is T'' such that $\Gamma, x : T \vdash y : \mathbf{chan}\,T''$ and $\Gamma, x : T, \Delta \vdash v : T''$. By $x \neq y$ and Lemma 35 $\Gamma \vdash y : \mathbf{chan}\,T''$.

(Struct) By the induction hypothesis and Lemma 39.

□

Bibliography

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proceedings of the Fourth ACM Conference on Computer and Communications Security, Zürich*, pages 36–47. ACM Press, April 1997.

[AP94] R. M. Amadio and S. Prasad. Localities and failures. In P. S. Thiagarajan, editor, *Proceedings of 14^{th} FSTTCS. LNCS 880*, pages 205–216. Springer-Verlag, 1994.

[BB92] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96:217–248, 1992.

[Bou92] Gérard Boudol. Asynchrony and the π-calculus (note). Rapport de Recherche 1702, INRIA Sofia-Antipolis, May 1992.

[CG98] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Proc. of Foundations of Software Science and Computation Structures (FoSSaCS), ETAPS'98, LNCS 1378*, pages 140–155, March 1998.

[CS00] Gian Luca Cattani and Peter Sewell. Models for name-passing processes: Interleaving and causal. In *Proceedings of LICS 2000: the 15th IEEE Symposium on Logic in Computer Science (Santa Barbara)*, pages 322–333, June 2000.

[FG96] Cédric Fournet and Georges Gonthier. The reflexive CHAM and the join-calculus. In *Proceedings of the 23rd POPL*, pages 372–385. ACM press, January 1996.

[FGL+96] Cédric Fournet, Georges Gonthier, Jean-Jacques Lévy, Luc Maranget, and Didier Rémy. A calculus of mobile agents. In *Proceedings of CONCUR '96. LNCS 1119*, pages 406–421. Springer-Verlag, August 1996.

[Fou98] Cédric Fournet. *The Join-Calculus: A Calculus for Distributed Mobile Programming.* PhD thesis, École Polytechnique, Paris, 1998.

[Gla90] R. J. van Glabbeek. The linear time – branching time spectrum. In *Proceedings of CONCUR '90, LNCS 458*, pages 278–297, 1990.

[Gla93] R. J. van Glabbeek. The linear time – branching time spectrum II; the semantics of sequential systems with silent moves. In *Proceedings of CONCUR'93, LNCS 715*, pages 66–81, 1993.

[HT91] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *Proceedings of ECOOP '91, LNCS 512*, pages 133–147, July 1991.

[Joi] The join-calculus. Papers and implementations available at `http://pauillac.inria.fr/join/`.

[Mil92] Robin Milner. Functions as processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[Mil99] Robin Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, May 1999.

[MPW92] R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I + II. *Information and Computation*, 100(1):1–77, 1992.

[MS92] Robin Milner and Davide Sangiorgi. Barbed bisimulation. In *Proceedings of 19th ICALP. LNCS 623*, pages 685–695, 1992.

[Nee89] R. M. Needham. Names. In S. Mullender, editor, *Distributed Systems*, pages 89–101. Addison-Wesley, 1989.

[Nes] Uwe Nestmann. Calculi for mobile processes. Available at `http://move.to/mobility` and `http://www.cs.auc.dk/mobility/`.

[NP96] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. In *Proceedings of CONCUR '96, LNCS 1119*, pages 179–194, August 1996.

[Pal97] Catuscia Palamidessi. Comparing the expressive power of the synchronous and the asynchronous π-calculus. In *Proceedings of POPL 97*, pages 256–265, 1997.

[PC98] Catuscia Palamidessi and Ilaria Castellani, editors. *EXPRESS '98: Expressiveness in Concurrency (Nice, France, September 7, 1998)*, volume 16.2 of *entcs*. Elsevier Science Publishers, 1998.

[Pie98] Benjamin C. Pierce. Type systems for concurrent calculi, September 1998. Invited tutorial at *CONCUR*, Nice, France. Slides available from `http://www.cis.upenn.edu/~bcpierce/`.

[PT98] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus, March 1998. Version 4.1. Compiler, documentation, demonstration programs, and standard libraries; available from `http://www.cis.upenn.edu/~bcpierce/`.

[PT00] Benjamin C. Pierce and David N. Turner. Pict: A programming language based on the pi-calculus. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.

[Qua99] P. Quaglia. The π-calculus: notes on labelled semantics. *Bulletin of the EATCS*, 68, June 1999. Preliminary version as BRICS Report LS-98-4.

[QW98] P. Quaglia and D. Walker. On encoding pπ in mπ. In *Proc. 18th FSTTCS, LNCS 1530*, pages 42–53. Springer Verlag, 1998.

[RH99] James Riely and Matthew Hennessy. Trust and partial typing in open systems of mobile agents. In *Proceedings of the 26th POPL*, San Antonio, January 1999. ACM Press.

[San93] Davide Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms.* PhD thesis, University of Edinburgh, 1993.

[Sew97] Peter Sewell. On implementations and semantics of a concurrent programming language. In *Proceedings of CONCUR '97. LNCS 1243*, pages 391–405, 1997.

[Sew98] Peter Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proceedings of ICALP '98. LNCS 1443*, pages 695–706. Springer-Verlag, July 1998. See also Technical Report 435, University of Cambridge, 1997.

[SV99] Peter Sewell and Jan Vitek. Secure composition of insecure components. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop. Mordano, Italy*, pages 136–150. IEEE Computer Society, June 1999.

[SV00] Peter Sewell and Jan Vitek. Secure composition of untrusted code: Wrappers and causality types. In *Proceedings of CSFW 00: The 13th IEEE Computer Security Foundations Workshop.*, pages 269–284. IEEE Computer Society, July 2000.

[SWP98] Peter Sewell, Paweł T. Wojciechowski, and Benjamin C. Pierce. Location independence for mobile agents. In *Workshop on Internet Programming Languages, Chicago.* Springer-Verlag, May 1998. Full version as Technical Report 462, University of Cambridge, and in LNCS 1686.

[Tho93] Bent Thomsen. Plain CHOCS. A second generation calculus for higher order processes. *Acta Informatica*, 30(1):1–59, 1993.

[TLK96] Bent Thomsen, Lone Leth, and Tsung-Min Kuo. A Facile tutorial. In *Proceedings of CONCUR '96. LNCS 1119*, pages 278–298. Springer-Verlag, August 1996.

[Tur96] David N. Turner. *The Polymorphic Pi-calculus: Theory and Implementation.* PhD thesis, University of Edinburgh, 1996.

[VC98] Jan Vitek and Guiseppe Castagna. Towards a calculus of mobile computations. In *Workshop on Internet Programming Languages, Chicago*, May 1998. Full version in LNCS 1686.

[Yos96] Nobuko Yoshida. Graph types for monadic mobile processes. In *Proceedings of FSTTCS 96, LNCS*, pages 371–386, 1996.