A Concurrent Object Calculus: Reduction and Typing

Andrew D. Gordon[*]
University of Cambridge Computer Laboratory

Paul D. Hankin
University of Cambridge Computer Laboratory

March 11, 1999

Abstract

We obtain a new formalism for concurrent object-oriented languages by extending Abadi and Cardelli's imperative object calculus with operators for concurrency from the π-calculus and with operators for synchronisation based on mutexes. Our syntax of terms is extremely expressive; in a precise sense it unifies notions of expression, process, store, thread, and configuration. We present a chemical-style reduction semantics, and prove it equivalent to a structural operational semantics. We identify a deterministic fragment that is closed under reduction and show that it includes the imperative object calculus. A collection of type systems for object-oriented constructs is at the heart of Abadi and Cardelli's work. We recast one of Abadi and Cardelli's first-order type systems with object types and subtyping in the setting of our calculus and prove subject reduction. Since our syntax of terms includes both stores and running expressions, we avoid the need to separate store typing from typing of expressions. We translate communication channels and the choice-free asynchronous π-calculus into our calculus to illustrate its expressiveness; the types of read-only and write-only channels are supertypes of read-write channels.

[*] Current affiliation: Microsoft Research

Contents

1 Motivation 1

1.1 Related work 2

1.2 Organisation of the paper 3

2 Concurrent Objects 3

2.1 Primitives for Concurrent Objects 4

2.2 Informal Semantics 5

2.3 Formal Semantics 6

2.4 Examples of Concurrent Objects 9

2.4.1 Imperative objects 9

2.4.2 Concurrent procedure calls 10

3 Synchronisation 11

3.1 Primitives for Synchronisation 12

3.2 Informal Semantics 12

3.3 Formal Semantics 13

3.4 Examples of Synchronisation 13

3.4.1 Mutual exclusion 13

3.4.2 Asynchronous channels 14

3.4.3 Synchronous channels 15

3.4.4 Fork and join 16

4 A Structural Characterisation of Reduction 16

4.1 Well-formed Terms 17

4.2 A Structural Operational Semantics 18

4.3 A Single-Threaded Fragment 22

5 A First-Order Type System 23

5.1 Typing 23

5.2 Examples of Typing and Subtyping 25

6 Conclusions 26

A Proofs 30

A.1 Facts about Structural Congruence 30

A.2 Proof of Proposition 1 32

A.3 Reformulating the Semantics of Section 4.2 38

A.4 Proof of Theorem 1 40

A.5 Proof of Theorem 2 53

A.6 Proof of Theorem 3 55

1 Motivation

A great deal of software is coded in terms of concurrent processes and objects. The purpose of our work is to develop a new formalism for expressing, typing, and reasoning about computations based on concurrent processes and objects.

We present a new formalism for expressing computations in terms of concurrent processes and objects. Our concurrent object calculus $\mathbf{conc\varsigma_m}$ consists of Abadi and Cardelli's imperative object calculus $\mathbf{imp\varsigma}$ extended with primitives for parallel composition and for synchronisation via mutexes. Our work extends the analysis by Abadi and Cardelli (1996) of object-oriented features to concurrent languages. At the heart of their work is a series of type systems able to express a great variety of object-oriented idioms. Given $\mathbf{conc\varsigma_m}$, we may smoothly and soundly extend these type systems to accommodate concurrency.

There are by now many formalisms capable of encoding objects and concurrency. Support of Abadi and Cardelli's type systems is one distinctive feature of our calculus. Others are the following. Unlike most process calculi, the syntax of $\mathbf{conc\varsigma_m}$ includes sequential composition of expressions that are expected to return results; there is no need to encode results in terms of continuations. Rather than reducing concurrent objects to other concepts, $\mathbf{conc\varsigma_m}$ treats objects as primitive. Rather than introduce auxiliary notions of stores or configurations or labelled transitions, we directly describe the semantics of $\mathbf{conc\varsigma_m}$ in terms of a reduction relation on expressions.

As evidence of the expressiveness of our calculus, we present a series of examples, including encodings of the π-calculus. Here are our main technical results:

(1) We describe a semantics for concurrent objects based on a reduction relation and a structural congruence relation in the style of Milner's reduction semantics (Milner 1992) for the π-calculus (Milner, Parrow, and Walker 1992). We prove that our reduction semantics is equivalent to a classical structural operational semantics defined using auxiliary notions of stores, threads, and configurations.

(2) We identify a single-threaded subset of our calculus that is preserved by reduction and includes the $\mathbf{imp\varsigma}$-calculus.

(3) The $\mathbf{Ob_{1<:}}$ calculus is Abadi and Cardelli's first-order calculus with objects and subtyping. Given a few simple rules for parallel composition and restriction, we confer the typing rules of $\mathbf{Ob_{1<:}}$ on our concurrent

calculus. We prove subject reduction for this system without needing any notion of store typing separate from the notion of expression typing.

1.1 Related work

We survey operational techniques for concurrent languages. We review work on formalisms based on functions as well as formalisms based on objects, since techniques suitable for functions are often applicable to objects.

Plotkin's structural operational semantics (1981) is a standard technique for concurrent languages. A computation is described as sequence of configurations. A configuration typically consists of a collection of runnable threads, a store, and other data such as the state of communication channels. Di Blasio and Fisher (1996) describe a concurrent version of the Fisher, Honsell, and Mitchell lambda-calculus of objects in this style. Other languages treated in this style include an actor language (Agha, Mason, Smith, and Talcott 1997) and CML (Reppy 1992) (Berry, Milner, and Turner 1992).

Ferreira, Hennessy, and Jeffrey (1995) avoid configurations in their operational semantics for CML by employing a CCS-style labelled transition system. In their work, and in ours, the parallel composition $a \upharpoonright b$ of two expressions a and b is an expression consisting of a and b running in parallel. Any result returned by b is returned by the whole composition; any result returned by a is discarded. So unlike the situation in most process calculi, parallel composition is not commutative: the effects of $a \upharpoonright b$ and $b \upharpoonright a$ are different. In implementation terms this is perfectly natural; running $a \upharpoonright b$ amounts to forking off a as a new thread and then running b. Another way of dealing with forked processes was investigated by Havelund and Larsen (1993): they present a form of CCS based on a binary operator for sequential composition and a unary operator that represents the forking of a parallel process.

Our reduction semantics is directly inspired by Milner's (1992) presentation of the chemical abstract machine of Berry and Boudol (1992). In a chemical semantics, a computation state is represented by a term of the calculus; there is no need for the auxiliary notion of a configuration. Previous chemical semantics for concurrent languages use evaluation contexts to treat sequential composition of expressions (Amadio, Leth, and Thomsen 1995) (Peyton Jones, Gordon, and Finne 1996) (Boudol 1997); instead, our semantics exploits a non-commutative parallel composition.

Di Blasio and Fisher's paper is the work most closely related to ours. Their principal results are the definition of a configuration-based reduction semantics for their calculus, a type soundness theorem, and the proof that certain guard expressions used for synchronisation have no side-effects. As in their work, we prove the soundness of a type system for concurrent objects. Our chemical semantics has no need for the auxiliary notions of configurations and reduction contexts used in theirs. Unlike their work, ours includes two independent but equivalent characterisations of our operational semantics.

Various formalisms in the π-calculus family have been used to model imperative or concurrent objects, for instance, in the work of Honda and Tokoro (1991), Jones (1993), Vasconcelos (1994), Pierce and Turner (1995), Walker (1995), Fournet and Gonthier (1996), Kleist and Sangiorgi (1998), and Dal Zilio (1998). All these models use formalisms based on processes, computations with no concept of returning a result, instead of expressions. The operation of returning a result is translated using continuations into sending a message on a result channel. Our **conc**$\boldsymbol{\varsigma}$-calculus is based on expressions that return results because its precursor **imp**$\boldsymbol{\varsigma}$ is based on expressions, because we do not wish to presuppose channel-based communication, and because expressions with results are a fundamental aspect of many programming languages and therefore deserve a semantics in their own right.

1.2 Organisation of the paper

In Section 2 we present the syntax and semantics of a core calculus of concurrent objects, the **conc**$\boldsymbol{\varsigma}$-calculus, and in Section 3 we add mutexes to obtain the **conc**$\boldsymbol{\varsigma_{\mathrm{m}}}$-calculus. Our syntax of terms unifies auxiliary notions of process, expression, store, and configuration, and hence supports a particularly simple reduction semantics. In Section 4 we prove that our semantics corresponds precisely to a more conventional, but more complex, semantics phrased in terms of configurations. In Section 5 we demonstrate the soundness of the $\mathbf{Ob}_{1<:}$ type system for **conc**$\boldsymbol{\varsigma_{\mathrm{m}}}$. Section 6 concludes the paper.

2 Concurrent Objects

We extend the imperative object calculus with primitives to assign a name to a stored object and to compose two terms in parallel. The resulting calculus allows us to express concurrent computations, but has no primitives to allow concurrent computations to synchronise. In Section 3, we extend this core calculus with mutex primitives for synchronisation.

2.1 Primitives for Concurrent Objects

We assume there are disjoint infinite sets of *names*, *variables*, and *labels*. We let p, q, and r range over names. We let x, y, and z range over variables. We let ℓ range over labels. We define the sets of *results*, *denotations*, and *terms* by the grammars:

Syntax of the concς-calculus

$u, v ::=$	results
$\quad x$	$\quad$ variable
$\quad p$	$\quad$ name
$d ::=$	denotations
$\quad [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$	$\quad$ object
$a, b, c ::=$	terms
$\quad u$	$\quad$ result
$\quad p \mapsto d$	$\quad$ denomination
$\quad u.\ell$	$\quad$ method select
$\quad u.\ell \Leftarrow \varsigma(x)b$	$\quad$ method update
$\quad \mathit{clone}(u)$	$\quad$ cloning
$\quad \mathit{let}\ x{=}a\ \mathit{in}\ b$	$\quad$ let
$\quad a \Rsh b$	$\quad$ parallel composition
$\quad (\nu p)a$	$\quad$ restriction

Syntactic conventions:

$$\begin{array}{rcl} (\nu p)a \Rsh b & \text{is read} & ((\nu p)a) \Rsh b \\ u.\ell \Leftarrow \varsigma(x)b \Rsh c & \text{is read} & (u.\ell \Leftarrow \varsigma(x)b) \Rsh c \\ \mathit{let}\ x{=}a\ \mathit{in}\ b \Rsh c & \text{is read} & (\mathit{let}\ x{=}a\ \mathit{in}\ b) \Rsh c \end{array}$$

Abbreviations:

$$(\nu \vec{p})a \quad \triangleq \quad (\nu p_1)(\nu p_2)\dots(\nu p_n)a \text{ where } \vec{p} = p_1, p_2, \dots, p_n$$

Given an object $[\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ we say that each $\varsigma(x_j)b_j$ for $j \in 1..n$ is a *method*, and that each $\ell_j = \varsigma(x_j)b_j$ for $j \in 1..n$ is a *component* of the object.

Here are the rules for scoping variables and names. In a method $\varsigma(x)b$, the variable x is bound; its scope is b. In a term $\mathit{let}\ x{=}a\ \mathit{in}\ b$, the variable x is bound; its scope is b. In a restriction, $(\nu p)a$, the name p is bound; its scope is a. Let $\mathit{fn}(a)$ be the set of names free in the term a. Let $\mathit{fv}(a)$ be the set of variables free in the term a. We say that a term a is *closed* if and only if $fv(a) = \varnothing$. We write $a\{\!\{x \leftarrow v\}\!\}$ for the outcome of a capture-avoiding substitution of the result v for each free occurrence of x in term a.

We write $a = b$ to mean that the terms a and b are equal up to the renaming of bound names and bound variables, and the reordering of components in objects.

As in the **impς**-calculus, our syntax distinguishes names, which represent the addresses of stored objects, from variables, which represent intermediate values. This is a helpful distinction but not essential; we believe it will be useful when we come to treat observational equivalences. Results in our syntax are atomic names or atomic variables; our techniques would easily extend to structured results, such as tuples or λ-abstractions. Our syntax separates name scoping, by restrictions, from name definition, by denominations. We separated scoping from definition to allow cyclic dependencies between definitions. An alternative is to use a single construct defining several names simultaneously with mutually recursive scopes, as in the join-calculus (Fournet and Gonthier 1996) for example. Due to the generality of our syntax, we need a simple type system, defined in Section 4.1, to rule out certain terms as not well-formed. For example, a process such as $(p \mapsto [] \Rsh p \mapsto []) \Rsh p$, that contains two denominations for the same name, is not well-formed.

Starting with the terms of the imperative object calculus, we arrive at our calculus by labelling each object with a name, and adding parallel composition $a \Rsh b$ and restriction $(\nu p)a$ from the π-calculus. As the next section explains, we obtain the semantics of our calculus by combining the semantics of the imperative object calculus with that of the π-calculus.

2.2 Informal Semantics

We may interpret a term of our object calculus either as a *process* or as an *expression.* A process is simply a concurrent computation. An expression is a concurrent computation that is expected to return a result. In fact, an expression may be regarded as a process, since we may always ignore any result that it returns.

The meanings of the first six primitives (result, denomination, method select, method update, cloning, and let) are much as in the **impς**-calculus:

- A result u is an expression that immediately returns itself.

- A denomination $p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ is a process that confers the name p on the object $[\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$. We say that the object $[\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ is the denotation of the name p. Intuitively, the process represents an object stored at a memory location and the name p represents the address of the object.

- A method select $p.\ell$ is an expression that invokes the method labelled ℓ of the object denoted by p. In the presence of a denomination $p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$, where $\ell = \ell_j$ for some $j \in 1..n$, the effect of $p.\ell$ is to run the expression $b_j\{\!\{x_j \leftarrow p\}\!\}$, that is, to run the body b_j of the method labelled ℓ, with the variable x_j bound to the name of the object itself.

- A method update $p.\ell \Leftarrow \varsigma(x)b$ is an expression that updates the method labelled ℓ of the object denoted by p. In the presence of a denomination $p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$, where $\ell = \ell_j$ for some $j \in 1..n$, the effect of $p.\ell \Leftarrow \varsigma(x)b$ is to update the denomination to be $p \mapsto [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i\ ^{i \in (1..n)-\{j\}}]$, and to return p as its result.

- A clone $\mathit{clone}(p)$ is an expression that makes a shallow copy of the object denoted by p. In the presence of a denomination $p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$, the effect of $\mathit{clone}(p)$ is to generate a fresh name q with denomination $q \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ and to return q as its result. After a clone, the names p and q denote two separate copies of the same denotation $[\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$; updates to one will not affect the other.

- A let $\mathit{let}\ x{=}a\ \mathit{in}\ b$ is an expression that first runs the expression a, and if it returns a result, calls it x, and then runs the expression b.

The meanings of the last two primitives (parallel composition and restriction) are much as in the π-calculus:

- A parallel composition $a \Rsh b$ is either an expression or a process, depending on whether b is an expression or a process. In $a \Rsh b$ the terms a and b are running in parallel. If b is an expression then $a \Rsh b$ is an expression, whose result, if any, is the result returned by b. Any result returned by a is ignored.

- A restriction $(\nu p)a$ is either an expression or a process, depending on whether a is an expression or a process. A restriction $(\nu p)a$ generates a fresh name p whose scope is a.

In this section, our intuitive explanations have depended on an informal distinction between processes and expressions. We make this distinction precise via judgments $a : \mathit{Proc}$ and $a : \mathit{Exp}$ in Section 4.1.

2.3 Formal Semantics

We base our operational semantics on structural congruence and reduction relations. Reduction represents individual computation steps, and is defined in terms of structural congruence. Structural congruence allows the rearrangement of the syntactic structure of a term so that reduction rules may be applied. We may regard our semantics as a concurrent extension of the small-step substitution-based semantics of **impς** described by Gordon, Hankin, and Lassen (1997).

The most interesting aspect of our formal semantics is the management of concurrent expressions that return results. We intend that the result of an expression be that returned from the right-hand side of the topmost parallel composition. Therefore, as we discussed in Section 1.1, in contexts expecting a result, parallel composition is not commutative. On the other hand, in contexts immediately to the left of a parallel composition, where any result is discarded, parallel composition is commutative. Therefore, structural congruence identifies $(a \Rsh b) \Rsh c$ with $(b \Rsh a) \Rsh c$, since any results returned by a or b are discarded.

The following two tables define the structural congruence relation $a \equiv b$.

Structural congruence: congruence rules

(Struct Refl) (Struct Symm) (Struct Trans)

$$\frac{}{a \equiv a} \qquad \frac{b \equiv a}{a \equiv b} \qquad \frac{a \equiv b \qquad b \equiv c}{a \equiv c}$$

(Struct Update) (Struct Let)

$$\frac{b \equiv b'}{u.\ell \Leftarrow \varsigma(x)b \equiv u.\ell \Leftarrow \varsigma(x)b'} \qquad \frac{a \equiv a' \qquad b \equiv b'}{let\ x{=}a\ in\ b \equiv let\ x{=}a'\ in\ b'}$$

(Struct Res) (Struct Par)

$$\frac{a \equiv a'}{(\nu p)a \equiv (\nu p)a'} \qquad \frac{a \equiv a' \qquad b \equiv b'}{a \Rsh b \equiv a' \Rsh b'}$$

(Struct Object)

$$\frac{b_i \equiv b_i' \quad \forall i \in 1..n}{p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}] \equiv p \mapsto [\ell_i = \varsigma(x_i)b_i'\ ^{i \in 1..n}]}$$

Structural congruence: basic axioms

(Struct Par Assoc) (Struct Par Comm) (Struct Res Res)

$$\frac{}{(a \Rsh b) \Rsh c \equiv a \Rsh (b \Rsh c)} \qquad \frac{}{(a \Rsh b) \Rsh c \equiv (b \Rsh a) \Rsh c} \qquad \frac{}{(\nu p)(\nu q)a \equiv (\nu q)(\nu p)a}$$

(Struct Par 1)

$$\frac{p \notin fn(a)}{(\nu p)(a \upharpoonright b) \equiv a \upharpoonright (\nu p)b}$$

(Struct Par 2)

$$\frac{p \notin fn(b)}{(\nu p)(a \upharpoonright b) \equiv ((\nu p)a) \upharpoonright b}$$

(Struct Let Assoc)

$$\frac{y \notin fv(c)}{let\ x{=}(let\ y{=}a\ in\ b)\ in\ c \equiv let\ y{=}a\ in\ (let\ x{=}b\ in\ c)}$$

(Struct Res Let)

$$\frac{p \notin fn(b)}{(\nu p) let\ x{=}a\ in\ b \equiv let\ x{=}(\nu p)a\ in\ b}$$

(Struct Par Let)

$$\frac{}{a \upharpoonright let\ x{=}b\ in\ c \equiv let\ x{=}(a \upharpoonright b)\ in\ c}$$

We explained (Struct Par Comm) earlier. The rules (Struct Par Assoc), (Struct Res Res), (Struct Par 1), and (Struct Par 2) are counterparts of similar rules for the π-calculus. (Struct Let Assoc) is a standard rule for let familiar from the computational λ-calculus (Moggi 1989). (Struct Res Let) and (Struct Fork Let) allow the term a in $let\ x{=}a\ in\ b$ to interact with parallel processes.

The following table defines the reduction relation $a \to b$:

Reduction

(Red Select)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i{}^{\,i \in 1..n}] \quad j \in 1..n}{(p \mapsto d) \upharpoonright p.\ell_j \to (p \mapsto d) \upharpoonright b_j\{\!\{x_j \leftarrow p\}\!\}}$$

(Red Update)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i{}^{\,i \in 1..n}] \quad j \in 1..n \quad d' = [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i{}^{\,i \in (1..n)-\{j\}}]}{(p \mapsto d) \upharpoonright (p.\ell_j \Leftarrow \varsigma(x)b) \to (p \mapsto d') \upharpoonright p}$$

(Red Clone)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i{}^{\,i \in 1..n}] \quad q \notin fn(d)}{(p \mapsto d) \upharpoonright clone(p) \to (p \mapsto d) \upharpoonright (\nu q)(q \mapsto d \upharpoonright q)}$$

(Red Let Result)

$$\frac{}{let\ x{=}p\ in\ b \to b\{\!\{x \leftarrow p\}\!\}}$$

(Red Res)

$$\frac{a \to a'}{(\nu p)a \to (\nu p)a'}$$

(Red Par 1)

$$\frac{a \to a'}{a \upharpoonright b \to a' \upharpoonright b}$$

(Red Par 2)

$$\frac{b \to b'}{a \upharpoonright b \to a \upharpoonright b'}$$

(Red Let) (Red Struct)

$$\frac{a \to a'}{\mathit{let}\ x{=}a\ \mathit{in}\ b \to \mathit{let}\ x{=}a'\ \mathit{in}\ b} \qquad \frac{a \equiv a' \quad a' \to b' \quad b' \equiv b}{a \to b}$$

Rules (Red Select), (Red Update), (Red Clone), and (Red Let Result) correspond to the basic computation steps of the **imp**$\boldsymbol{\varsigma}$-calculus. (Red Res), (Red Par 1), (Red Par 2), and (Red Let) are congruence rules. (Red Struct) is a standard rule allowing a term to be re-arranged up to structural congruence during reduction.

2.4 Examples of Concurrent Objects

We illustrate the operational semantics of **conc**$\boldsymbol{\varsigma}$ via examples drawn from encodings of the **imp**$\boldsymbol{\varsigma}$-calculus and the call-by-value λ-calculus.

2.4.1 Imperative objects

We may embed all the expressions of the **imp**$\boldsymbol{\varsigma}$-calculus in **conc**$\boldsymbol{\varsigma}$ via the following abbreviations. We show in Section 4.3 that the reductions of any term of **imp**$\boldsymbol{\varsigma}$ embedded in **conc**$\boldsymbol{\varsigma}$ are deterministic.

The imp$\boldsymbol{\varsigma}$**-calculus**

d (as a term)	$\triangleq$	$(\nu p)(p \mapsto d \upharpoonright p)$	for $p \notin \mathit{fn}(d)$
$a.\ell$	$\triangleq$	$\mathit{let}\ x{=}a\ \mathit{in}\ x.\ell$	for a not a result
$a.\ell \Leftarrow \varsigma(x)b$	$\triangleq$	$\mathit{let}\ y{=}a\ \mathit{in}\ y.\ell \Leftarrow \varsigma(x)b$	for a not a result and $y \notin \mathit{fv}(b)$
$\mathit{clone}(a)$	$\triangleq$	$\mathit{let}\ x{=}a\ \mathit{in}\ \mathit{clone}(x)$	for a not a result

Here is an example, from Abadi and Cardelli's book, of a computation involving method update and method select:

$$\begin{aligned}
&[\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x].\ell \\
&\quad = \quad \mathit{let}\ z{=}[\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x]\ \mathit{in}\ z.\ell \\
&\quad = \quad \mathit{let}\ z{=}(\nu p)(p \mapsto [\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x] \upharpoonright p)\ \mathit{in}\ z.\ell \\
&\quad \equiv \quad (\nu p)(p \mapsto [\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x] \upharpoonright \mathit{let}\ z{=}p\ \mathit{in}\ z.\ell) \\
&\quad \to \quad (\nu p)(p \mapsto [\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x] \upharpoonright p.\ell) \\
&\quad \to \quad (\nu p)(p \mapsto [\ell = \varsigma(x)x.\ell \Leftarrow \varsigma(y)x] \upharpoonright p.\ell \Leftarrow \varsigma(y)p) \\
&\quad \to \quad (\nu p)(p \mapsto [\ell = \varsigma(y)p] \upharpoonright p)
\end{aligned}$$

Here is an example that illustrates the interaction between let and composition:

$$
\begin{array}{rcl}
p \mapsto [\ell = \varsigma(x)b] \upharpoonright \mathit{let}\ x{=}p.\ell\ \mathit{in}\ c & \equiv & \mathit{let}\ x{=}(p \mapsto [\ell = \varsigma(x)b] \upharpoonright p.\ell)\ \mathit{in}\ c \\
& \rightarrow & \mathit{let}\ x{=}(p \mapsto [\ell = \varsigma(x)b] \upharpoonright b\{\!\{x \leftarrow p\}\!\})\ \mathit{in}\ c \\
& \equiv & p \mapsto [\ell = \varsigma(x)b] \upharpoonright \mathit{let}\ x{=}b\{\!\{x \leftarrow p\}\!\}\ \mathit{in}\ c
\end{array}
$$

We may generate a cyclic dependency between denominations:

$$
\begin{array}{l}
\mathit{let}\ x_1{=}[\ell = \varsigma(y_1)y_1]\ \mathit{in}\ \mathit{let}\ x_2{=}[\ell = \varsigma(y_2)x_1]\ \mathit{in}\ x_1.\ell \Leftarrow \varsigma(y_1)x_2 \\
\quad \rightarrow^* \quad (\nu p_1)(\nu p_2)(p_1 \mapsto [\ell = \varsigma(y_1)p_2] \upharpoonright p_2 \mapsto [\ell = \varsigma(y_2)p_1] \upharpoonright p_1)
\end{array}
$$

2.4.2 Concurrent procedure calls

We encode λ-abstraction and application as in Gordon, Hankin, and Lassen (1997):

The call-by-value λ-calculus

$$
\begin{array}{rcl}
\lambda(x)b & \triangleq & [\mathit{arg} = \varsigma(s)s.\mathit{arg}, \mathit{val} = \varsigma(s)\mathit{let}\ x{=}s.\mathit{arg}\ \mathit{in}\ b] \quad \text{for } s \notin \mathit{fv}(b) \\
b(a) & \triangleq & (b.\mathit{arg} \Leftarrow \varsigma(x)a).\mathit{val}
\end{array}
$$

To illustrate the action of procedure calls, let fact_p be an object representing some procedure, for example, factorial.

$$
\mathit{fact}_p \triangleq [\mathit{arg} = \varsigma(x)p, \mathit{val} = \varsigma(s)\mathit{let}\ y{=}s.\mathit{arg}\ \mathit{in}\ \mathit{body}\{y\}]
$$

We assume for the purpose of this example some implementation of numerals as objects. We write the number n for the name of an object representing n.

The following illustrates a procedure call:

$$
\begin{array}{ll}
\multicolumn{2}{l}{\mathit{let}\ f{=}\mathit{fact}_0\ \mathit{in}\ f(10)} \\
= & \mathit{let}\ f{=}(\nu p)(p \mapsto \mathit{fact}_0 \upharpoonright p)\ \mathit{in}\ f(10) \\
\equiv & (\nu p)(p \mapsto \mathit{fact}_0 \upharpoonright \mathit{let}\ f{=}p\ \mathit{in}\ f(10)) \\
\rightarrow & (\nu p)(p \mapsto \mathit{fact}_0 \upharpoonright p(10)) \\
= & (\nu p)(p \mapsto \mathit{fact}_0 \upharpoonright (p.\mathit{arg} \Leftarrow \varsigma(x)10).\mathit{val}) \\
\rightarrow & (\nu p)(p \mapsto \mathit{fact}_{10} \upharpoonright p.\mathit{val}) \\
\rightarrow^2 & (\nu p)(p \mapsto \mathit{fact}_{10} \upharpoonright \mathit{body}\{\!\{10\}\!\}) \\
\equiv & (\nu p)(p \mapsto \mathit{fact}_{10}) \upharpoonright \mathit{body}\{\!\{10\}\!\}
\end{array}
$$

A first try at writing two concurrent procedure calls is:

$$
\mathit{let}\ f{=}\mathit{fact}_0\ \mathit{in}\ (f(10) \upharpoonright f(20))
$$

Since there is no synchronisation between the applications $f(10)$ and $f(20)$, one may interfere with the other. To avoid this, we use clone:

$$\mathit{let}\ f{=}\mathit{fact}_0\ \mathit{in}\ (\mathit{clone}(f)(10) \Rsh \mathit{clone}(f)(20))$$

The code above returns the result of $\mathit{clone}(f)(20)$ but discards the result of $\mathit{clone}(f)(10)$. The following example shows how a let may be used to process the results from both the calls.

$$\begin{array}{l} \mathit{let}\ f{=}\mathit{fact}_0\ \mathit{in} \\ \quad ((\mathit{let}\ x_1{=}\mathit{clone}(f)(10)\ \mathit{in}\ Q_1) \Rsh (\mathit{let}\ x_2{=}\mathit{clone}(f)(20)\ \mathit{in}\ Q_2)) \end{array}$$

Although the result of Q_1 will ultimately be discarded, Q_1 may still influence the result of the whole computation by communicating with Q_2, for example.

3 Synchronisation

Different object-oriented languages use a variety of techniques to synchronise concurrent processes. We need some way to model process synchronisation within our calculus.

One approach would be to encode synchronisation in terms of critical sections. The mutual exclusion problem is the problem of enforcing mutually exclusive access to a critical section in the presence of several concurrent processes. Starting with Dijkstra (1965), many algorithms have been proposed to solve this problem in terms of primitives for atomic reads and writes on a shared memory. Since we can encode these primitives within the $\mathbf{conc}\varsigma$-calculus, we can also encode any of the solutions to the mutual exclusion problem. Therefore, we could use critical sections to encode higher level synchronisation mechanisms like object locking or communication channels within the $\mathbf{conc}\varsigma$-calculus.

We prefer not to use such an encoding for two reasons. First, the encoding is anachronistic since mutual exclusion is normally solved using hardware primitives (such as inhibition of interrupts) rather than reads and writes to a shared memory. Second, the encoding would lead to complicated calculations about the reduction behaviour of higher-level synchronisation mechanisms.

Instead, we prefer to encode synchronisation mechanisms in a calculus $\mathbf{conc}\varsigma_{\mathbf{m}}$ obtained by extending the $\mathbf{conc}\varsigma$-calculus with mutexes (binary semaphores). Unlike shared variable mutual exclusion algorithms, mutexes are commonly used in the runtime systems of object-oriented languages and have simple reduction rules. We have defined a compositional translation of

$\mathbf{conc}\varsigma_{\mathbf{m}}$ into $\mathbf{conc}\varsigma$ using a two process mutual exclusion algorithm (Lamport 1985) to guarantee exclusive access to the objects representing mutexes. We conjecture that this translation is sound with respect to a suitable notion of observational equivalence, but not fully abstract.

A third approach would be to add synchronisation mechanisms to the primitive operations on objects, as in the calculus of Di Blasio and Fisher (1996). To keep the primitives of our calculus simple, we prefer not to integrate a specific synchronisation construct into the semantics of method select and method update.

3.1 Primitives for Synchronisation

We extend the $\mathbf{conc}\varsigma$-calculus with mutexes as follows:

Syntax of the $\mathbf{conc}\varsigma_{\mathbf{m}}$-calculus

$d ::=$	denotation
$\quad\ldots$	as in Section 2.1
$\quad$*locked*	locked mutex
$\quad$*unlocked*	unlocked mutex
$a, b, c ::=$	term
$\quad\ldots$	as in Section 2.1
$\quad$*acquire*(u)	mutex acquisition
$\quad$*release*(u)	mutex release

As in Section 2.4.1, we adopt a convention allowing denotations to be used as terms. As a term, let *locked* be short for $(\nu p)(p \mapsto \mathit{locked} \upharpoonright p)$. Similarly, let *unlocked* be short for $(\nu p)(p \mapsto \mathit{unlocked} \upharpoonright p)$. Moreover, if a is not a result, let *acquire*(a) and *release*(a) be short for *let* $x{=}a$ *in acquire*(x) and *let* $x{=}a$ *in release*(x), respectively.

3.2 Informal Semantics

We may explain the semantics of mutexes as follows:

- A denomination $p \mapsto \mathit{locked}$ or $p \mapsto \mathit{unlocked}$ represents a mutex, denoted by p, whose state is locked or unlocked, respectively. Intuitively, the mutex is a bit stored at memory location p.

- A mutex acquisition *acquire*(p) attempts to lock the mutex denoted by p. If a denomination $p \mapsto \mathit{unlocked}$ is present, the acquisition *acquire*(p) changes its state to $p \mapsto \mathit{locked}$, and returns p as its result. Otherwise the acquisition blocks.

- A mutex release $\mathit{release}(p)$ unconditionally unlocks the mutex denoted by p. If a denomination $p \mapsto d$ is present, for $d \in \{\mathit{locked}, \mathit{unlocked}\}$, the release $\mathit{release}(p)$ sets its state to $p \mapsto \mathit{unlocked}$, and returns p as its result.

3.3 Formal Semantics

We define the structural congruence relation $\equiv$ by exactly the same rules as in Section 2.3. The reduction relation $\to$ is defined by the rules in Section 2.3 together with two new rules for mutex acquisition and release:

Reduction

$$\begin{array}{ll}
(p \mapsto \mathit{unlocked}) \Rsh \mathit{acquire}(p) \to (p \mapsto \mathit{locked}) \Rsh p & \text{(Red Acquire)} \\
(p \mapsto d) \Rsh \mathit{release}(p) \to (p \mapsto \mathit{unlocked}) \Rsh p & \text{(Red Release)} \\
\quad \text{for } d \in \{\mathit{locked}, \mathit{unlocked}\} &
\end{array}$$

3.4 Examples of Synchronisation

3.4.1 Mutual exclusion

We may protect access to shared state with a mutex to prevent interference between concurrent threads. The operation $\mathit{lock}\ u\ \mathit{in}\ a$ blocks until it can acquire the mutex u, runs a, then releases u.

$$\begin{array}{rcl}
a; b & \triangleq & \mathit{let}\ x{=}a\ \mathit{in}\ b \\
\mathit{lock}\ u\ \mathit{in}\ a & \triangleq & \mathit{acquire}(u); \mathit{let}\ y{=}a\ \mathit{in}\ (\mathit{release}(u); y)
\end{array}$$

In Section 2.4.2 we used cloning to prevent interference between two concurrent calls to a shared procedure. We may protect access to the shared procedure with a mutex as follows:

$$\mathit{let}\ x{=}\mathit{unlocked}\ \mathit{in}\ \mathit{let}\ f{=}\mathit{fact}_0\ \mathit{in}\ (\mathit{lock}\ x\ \mathit{in}\ f(10)) \Rsh (\mathit{lock}\ x\ \mathit{in}\ f(20))$$

With this idiom for calling a shared procedure the calls $f(10)$ and $f(20)$ are serialised; the first to run must terminate before the second may run. Hence this idiom allows for less concurrency than the one in Section 2.4.2 using clone. Serialisation is necessary if the shared procedure accesses some persistent state.

3.4.2 Asynchronous channels

Consider an asynchronous communications channel as in Pict (Pierce and Turner 1997) or Concurrent Haskell (Peyton Jones, Gordon, and Finne 1996). Such a channel is an object named by p, that either contains a result or is empty, and has two methods *read* and *write*. If the object p is empty, the operation $p.\mathit{write}(v)$ updates p so that it contains v, while the operation $p.\mathit{read}$ blocks. If the object p contains the result v, the operation $p.\mathit{read}$ returns v and updates p so that it is empty, while the operation $p.\mathit{write}(u)$ blocks. We code this behaviour as follows, where *nil* is a name used to initialise the channel. (Di Blasio and Fisher (1996) implement a similar abstraction in their calculus of concurrent objects.)

Asynchronous channels

```
$\mathit{chan}_a\ u_{rd}\ u_{wr}\ v \triangleq$
  $[\mathit{reader} = u_{rd}, \mathit{writer} = u_{wr}, \mathit{val} = v,$
   $\mathit{read} = \varsigma(s)$
      $\mathit{acquire}(s.\mathit{reader}); \mathit{let}\ x{=}s.\mathit{val}\ \mathit{in}\ (\mathit{release}(s.\mathit{writer}) \Rsh x),$
   $\mathit{write} = \varsigma(s)\lambda(x)($
      $\mathit{acquire}(s.\mathit{writer}); s.\mathit{val} \Leftarrow \varsigma(s)x; \mathit{release}(s.\mathit{reader})) \Rsh x)]$
$\mathit{newChan}_a \triangleq$
  $\mathit{let}\ \mathit{reader}{=}\mathit{locked}\ \mathit{in}\ \mathit{let}\ \mathit{writer}{=}\mathit{unlocked}\ \mathit{in}$
     $\mathit{chan}_a\ \mathit{reader}\ \mathit{writer}\ \mathit{nil}$
```

This code maintains the invariant that at any time at most one of the locks *reader* and *writer* is unlocked. If *reader* is unlocked, the result in *val* is the contents of the channel. If *writer* is unlocked, the channel is empty.

The body of the *write* method is a λ-abstraction. Each call to this method allocates a fresh object that represents the λ-abstraction. Therefore concurrent calls to *write* do not interfere with each other.

We make the following definitions to represent states of a channel:

$$\begin{aligned}
p \mapsto \mathit{channel}_a\ d_{rd}\ d_{wr}\ v &\triangleq (\nu q_{rd})(\nu q_{wr})(q_{rd} \mapsto d_{rd} \Rsh q_{wr} \mapsto d_{wr} \Rsh \\
&\qquad p \mapsto \mathit{chan}_a\ q_{rd}\ q_{wr}\ v) \\
p \mapsto \mathit{empty}\ u &\triangleq p \mapsto \mathit{channel}_a\ \mathit{locked}\ \mathit{unlocked}\ u \\
p \mapsto \mathit{full}\ u &\triangleq p \mapsto \mathit{channel}_a\ \mathit{unlocked}\ \mathit{locked}\ u
\end{aligned}$$

We can show that our implementation has the following properties:

$$\begin{aligned}
\mathit{newChan}_a &\to^* (\nu p)(p \mapsto \mathit{empty}\ \mathit{nil} \Rsh p) \\
p \mapsto \mathit{empty}\ q \Rsh p.\mathit{write}(q') &\to^* p \mapsto \mathit{full}\ q' \Rsh q' \\
p \mapsto \mathit{full}\ q \Rsh p.\mathit{read} &\to^* p \mapsto \mathit{empty}\ q \Rsh q
\end{aligned}$$

Given asynchronous channels, we can encode the asynchronous π-calculus:

Encoding the asynchronous π-calculus

$$
\begin{array}{rcl}
[\![x]\!] & \triangleq & x \\
[\![\overline{x}y]\!] & \triangleq & x.\mathit{write}(y) \\
[\![x(y).P]\!] & \triangleq & \mathit{let}\ y{=}x.\mathit{read}\ \mathit{in}\ [\![P]\!] \\
[\![!x(y).P]\!] & \triangleq & [\mathit{rep} = \varsigma(s)\mathit{let}\ y{=}x.\mathit{read}\ \mathit{in}\ ([\![P]\!] \Rsh s.\mathit{rep})].\mathit{rep} \\
 & & \quad \text{for } s \notin \{x, y\} \cup \mathit{fv}(P) \\
[\![P \mid Q]\!] & \triangleq & [\![P]\!] \Rsh [\![Q]\!] \\
[\![(\mathit{new}\ x)P]\!] & \triangleq & \mathit{let}\ x{=}\mathit{newChan}_a\ \mathit{in}\ [\![P]\!]
\end{array}
$$

We conjecture that this translation is sound with respect to a suitable notion of observational equivalence. This particular translation is not fully abstract, since the encoding of channels allows an observer to discover the last message sent on a channel.

3.4.3 Synchronous channels

The implementation of channels in the previous section is asynchronous in the sense that a writer $p.\mathit{write}(v)$ returns as soon as it has deposited v in the channel p, and does not wait until a reader $p.\mathit{read}$ has obtained v. In the following code, a reader $p.\mathit{read}$ signals to a writer $p.\mathit{write}(v)$ that it has obtained v by releasing the lock $p.\mathit{ack}$. To prevent races between multiple writers, we serialise calls to the $p.\mathit{write}$ method using a lock $p.\mathit{writeLock}$.

Synchronous channels

$$
\begin{array}{l}
\mathit{chan}_s\ u_{\mathit{ch}}\ u_{\mathit{ack}}\ u_{\mathit{wr}}\ v \triangleq \\
\quad [\mathit{ch} = u_{\mathit{ch}}, \mathit{ack} = u_{\mathit{ack}}, \mathit{writeLock} = u_{\mathit{wr}}, \\
\quad\ \mathit{read} = \varsigma(s)\mathit{let}\ x{=}s.\mathit{ch}.\mathit{read}\ \mathit{in}\ (\mathit{release}(s.\mathit{ack}) \Rsh x) \\
\quad\ \mathit{write} = \varsigma(s)\lambda(x)\mathit{lock}\ s.\mathit{writeLock}\ \mathit{in}\ (s.\mathit{ch}.\mathit{write}(x); \mathit{acquire}(s.\mathit{ack}); x)] \\
\mathit{newChan}_s \triangleq \\
\quad \mathit{let}\ \mathit{ch}{=}\mathit{newChan}_a\ \mathit{in}\ \mathit{let}\ \mathit{ack}{=}\mathit{locked}\ \mathit{in}\ \mathit{let}\ \mathit{lock}{=}\mathit{unlocked}\ \mathit{in} \\
\qquad \mathit{chan}_s\ \mathit{ch}\ \mathit{ack}\ \mathit{lock}\ \mathit{nil}
\end{array}
$$

Given synchronous channels, we can encode the choice-free synchronous π-calculus by revising and extending the previous translation:

Encoding the choice-free synchronous π-calculus

$$\begin{aligned}[\![\overline{x}y.P]\!] &\triangleq x.write(y);[\![P]\!]\\ [\![(new\ x)P]\!] &\triangleq let\ x{=}newChan_s\ in\ [\![P]\!]\end{aligned}$$

We leave an encoding of guarded choice as future work.

3.4.4 Fork and join

A common pattern of concurrency is to fork off a thread to compute a result, and later to await this result using a join operation. We may easily code these operations in terms of mutexes, but there is a particularly simple implementation using asynchronous channels:

$$\begin{aligned}fork\ b &\triangleq let\ ch{=}newChan_a\ in\ (let\ x{=}b\ in\ ch.write(x) \Rsh ch)\\ join\ u &\triangleq u.read\end{aligned}$$

To illustrate fork and join, suppose we have some binary operation $u \oplus v$ on results. We can extend this to an operation $a \oplus b$ on arbitrary terms that evaluates a and b in parallel:

$$a \oplus b \triangleq let\ th{=}fork\ b\ in\ let\ x{=}a\ in\ let\ y{=}join\ th\ in\ x \oplus y$$

4 A Structural Characterisation of Reduction

The purpose of this section is to characterise our reduction semantics in terms of a more conventional structural operational semantics. This is desirable for two reasons. First, it increases our confidence in the correctness of our semantics. Second, it provides a convenient way to enumerate all possible reductions of a term.

Section 4.1 describes the well-formed terms of **conc**$\boldsymbol{\varsigma}_{\mathbf{m}}$ using a rudimentary type system that distinguishes expressions (terms expected to return a result) from processes. In Section 4.2, we demonstrate that on well-formed terms our reduction semantics coincides with a structural operational semantics defined using configurations. Finally, in Section 4.3, we identify a single-threaded fragment of **conc**$\boldsymbol{\varsigma}$ by omitting a single rule from the rudimentary type system. We show this fragment is deterministic and includes the **imp**$\boldsymbol{\varsigma}$-calculus.

4.1 Well-formed Terms

We present a type system for well-formed terms that distinguishes expressions from processes. In this type system, there are only two types *Proc* and *Exp*, representing processes and expressions, respectively. Since we may always ignore the result of an expression, any term of type *Exp* is also a term of type *Proc*. The type system is very liberal and provides only two guarantees about well-formed terms. First, it guarantees that a proper process does not occur in a context expecting an expression. Second, it guarantees that the top-level denominations of free names in a term represent a partial function from names to objects whose domain is preserved by computation steps. Later, in Section 5, we study a stronger type system that prevents "message not understood" errors.

The top-level denominations in a term play the role of locations in a store. It is convenient to define the *domain* of a term a, $dom(a)$, to be the set of free names named by top-level denominations in a:

Domain of a term

$$\begin{array}{lcll}
dom(p \mapsto d) & \triangleq & \{p\} & \\
dom(let\ x{=}a\ in\ b) & \triangleq & dom(a) & \\
dom(a \upharpoonright b) & \triangleq & dom(a) \cup dom(b) & \\
dom((\nu p)a) & \triangleq & dom(a) - \{p\} & \\
dom(a) & \triangleq & \varnothing & \text{for any other kind of } a
\end{array}$$

Let T stand for either *Proc* or *Exp*. The well-formed terms are given by the judgment $a : T$ defined in the following table. We say that term a is a *process* if and only if $a : Proc$. Similarly, we say that a term a is an *expression* if and only if $a : Exp$.

Well-formed terms

(Well Result)
$$\frac{}{u : Exp}$$

(Well Object)
$$\frac{b_i : Exp \quad dom(b_i) = \varnothing \quad \forall i \in 1..n}{p \mapsto [\ell_i = \varsigma(x_i)b_i\ {}^{i \in 1..n}] : Proc}$$

(Well Mutex)
$$\frac{d \in \{locked, unlocked\}}{p \mapsto d : Proc}$$

(Well Select)
$$\frac{}{u.\ell : Exp}$$

(Well Update)
$$\frac{b : Exp \quad dom(b) = \varnothing}{u.\ell \Leftarrow \varsigma(x)b : Exp}$$

(Well Clone)
$$\frac{}{clone(u) : Exp}$$

(Well Acquire)
$$\frac{}{acquire(u) : Exp}$$

(Well Release)
$$\frac{}{release(u) : Exp}$$

(Well Let)
$$\frac{a : Exp \quad b : Exp \quad dom(b) = \varnothing}{let\ x{=}a\ in\ b : Exp}$$

(Well Res)
$$\frac{a : T \quad p \in dom(a)}{(\nu p)a : T}$$

(Well Par)

$$\frac{a : \mathit{Proc} \quad b : T \quad \mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing}{a \upharpoonright b : T}$$

(Well Concur)

$$\frac{a : \mathit{Exp}}{a : \mathit{Proc}}$$

Results, method selects and updates, clones, mutex acquisitions and releases, and lets are all expected to return a result, so the rules (Well Result), (Well Select), (Well Update), (Well Clone), (Well Acquire), (Well Release), and (Well Let) assign them all the type *Exp*. The rules (Well Object) and (Well Mutex) assign the type *Proc* to denominations, since they do not return results. The conditions on method bodies in the rules (Well Object) and (Well Update) guarantee that method selects yield expressions, and that method selects do not affect the domain of the term. In (Well Let), the condition $\mathit{dom}(b) = \varnothing$ guarantees that if b ever runs it will not alter the domain of the term.

The rule (Well Res) allows a restriction $(\nu p)a$ to be of either type, depending on the type of its body a. The condition $p \in \mathit{dom}(a)$ guarantees that any selects, updates or clones of p within a cannot block because no object is named by p. In other words, if we think of the name p as a pointer, and a denomination $p \mapsto d$ as the memory location to which p points, the condition $p \in \mathit{dom}(a)$ guarantees that no occurrence of p within a is a dangling pointer.

The rule (Well Par) allows a composition $a \upharpoonright b$ to be of either type, depending on the type of the term b. The condition $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$ prevents there being a denomination of the same name in both a and b. Finally, the rule (Well Concur) allows an expression to be treated as a process.

For example, we may derive $(\nu p)(p \mapsto d \upharpoonright p.\ell) : \mathit{Exp}$ where $d = [\ell = \varsigma(x)x.\ell]$. By (Well Select), $x.\ell : \mathit{Exp}$. By (Well Object), this implies $p \mapsto d : \mathit{Proc}$. By (Well Select), $p.\ell : \mathit{Exp}$. By (Well Par), the latter two judgments imply $(p \mapsto d \upharpoonright p.\ell) : \mathit{Exp}$. By (Well Res), this implies $(\nu p)(p \mapsto d \upharpoonright p.\ell) : \mathit{Exp}$.

Terms that are not well-formed include $p \mapsto d_1 \upharpoonright p \mapsto d_2$, $\mathit{let}\ x{=}p \mapsto d\ \mathit{in}\ b$, $(\nu p)p$, and $p \mapsto [\ell = \varsigma(x)q \mapsto d]$. None of these receives a type.

Structural congruence and reduction respect typing:

Proposition 1

(1) *If $a : T$ and $a \equiv b$ then $b : T$ and $\mathit{dom}(a) = \mathit{dom}(b)$.*

(2) *If $a : T$ and $a \to b$ then $b : T$ and $\mathit{dom}(a) = \mathit{dom}(b)$.*

4.2 A Structural Operational Semantics

A conventional technique for describing the semantics of concurrent languages with state relies on a syntactic category of configurations, which consist of a store paired with a set of runnable threads. To mimic this technique, we identify sets of terms that represent threads, stores, and configurations.

We begin with a grammar for threads, terms representing a single flow of control:

Threads

$e ::=$	elementary threads
$\quad u$	$\quad$ result
$\quad u.\ell$	$\quad$ method select
$\quad u.\ell \Leftarrow \varsigma(x)b$	$\quad$ method update
$\quad \mathit{clone}(u)$	$\quad$ cloning
$\quad \mathit{acquire}(u)$	$\quad$ mutex acquisition
$\quad \mathit{release}(u)$	$\quad$ mutex release
$t ::= e \mid \mathit{let}\ x{=}t\ \mathit{in}\ b$	threads

To define configurations, let σ range over a sequence of denominations $p_i \mapsto d_i{}^{i \in 1..n}$, which we call a *store*, and let ρ range over a sequence of threads, $t_1, \ldots, t_n$. Then let a *configuration*, $(\nu\vec{q})\langle p_i \mapsto d_i{}^{\,i \in 1..m} \parallel t_1, \ldots, t_n\rangle$, be an abbreviation for the term $(\nu\vec{q})(p_1 \mapsto d_1 \Rsh \cdots \Rsh p_m \mapsto d_m \Rsh t_1 \Rsh \cdots \Rsh t_n)$. This notation is well defined only if $m + n > 0$. Intuitively, a configuration is a term consisting of a possibly multi-threaded computation $a_1 \Rsh \cdots \Rsh a_n$ interacting with a store $p_1 \mapsto d_1 \Rsh \cdots \Rsh p_m \mapsto d_m$, with the names $\vec{q}$ hidden from its environment.

We may transform any term into a configuration as follows:

Normalising terms to configurations

$\mathcal{N}(e)$	$\triangleq$	$\langle \varnothing \parallel e\rangle$
$\mathcal{N}(p \mapsto d)$	$\triangleq$	$\langle p \mapsto d \parallel \varnothing\rangle$
$\mathcal{N}(\mathit{let}\ x{=}a\ \mathit{in}\ b)$	$\triangleq$	$\langle \sigma \parallel \rho, \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle$
		where $\mathcal{N}(a) = (\nu\vec{p})\langle \sigma \parallel \rho, t\rangle$ and $\{\vec{p}\} \cap \mathit{fn}(b) = \varnothing$
$\mathcal{N}((\nu p)a)$	$\triangleq$	$(\nu p)\mathcal{N}(a)$
$\mathcal{N}(a \Rsh b)$	$\triangleq$	$(\nu\vec{p})(\nu\vec{q})\langle \sigma, \sigma' \parallel \rho, \rho'\rangle$
		where $\mathcal{N}(a) = (\nu\vec{p})\langle \sigma, \rho\rangle$, $\mathcal{N}(b) = (\nu\vec{q})\langle \sigma', \rho'\rangle$, and
		$\{\vec{p}\} \cap (\mathit{fn}(\sigma') \cup \mathit{fn}(\rho')) = \{\vec{q}\} \cap (\mathit{fn}(\sigma) \cup \mathit{fn}(\rho)) = \varnothing$

We can show by induction on the derivation of $a : T$, that $a : T$ implies that $\mathcal{N}(a)$ is well defined and in particular that $T = \mathit{Exp}$ implies that $\mathcal{N}(a)$ takes the form $(\nu\vec{p})\langle \sigma \parallel \rho, t\rangle$.

The two interesting cases of the definition are for lets and parallel compositions. When computing $\mathcal{N}(\mathit{let}\ x{=}a\ \mathit{in}\ b)$, we normalise a and b and pull the restrictions, store and extra threads from a outside the let. It is in this case that we need *let x=a in b* to be well-formed; if so, we have $a : \mathit{Exp}$, which implies that there is at least one thread in $\mathcal{N}(a)$. When computing $\mathcal{N}(a \Rsh b)$, we normalise a and b and concatenate the stores from $\mathcal{N}(a)$ and $\mathcal{N}(b)$ to produce the new store, and concatenate the thread lists to form the new thread list. We pull the restrictions from $\mathcal{N}(a)$ and $\mathcal{N}(b)$ to the outside; the conditions on the restricted names ensure there are no name clashes in the combined store.

We can show that there is a configuration structurally congruent to every expression, and normalisation is the identity function on configurations:

Lemma 2 *If $a : \mathit{Exp}$ then $\mathcal{N}(a) \equiv a$.*

Lemma 3 $\mathcal{N}((\nu\vec{p})\langle\sigma \parallel \rho\rangle) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$.

Having mimicked configurations within our syntax of terms, we may define a fairly conventional structural operational semantics, $a \xrightarrow{SOS} b$, as follows:

Structural operational semantics

(SOS Select) (where $\{\vec{p}\} \cap \mathit{fn}(\sigma, \rho_1, \rho_2) = \varnothing$)

$$\frac{\sigma = \sigma_1, p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{i\in 1..n}], \sigma_2 \quad j \in 1..n \quad \mathcal{N}(b_j\{\!\{x_j \leftarrow p\}\!\}) = (\nu\vec{p})\langle\sigma' \parallel \rho'\rangle}{\langle\sigma \parallel \rho_1, p.\ell_j, \rho_2\rangle \xrightarrow{SOS} (\nu\vec{p})\langle\sigma, \sigma' \parallel \rho_1, \rho', \rho_2\rangle}$$

(SOS Update)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i{}^{i\in 1..n}] \quad d' = [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i{}^{i\in(1..n)-\{j\}}]}{\langle\sigma_1, p \mapsto d, \sigma_2 \parallel \rho_1, p.\ell_j \Leftarrow \varsigma(x)b, \rho_2\rangle \xrightarrow{SOS} \langle\sigma_1, p \mapsto d', \sigma_2 \parallel \rho_1, p, \rho_2\rangle}$$

(SOS Clone) (where $q \notin \mathit{fn}(\sigma, \rho_1, \rho_2)$)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i{}^{i\in 1..n}] \quad \sigma = \sigma_1, p \mapsto d, \sigma_2}{\langle\sigma \parallel \rho_1, \mathit{clone}(p), \rho_2\rangle \xrightarrow{SOS} (\nu q)\langle\sigma, q \mapsto d \parallel \rho_1, q, \rho_2\rangle}$$

(SOS Acquire)

$$\frac{}{\langle\sigma_1, p \mapsto \mathit{unlocked}, \sigma_2 \parallel \rho_1, \mathit{acquire}(p), \rho_2\rangle \xrightarrow{SOS} \langle\sigma_1, p \mapsto \mathit{locked}, \sigma_2 \parallel \rho_1, p, \rho_2\rangle}$$

(SOS Release)

$$\frac{d \in \{\mathit{locked}, \mathit{unlocked}\}}{\langle\sigma_1, p \mapsto d, \sigma_2 \parallel \rho_1, \mathit{release}(p), \rho_2\rangle \xrightarrow{SOS} \langle\sigma_1, p \mapsto \mathit{unlocked}, \sigma_2 \parallel \rho_1, p, \rho_2\rangle}$$

(SOS Let Result) (where $\{\vec{p}\} \cap \mathit{fn}(\sigma, \rho_1, \rho_2) = \varnothing$)

$$\frac{\mathcal{N}(b\{\!\{x \leftarrow p\}\!\}) = (\nu\vec{p})\langle\sigma' \parallel \rho'\rangle}{\langle\sigma \parallel \rho_1, \mathit{let}\ x{=}p\ \mathit{in}\ b, \rho_2\rangle \xrightarrow{SOS} (\nu\vec{p})\langle\sigma, \sigma' \parallel \rho_1, \rho', \rho_2\rangle}$$

(SOS Let) (where $\{\vec{p}\} \cap \mathit{fn}(\rho_1, b, \rho_2) = \varnothing$)

$$\frac{\langle \sigma \parallel t \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho', t' \rangle}{\langle \sigma \parallel \rho_1, \mathit{let}\ x{=}t\ \mathit{in}\ b, \rho_2 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho_1, \rho', \mathit{let}\ x{=}t'\ \mathit{in}\ b, \rho_2 \rangle}$$

(SOS Res)

$$\frac{a \xrightarrow{SOS} (\nu\vec{p})(\sigma \parallel \rho)}{(\nu p)a \xrightarrow{SOS} (\nu p)(\nu\vec{p})(\sigma \parallel \rho)}$$

(SOS Norm)

$$\frac{\mathcal{N}(a) \xrightarrow{SOS} (\nu\vec{p})\langle \sigma \parallel \rho \rangle}{a \xrightarrow{SOS} (\nu\vec{p})\langle \sigma \parallel \rho \rangle}$$

There are many examples of semantics of this kind in the literature, such as the semantics by Di Blasio and Fisher (1996) for their calculus of concurrent objects. We may show for any derivation of $a \xrightarrow{SOS} b$ that b is a configuration; moreover, a is also a configuration, unless the last rule in the derivation is (SOS Norm). The purpose of (SOS Norm) is to allow reduction of arbitrary terms. The other rules correspond to the reduction rules for threads in Section 2.3 and Section 3.3, except that rules (SOS Select) and (SOS Let Result) use the normalisation function so that their outcome is a configuration.

Our main theorem about the structural operational semantics is that it coincides with the reduction semantics up to structural congruence. We write $a \xrightarrow{SOS}\equiv b$ to mean there is c such that $a \xrightarrow{SOS} c$ and $c \equiv b$.

Theorem 1 *For all $a, b : \mathit{Exp}$, $a \rightarrow b$ if and only $a \xrightarrow{SOS}\equiv b$.*

This is instructive for two reasons. First, the theorem legitimates our chemical-style reduction relation by demonstrating its correspondence, modulo structural congruence, to a rather more conventional semantics. Second, the theorem suggests a procedure for discovering all possible reductions of an expression: normalise the expression, then see what $\xrightarrow{SOS}$ reductions are derivable. It is not obvious how to use the $\rightarrow$ relation directly to discover all possible reductions of an expression, since they are defined up to structural congruence.

Theorem 1 fails to hold for processes that are not expressions. Consider the process $p.\ell \Rsh p \mapsto [\ell = \varsigma(s)s]$. This term has type *Proc* but not *Exp*. It has no reductions, because composition is not commutative. On the other hand, it is normalised to a configuration $\langle p \mapsto [\ell = \varsigma(s)s] \parallel p.\ell \rangle$ and we have $\langle p \mapsto [\ell = \varsigma(s)s] \parallel p.\ell \rangle \xrightarrow{SOS} \langle p \mapsto [\ell = \varsigma(s)s] \parallel p \rangle$.

The difficulty here is that the reduction relation $a \rightarrow b$ does not represent all of the behaviour of processes that are running as subterms to the left of a composition, where composition is commutative. To remedy this situation, we define versions of structural congruence and reduction specialised to processes situated to the left of a composition. Let $a \overset{Proc}{\equiv} b$ if and only if there is $p \notin fn(a) \cup fn(b)$ such that $a \upharpoonright p \equiv b \upharpoonright p$. Roughly, $\overset{Proc}{\equiv}$ is the same as $\equiv$, except that composition is commutative at the top level. Let $a \overset{Proc}{\rightarrow} b$ if and only if $a \overset{Proc}{\equiv} a'$, $a' \rightarrow b'$, and $b' \overset{Proc}{\equiv} b$. (An alternative definition is to specify these relations by a set of inference rules, simultaneously with the definitions of $a \equiv b$ and $a \rightarrow b$.) We can show that $a \upharpoonright b \overset{Proc}{\equiv} b \upharpoonright a$ and that $p.\ell \upharpoonright p \mapsto [\ell = \varsigma(s)s] \overset{Proc}{\rightarrow} p \upharpoonright p \mapsto [\ell = \varsigma(s)s]$. Moreover, we have:

Proposition 4 *For all $a, b : Proc$, $a \overset{Proc}{\rightarrow} b$ if and only if $a \xrightarrow{SOS} \overset{Proc}{\equiv} b$.*

4.3 A Single-Threaded Fragment

In this section, we adapt the type system from Section 4.1 to identity a deterministic single-threaded fragment of **concς**, and show that it includes Abadi and Cardelli's **impς**-calculus.

It is only the rule (Well Concur) from the type system in Section 4.1 that allows for multi-threaded computations. To see this, let the *single-threaded type system* for **concς** be the judgment $a :^1 T$ defined by the typing rules from Section 4.1, omitting (Well Concur), (Well Mutex), (Well Acquire), and (Well Release). We can show for every thread t that $t :^1 T$ implies that $T = Exp$. Therefore a binary composition of threads $t_1 \upharpoonright t_2$ cannot be typed in this system, since the rule (Well Par) requires $t_1 :^1 Proc$.

The single-threaded type system enjoys the following properties:

Lemma 5

(1) *If $a :^1 T$ and $a \equiv b$ then $b :^1 T$.*

(2) *If $a :^1 T$ and $a \rightarrow b$ then $b :^1 T$.*

(3) *For all $a, b :^1 Exp$, $a \rightarrow b$ if and only $a \xrightarrow{SOS}\equiv b$.*

(4) *If $a :^1 Proc$ then $\mathcal{N}(a)$ takes the form $(\nu\vec{p})\langle\sigma \parallel \varnothing\rangle$.*

(5) *If $a :^1 Exp$ then $\mathcal{N}(a)$ takes the form $(\nu\vec{p})\langle\sigma \parallel t\rangle$.*

Using the lemma, we obtain that unlike the full calculus, the fragment specified by the single-threaded type system is deterministic:

Theorem 2 *Suppose $a :^1 Exp$. If $a \rightarrow a'$ and $a \rightarrow a''$ then $a' \equiv a''$.*

Recall that any term of the imperative object calculus **impς** may be expressed within **concς** using the abbreviations stated in Section 2.4.1.

Proposition 6 *If a represents a term of* **impς**, *we can derive* $a :^1 \mathit{Exp}$.

All this establishes that we can embed **impς** within a deterministic fragment of **concς** closed under reduction.

5 A First-Order Type System

We turn in this section to demonstrating that the typing rules for Abadi and Cardelli's type system $\mathbf{Ob_{1<:}}$ simply and smoothly extend to typing our concurrent object calculus.

5.1 Typing

The types of our type system consist of the first-order object types of Abadi and Cardelli's $\mathbf{Ob_{1<:}}$ together with types for mutexes, processes, and expressions:

Types and environments

$A, B ::= [\ell_i : A_i{}^{i \in 1..n}] \mid \mathit{Mutex} \mid \mathit{Proc} \mid \mathit{Exp}$	types
$E ::= \varnothing, v_1 : A_1, \ldots, v_n : A_n$	environments

As in the rudimentary type system, *Exp* is the type of expressions, terms expected to return results, and *Proc* is the type of processes, terms that may not be expected to return results. As in $\mathbf{Ob_{1<:}}$, $[\ell_i : A_i{}^{i \in 1..n}]$ is the type of objects with methods $\ell_1, \ldots, \ell_n$ returning results of types $A_1, \ldots, A_n$, respectively; we identify object types up reordering of their components. Finally, *Mutex* is the type of mutexes.

System $\mathbf{Ob_{1<:}}$ is based on four judgments, which we define inductively by the rules in the following table.

Judgments

$E \vdash \diamond$	E is a well-formed environment
$E \vdash A$	given E, type A is well-formed
$E \vdash A <: B$	given E, A is a subtype of B
$E \vdash a : A$	given E, term a has type A

Typing rules

(Env $\varnothing$)
$$\frac{}{\varnothing \vdash \diamond}$$

(Env u)
$$\frac{E \vdash A \quad u \notin \mathit{dom}(E)}{E, u : A \vdash \diamond}$$

(Type Object) (ℓ_i distinct)
$$\frac{E \vdash \diamond \quad E \vdash B_i{<:}\mathit{Exp} \quad \forall i \in 1..n}{E \vdash [\ell_i : B_i{}^{i \in 1..n}]}$$

(Type Mutex)
$$\frac{E \vdash \diamond}{E \vdash \mathit{Mutex}}$$

(Type Proc)
$$\frac{E \vdash \diamond}{E \vdash \mathit{Proc}}$$

(Type Exp)
$$\frac{E \vdash \diamond}{E \vdash \mathit{Exp}}$$

(Sub Refl)
$$\frac{E \vdash A}{E \vdash A{<:}A}$$

(Sub Trans)
$$\frac{E \vdash A{<:}B \quad E \vdash B{<:}C}{E \vdash A{<:}C}$$

(Sub Object) (ℓ_i distinct)
$$\frac{E \vdash \diamond \quad E \vdash B_i{<:}\mathit{Exp} \quad \forall i \in 1..n+m}{E \vdash [\ell_i : B_i{}^{i \in 1..n+m}]{<:}[\ell_i : B_i{}^{i \in 1..n}]}$$

(Sub Exp)
$$\frac{E \vdash A \quad A \neq \mathit{Proc}}{E \vdash A{<:}\mathit{Exp}}$$

(Sub Proc)
$$\frac{E \vdash \diamond}{E \vdash \mathit{Exp}{<:}\mathit{Proc}}$$

(Val Subsumption)
$$\frac{E \vdash a : A \quad E \vdash A{<:}B}{E \vdash a : B}$$

(Val u)
$$\frac{E, u : A, E' \vdash \diamond}{E, u : A, E' \vdash u : A}$$

(Val Object) (where $A = [\ell_i : B_i{}^{i \in 1..n}]$)
$$\frac{E = E_1, p : A, E_2 \quad E, x_i : A \vdash b_i : B_i \quad \mathit{dom}(b_i) = \varnothing \quad \forall i \in 1..n}{E \vdash p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}] : \mathit{Proc}}$$

(Val Mutex)
$$\frac{E \vdash p : \mathit{Mutex} \quad d \in \{\mathit{locked}, \mathit{unlocked}\}}{E \vdash p \mapsto d : \mathit{Proc}}$$

(Val Select)
$$\frac{E \vdash u : [\ell_i : B_i{}^{i \in 1..n}] \quad j \in 1..n}{E \vdash u.\ell_j : B_j}$$

(Val Update) (where $A = [\ell_i : B_i{}^{i \in 1..n}]$)
$$\frac{E \vdash u : A \quad j \in 1..n \quad E, x : A \vdash b : B_j \quad \mathit{dom}(b) = \varnothing}{E \vdash u.\ell_j \Leftarrow \varsigma(x)b : A}$$

(Val Clone)
$$\frac{E \vdash u : [\ell_i : B_i{}^{i \in 1..n}]}{E \vdash \mathit{clone}(u) : [\ell_i : B_i{}^{i \in 1..n}]}$$

(Val Acquire)
$$\frac{E \vdash u : \mathit{Mutex}}{E \vdash \mathit{acquire}(u) : \mathit{Mutex}}$$

(Val Release)
$$\frac{E \vdash u : \mathit{Mutex}}{E \vdash \mathit{release}(u) : \mathit{Mutex}}$$

(Val Let)

$$\frac{E \vdash a : A \quad E, x : A \vdash b : B \quad dom(b) = \varnothing \quad E \vdash A{<:}\mathit{Exp} \quad E \vdash B{<:}\mathit{Exp}}{E \vdash \mathit{let}\ x{=}a\ \mathit{in}\ b : B}$$

(Val Par) (where $dom(a) \cap dom(b) = \varnothing$)

$$\frac{E \vdash a : \mathit{Proc} \quad E \vdash b : B}{E \vdash a \Rsh b : B}$$

(Val Res)

$$\frac{E, p : A \vdash a : B \quad p \in dom(a)}{E \vdash (\nu p)a : B}$$

This type system combines Abadi and Cardelli's $\mathbf{Ob_{1<:}}$ and the rudimentary type system from Section 4.1. The rules for well-formed environments are standard. (Type Object) is the only noteworthy rule for deriving well-formed types. It insists that the type of every method is a subtype of *Exp*; this corresponds to the restriction in (Well Object) that methods be of type *Exp*. There are two non-standard subtyping rules: (Sub Exp) ensures that object types and the type *Mutex* are subtypes of *Exp*, and (Sub Proc) ensures that every type is a subtype of *Proc*. The rules for typing terms are a straightforward combination of the rules of $\mathbf{Ob_{1<:}}$ and the rules from Section 4.1.

This type system refines the rudimentary type system of Section 4.1 in the following sense:

Lemma 7 *If $E \vdash a : A$ and $E \vdash A{<:}T$ then $a : T$.*

Our typing rules respect structural congruence and reduction:

Theorem 3

(1) *If $E \vdash a : A$ and $a \equiv b$ then $E \vdash b : A$.*

(2) *If $E \vdash a : A$ and $a \to b$ then $E \vdash b : A$.*

To prove such a subject reduction theorem for typed forms of **impς**, Abadi and Cardelli need to introduce the standard auxiliary notion of store typing. Since the terms of our calculus include both sequential threads and stores, we have no need to separate the notion of store typing from the notion of a typable term. The outcome is a crisper statement of subject reduction than for the imperative form of $\mathbf{Ob_{1<:}}$ in Abadi and Cardelli's book.

5.2 Examples of Typing and Subtyping

Let $A \to B$ be short for $[\mathit{arg} : A, \mathit{val} : B]$, as usual in object calculi. If we make the definitions,

$$\begin{aligned} \mathit{Chan}_s\, A \;\triangleq\; & [\mathit{reader} : \mathit{Mutex}, \mathit{writer} : \mathit{Mutex}, \\ & \;\mathit{val} : A, \mathit{read} : A, \mathit{write} : A \to A] \end{aligned}$$

$$\begin{aligned} \mathit{Chan}_a\, A \;\triangleq\; & [\mathit{ch} : \mathit{Chan}_s, \mathit{ack} : \mathit{Mutex}, \mathit{writeLock} : \mathit{Mutex}, \\ & \mathit{val} : A, \mathit{read} : A, \mathit{write} : A \to A] \end{aligned}$$

we may derive:

$$\begin{gathered} \varnothing, \mathit{nil} : A \vdash \mathit{newChan}_s : \mathit{Chan}_s\, A \\ \varnothing, \mathit{nil} : A \vdash \mathit{newChan}_a : \mathit{Chan}_a\, A \end{gathered}$$

These typings expose more of the internal state of channels than is desirable. Let $\updownarrow A$ be the type $[\mathit{read} : A, \mathit{write} : A \to A]$. Since both $\mathit{Chan}_s\, A <: \updownarrow A$ and $\mathit{Chan}_s\, A <: \updownarrow A$, we may use subsumption to derive:

$$\begin{gathered} \varnothing, \mathit{nil} : A \vdash \mathit{newChan}_s : \updownarrow A \\ \varnothing, \mathit{nil} : A \vdash \mathit{newChan}_a : \updownarrow A \end{gathered}$$

To further refine usage of these channel types we define a type of write-only channels, $\uparrow A = [\mathit{write} : A \to A]$, and a type of read-only channels, $\downarrow A = [\mathit{read} : A]$, as in the work of Pierce and Sangiorgi (1996). The inclusions $\updownarrow A <: \uparrow A$ and $\updownarrow A <: \downarrow A$ are part of the definition of Pierce and Sangiorgi's system but are derivable in ours.

6 Conclusions

We described a concurrent extension of Abadi and Cardelli's imperative object calculus, **impς**. The syntax of our calculus is essentially that of **impς** together with parallel composition and restriction from the π-calculus, and new primitives for synchronisation via mutexes. This syntax is extremely expressive; in a precise sense it unifies notions of expression, process, store, thread, and configuration. We presented a novel reduction semantics for concurrent expressions, without any need for evaluation contexts, and proved that it corresponds to a more conventional structural operational semantics defined in terms of configurations. We exhibited translations of the asynchronous π-calculus and the **impς**-calculus into our calculus.

One of Abadi and Cardelli's notable achievements in their theory of objects is a range of type systems that allow type-checking of various styles of object-oriented programming. By studying one of their standard type systems we demonstrated that our semantic techniques allow their type systems to be smoothly extended to encompass concurrency.

An important avenue for future work is the study of observational equivalence for our calculus. Another avenue to investigate is the encoding of other concurrency primitives, like monitors, condition variables, and named threads. Finally, it would be valuable to extend our semantics of expression-based concurrency to handle the mobile processes found in object-oriented languages like Telescript or Obliq.

Acknowledgements

Thanks to Alan Jeffrey and Søren Lassen for useful conversations about concurrent objects. Martín Abadi, Luca Cardelli, Søren Lassen, and Andy Pitts commented on a draft of this paper.

This work was supported by a Royal Society University Research Fellowship, an EPSRC Research Studentship, and by the EPSRC project "An Operational Theory of Objects". Gordon's current affiliation is Microsoft Research.

References

Abadi, M. and L. Cardelli (1996). *A Theory of Objects*. Springer-Verlag.

Agha, G., I. Mason, S. Smith, and C. Talcott (1997, January). A foundation for actor computation. *Journal of Functional Programming 7*(1).

Amadio, R., L. Leth, and B. Thomsen (1995). From a concurrent λ-calculus to the π-calculus. In *Proceedings Foundations of Computation Theory 95*, Volume 965 of *Lecture Notes in Computer Science*. Springer-Verlag.

Berry, D., R. Milner, and D. N. Turner (1992). A semantics for ML concurrency primitives. In *Proceedings POPL'92*, pp. 119–129.

Berry, G. and G. Boudol (1992, April). The chemical abstract machine. *Theoretical Computer Science 96*(1), 217–248.

Boudol, G. (1997). The π-calculus in direct style. In *Proceedings POPL'97*, pp. 228–241.

Dal Zilio, S. (1998). Concurrent objects in the blue calculus. Draft.

Di Blasio, P. and K. Fisher (1996, August). A calculus for concurrent objects. In *Proceedings CONCUR'96*.

Dijkstra, E. W. (1965, September). Solution of a problem in concurrent programming control. *Communications of the ACM 8*(9), 569.

Ferreira, W., M. Hennessy, and A. Jeffrey (1995). A theory of weak bisimulation for core CML. Technical Report 95:05, Computer Science, School of Cognitive and Computing Sciences, University of Sussex.

Fournet, C. and G. Gonthier (1996, January). The reflexive CHAM and the Join-calculus. In *Proceedings POPL'96*, pp. 372–385.

Gordon, A. D., P. D. Hankin, and S. B. Lassen (1997). Compilation and equivalence of imperative objects. In *Proceedings FST&TCS'97*, Volume 1346 of *Lecture Notes in Computer Science*, pp. 74–87. Springer-Verlag. Full version available as Technical Report 429, University of Cambridge Computer Laboratory, 1997.

Havelund, K. and K. Larsen (1993, July). The fork calculus. In A. Lingas, R. Karlsson, and S. Carlsson (Eds.), *Proceedings ICALP'93*, Volume 700 of *Lecture Notes in Computer Science*. Springer-Verlag.

Honda, K. and M. Tokoro (1991). An object calculus for asynchronous communication. In *Proceedings ECOOP'91*, Volume 512 of *Lecture Notes in Computer Science*, pp. 133–147. Springer-Verlag.

Jones, C. (1993). A pi-calculus semantics for an object-based design notation. In *Proceedings CONCUR'93*, Volume 715 of *Lecture Notes in Computer Science*, pp. 158–172. Springer-Verlag.

Kleist, J. and D. Sangiorgi (1998). Imperative objects and mobile processes. In *Proceedings PROCOMET'98*.

Lamport, L. (1985, November). A fast mutual exclusion algorithm. Technical Report 7, Digital Systems Research Center.

Milner, R. (1992). Functions as processes. *Mathematical Structures in Computer Science 2*, 119–141.

Milner, R., J. Parrow, and D. Walker (1992). A calculus of mobile processes, parts I and II. *Information and Computation 100*, 1–77.

Moggi, E. (1989). Notions of computations and monads. *Information and Computation 93*, 55–92.

Peyton Jones, S. L., A. D. Gordon, and S. Finne (1996). Concurrent Haskell. In *Proceedings POPL'96*, pp. 295–308.

Pierce, B. C. and D. Sangiorgi (1996). Typing and subtyping for mobile processes. *Mathematical Structures in Computer Science 6*(5), 409–454.

Pierce, B. C. and D. N. Turner (1995). Concurrent objects in a process calculus. In *Proceedings TPPP'94*, Volume 907 of *Lecture Notes in Computer Science*, pp. 187–215. Springer-Verlag.

Pierce, B. C. and D. N. Turner (1997). Pict: A programming language based on the pi-calculus. Technical Report CSCI 476, Computer Science Department, Indiana University. To appear in *Proof, Language and Interaction: Essays in Honour of Robin Milner*, G. Plotkin, C. Stirling, and M. Tofte, editors, MIT Press, 1998.

Plotkin, G. D. (1981, September). A structural approach to operational semantics. Technical Report FN–19, DAIMI, Aarhus University.

Reppy, J. H. (1992). *Higher-Order Concurrency.* Ph. D. thesis, Department of Computer Science, Cornell University. Available as Technical Report 92–1285.

Vasconcelos, V. T. (1994). Typed concurrent objects. In *Proceedings ECOOP'94.*

Walker, D. (1995, February). Objects in the pi-calculus. *Information and Computation 116*(2), 253–271.

A Proofs

In this appendix we prove all the results stated in the body of the paper.

We defined the structural congruence relation $a \equiv b$ and the reduction relation $a \to b$ by rules in Sections 2.3 and 3.3. In Section 4.2 we defined *Proc*-indexed relations $a \overset{\mathit{Proc}}{\equiv} b$ and $a \overset{\mathit{Proc}}{\to} b$. For stating some results in this appendix, it is convenient to introduce the notations $a \overset{\mathit{Exp}}{\equiv} b$ and $a \overset{\mathit{Exp}}{\to} b$ as shorthands for $a \equiv b$ and $a \to b$ respectively. The following table summarises these T-indexed notations for structural congruence and reduction:

Relations $\overset{T}{\equiv}$ and $\overset{T}{\to}$ where $T ::= \mathit{Exp} \mid \mathit{Proc}$

$$\begin{aligned}
a \overset{\mathit{Exp}}{\equiv} b &\triangleq a \equiv b \\
a \overset{\mathit{Proc}}{\equiv} b &\triangleq \exists p \notin \mathit{fn}(a) \cup \mathit{fn}(b)(a \Rsh p \equiv b \Rsh p) \\
a \overset{\mathit{Exp}}{\to} b &\triangleq a \to b \\
a \overset{\mathit{Proc}}{\to} b &\triangleq \exists a', b'(a \overset{\mathit{Proc}}{\equiv} a', a' \to b', b' \overset{\mathit{Proc}}{\equiv} b)
\end{aligned}$$

A.1 Facts about Structural Congruence

The main body of the paper relies on an operation, $b\{\!\!\{y \leftarrow v\}\!\!\}$, which denotes the outcome of substituting the result v for each free occurrence of the variable y in the term b. To state the following lemma we extend this operation to allow the substitution of a term for a result (either a name or a variable).

Substitution of a term for a result: $b\{\!\!\{v \leftarrow c\}\!\!\}$

$$\begin{aligned}
u\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq c \text{ if } u = v \text{ or } u \text{ if } u \neq v \\
(p \mapsto [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}])\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq p \mapsto [\ell_i = \varsigma(x_i)(b_i\{\!\!\{v \leftarrow c\}\!\!\})\ ^{i \in 1..n}] \\
&\qquad \text{for } x_i \notin \{v\} \cup \mathit{fv}(c) \\
(p \mapsto \mathit{locked})\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq p \mapsto \mathit{locked} \\
(p \mapsto \mathit{unlocked})\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq p \mapsto \mathit{unlocked} \\
(u.\ell)\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq (u\{\!\!\{v \leftarrow c\}\!\!\}).\ell \\
(u.\ell \Leftarrow \varsigma(x)b)\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq (u\{\!\!\{v \leftarrow c\}\!\!\}).\ell \Leftarrow \varsigma(x)(b\{\!\!\{v \leftarrow c\}\!\!\}) \\
&\qquad \text{for } x \notin \{v\} \cup \mathit{fv}(c) \\
(\mathit{clone}(u))\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq \mathit{clone}(u\{\!\!\{v \leftarrow c\}\!\!\}) \\
(\mathit{let}\ x{=}a\ \mathit{in}\ b)\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq \mathit{let}\ x{=}a\{\!\!\{v \leftarrow c\}\!\!\}\ \mathit{in}\ (b\{\!\!\{v \leftarrow c\}\!\!\}) \\
&\qquad \text{for } x \notin \{v\} \cup \mathit{fv}(c) \\
(a \Rsh b)\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq (a\{\!\!\{v \leftarrow c\}\!\!\}) \Rsh (b\{\!\!\{v \leftarrow c\}\!\!\}) \\
((\nu p)a)\{\!\!\{v \leftarrow c\}\!\!\} &\triangleq (\nu p)(a\{\!\!\{v \leftarrow c\}\!\!\}) \quad \text{for } p \notin \{v\} \cup \mathit{fn}(c)
\end{aligned}$$

This definition depends on the shorthands $a.\ell$, $a.\ell \Leftarrow \varsigma(x)b$, $\mathit{clone}(a)$, $\mathit{acquire}(a)$ and $\mathit{release}(a)$ for an arbitrary term a, defined in Sections 2.4.1 and 3.1. For example, $\mathit{clone}(x)\{\!\{x \leftarrow p.\ell\}\!\}$ is defined to be $\mathit{clone}(p.\ell)$, which is a shorthand for $\mathit{let}\ y{=}p.\ell\ \mathit{in}\ \mathit{clone}(y)$.

Lemma 8

(1) *If* $a \overset{T}{\equiv} b$ *then* $a\{\!\{u \leftarrow v\}\!\} \overset{T}{\equiv} b\{\!\{u \leftarrow v\}\!\}$.

(2) *The relation* $\overset{\mathit{Proc}}{\equiv}$ *is reflexive, symmetric, and transitive.*

(3) *If* $a \overset{\mathit{Proc}}{\equiv} b$ *then* $(\nu p)a \overset{\mathit{Proc}}{\equiv} (\nu p)b$.

(4) *If* $a \equiv b$ *then* $a\{\!\{v \leftarrow c\}\!\} \equiv b\{\!\{v \leftarrow c\}\!\}$.

(5) *If* $a \overset{\mathit{Proc}}{\equiv} a'$ *and* $b \overset{T}{\equiv} b'$ *then* $a \upharpoonright b \overset{T}{\equiv} a' \upharpoonright b'$.

(6) $a \overset{\mathit{Proc}}{\equiv} b$ *if and only if, for all* $p \notin \mathit{fn}(a) \cup \mathit{fn}(b)$, $a \upharpoonright p \equiv b \upharpoonright p$.

Proof

(1) In the case $T = \mathit{Exp}$, we may prove the lemma by an easy induction on the derivation of $a \equiv b$.

In the case $T = \mathit{Proc}$, we have $a \upharpoonright p \equiv b \upharpoonright p$ for some $p \notin \mathit{fn}(a, b)$. Pick $q \notin \mathit{fn}(a, b) \cup \{u, v\}$. Then by the Exp case we have $(a \upharpoonright p)\{\!\{p \leftarrow q\}\!\} \equiv (b \upharpoonright p)\{\!\{p \leftarrow q\}\!\}$, that is, $a \upharpoonright q \equiv b \upharpoonright q$. Using the Exp case again, we get $(a \upharpoonright q)\{\!\{u \leftarrow v\}\!\} \equiv (b \upharpoonright q)\{\!\{u \leftarrow v\}\!\}$, that is, $a\{\!\{u \leftarrow v\}\!\} \upharpoonright q \equiv b\{\!\{u \leftarrow v\}\!\} \upharpoonright q$. Since $q \notin \mathit{fn}(a\{\!\{u \leftarrow v\}\!\}, b\{\!\{u \leftarrow v\}\!\})$ we have $a \overset{\mathit{Proc}}{\equiv} b$ as required.

(2) Reflexivity and symmetry are clear. For transitivity, suppose $a \overset{\mathit{Proc}}{\equiv} b$ and $b \overset{\mathit{Proc}}{\equiv} c$. This means $a \upharpoonright p \equiv b \upharpoonright p$ and $b \upharpoonright q \equiv c \upharpoonright q$ for some names p and q where $p \notin \mathit{fn}(a, b)$ and $q \notin \mathit{fn}(b, c)$. Pick $r \notin \mathit{fn}(a, b, c)$. Then by part (1), $(a \upharpoonright p)\{\!\{p \leftarrow r\}\!\} \equiv (b \upharpoonright p)\{\!\{p \leftarrow r\}\!\}$, that is, $a \upharpoonright r \equiv b \upharpoonright r$. We may similarly infer that $b \upharpoonright r \equiv c \upharpoonright r$. By (Struct Trans) we deduce $a \upharpoonright r \equiv c \upharpoonright r$, so $a \overset{\mathit{Proc}}{\equiv} c$.

(3) If $a \overset{\mathit{Proc}}{\equiv} b$ then $a \upharpoonright q \equiv b \upharpoonright q$ for some $q \notin \mathit{fn}(a, b)$. Pick r such that $r \notin \mathit{fn}(a, b) \cup \{p\}$. Then $(a \upharpoonright q)\{\!\{q \leftarrow r\}\!\} \equiv (b \upharpoonright q)\{\!\{q \leftarrow r\}\!\}$ by part (1). This means $a \upharpoonright r \equiv b \upharpoonright r$. Then by (Struct Res), $(\nu p)(a \upharpoonright r) \equiv (\nu p)(b \upharpoonright r)$. Since $r \neq p$ we have by (Struct Par 1), $(\nu p)(a \upharpoonright r) \equiv (\nu p)a \upharpoonright r$ and $(\nu p)(b \upharpoonright r) \equiv (\nu p)b \upharpoonright r$. By (Struct Trans), $(\nu p)a \upharpoonright r \equiv (\nu p)b \upharpoonright r$. Since $r \notin \mathit{fn}((\nu p)a) \cup \mathit{fn}((\nu p)b)$, we have $(\nu p)a \overset{\mathit{Proc}}{\equiv} (\nu p)b$.

(4) An easy induction on the derivation of $a \equiv b$.

(5) If $T = \mathit{Exp}$, we have $a \upharpoonright p \equiv a' \upharpoonright p$ for some $p \notin \mathit{fn}(a, a')$, and $b \equiv b'$. By (4), $a \upharpoonright b \equiv a' \upharpoonright b$. By (Struct Par), $a' \upharpoonright b \equiv a' \upharpoonright b'$. Hence by (Struct Trans), $a \upharpoonright b \equiv a' \upharpoonright b'$. Otherwise, if $T = \mathit{Proc}$, we get $a \upharpoonright p \equiv a' \upharpoonright p$ and $b \upharpoonright p \equiv b' \upharpoonright p$ for some p such that $p \notin \mathit{fn}(a, a', b, b')$. Then we compute:

$$\begin{array}{rcll} a \upharpoonright b \upharpoonright p & \equiv & a \upharpoonright b' \upharpoonright p & \text{by (Struct Par)} \\ & \equiv & b' \upharpoonright a \upharpoonright p & \text{by (Struct Par Comm)} \\ & \equiv & b' \upharpoonright a' \upharpoonright p & \text{by (Struct Par)} \\ & \equiv & a' \upharpoonright b' \upharpoonright p & \text{by (Struct Par Comm)} \end{array}$$

Hence by (Struct Trans), $a \upharpoonright b \upharpoonright p \equiv a' \upharpoonright b' \upharpoonright p$ and hence $a \upharpoonright b \overset{\mathit{Proc}}{\equiv} a' \upharpoonright b'$.

(6) If $a \overset{\mathit{Proc}}{\equiv} b$ then there is $q \notin \mathit{fn}(a, b)$ such that $a \upharpoonright q \equiv b \upharpoonright q$. Now suppose $p \notin \mathit{fn}(a, b)$. By (4) we have $(a \upharpoonright q)\{\!\{q \leftarrow p\}\!\} \equiv (b \upharpoonright q)\{\!\{q \leftarrow p\}\!\}$ and hence $a \upharpoonright p \equiv b \upharpoonright p$. The converse, namely that $a \upharpoonright p \equiv b \upharpoonright p$ for all $p \notin \mathit{fn}(a, b)$ implies $a \upharpoonright p \equiv b \upharpoonright p$ for some $p \notin \mathit{fn}(a, b)$ is clear. □

The last part of the lemma allows us to derive $a \upharpoonright p \equiv b \upharpoonright p$ for any fresh p when we know $a \overset{\mathit{Proc}}{\equiv} b$. We will use this implicitly in proofs without referencing the lemma.

A.2 Proof of Proposition 1

Our aim in this section is to prove Proposition 1, subject reduction for the rudimentary type system. We begin with two preliminary lemmas.

Lemma 9

(1) *For all terms a, $\mathit{dom}(a) \subseteq \mathit{fn}(a)$.*

(2) *If $a \equiv b$ then $a \overset{\mathit{Proc}}{\equiv} b$.*

(3) *If $a \to b$ then $a \overset{\mathit{Proc}}{\to} b$.*

(4) *If $a \equiv b$ then $\mathit{dom}(a) = \mathit{dom}(b)$.*

(5) *If $a \to b$ then $\mathit{dom}(a) = \mathit{dom}(b)$.*

(6) *If $a : T$ then $a\{\!\{x \leftarrow u\}\!\} : T$.*

Proof

(1) By induction on the structure of a.

(2) The relation $\overset{\mathit{Proc}}{\equiv}$ is defined by $a \overset{\mathit{Proc}}{\equiv} b$ if and only if $a \Rsh p \equiv b \Rsh p$ for some $p \notin \mathit{fn}(a)$. Now, if $a \equiv b$ then $a \Rsh p \equiv b \Rsh p$ by (Struct Par) since $p \equiv p$ by (Struct Refl).

(3) The relation $\overset{\mathit{Proc}}{\rightarrow}$ is defined by $a \overset{\mathit{Proc}}{\rightarrow} b$ if and only if $a \overset{\mathit{Proc}}{\equiv} a' \rightarrow b' \overset{\mathit{Proc}}{\equiv} b$. If $a \rightarrow b$ then by reflexivity of $\overset{\mathit{Proc}}{\equiv}$, $a \overset{\mathit{Proc}}{\equiv} a \rightarrow b \overset{\mathit{Proc}}{\equiv} b$, so $a \overset{\mathit{Proc}}{\rightarrow} b$.

(4) By induction on the derivation of $a \equiv b$.

(5) By induction on the derivation of $a \rightarrow b$.

(6) By induction on the derivation of $a : T$. □

Lemma 10

(1) *If $a \Rsh b : T$ then $a : \mathit{Proc}$, $b : T$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$.*

(2) *If $\mathit{let}\ x{=}a\ \mathit{in}\ b : T$ then $a : \mathit{Exp}$, $b : \mathit{Exp}$ and $\mathit{dom}(b) = \varnothing$.*

(3) *If $(\nu p)a : T$ then $a : T$ and $p \in \mathit{dom}(a)$.*

(4) *If $p \mapsto [\ell_i = \varsigma(x_i)b_i\ {}^{i\in 1..n}] : T$ then $T = \mathit{Proc}$, $b_i : \mathit{Exp}$ and $\mathit{dom}(b_i) = \varnothing$ for all $i \in 1..n$.*

(5) *If $u.\ell \Leftarrow \varsigma(x)b : T$ then $b : \mathit{Exp}$ and $\mathit{dom}(b) = \varnothing$.*

Proof Each of these statements follows by an easy induction on the derivation of the typing derivation. □

The next two lemmas furnish the two parts of Proposition 1.

Lemma 11 *If $a : T$ and $a \equiv b$ then $b : T$.*

Proof We first symmetrise the lemma:

(1) If $a : T$ and $a \equiv b$ then $b : T$.

(2) If $b : T$ and $a \equiv b$ then $a : T$.

We prove this by induction on the derivation of $a \equiv b$. We consider each of the rules which may derive $a \equiv b$ in turn:

(Struct Refl) We have $a \equiv a$, and the result is trivial.

(Struct Symm) We have $a \equiv b$ obtained from $b \equiv a$. Because of the symmetrised form of the lemma, the result is trivial.

(Struct Trans) We have $a \equiv c$ obtained from $a \equiv b$ and $b \equiv c$. If $a : T$ then the induction hypothesis applied to $a \equiv b$ gives $b : T$. The induction hypothesis applied to $b \equiv c$ gives $c : T$. Conversely, if $c : T$ then we deduce $b : T$ from $b \equiv c$. Similarly, we deduce $a : T$ from $a \equiv b$.

(Struct Update) We have $u.\ell \Leftarrow \varsigma(x)b \equiv u.\ell \Leftarrow \varsigma(x)b'$ obtained from $b \equiv b'$. For part (1), if $u.\ell \Leftarrow \varsigma(x)b : T$ then by Lemma 10(5) we have $b : \mathit{Exp}$ and $\mathit{dom}(b) = \varnothing$. The induction hypothesis applied to $b \equiv b'$ gives $b' : \mathit{Exp}$ and Lemma 9(4) gives $\mathit{dom}(b') = \mathit{dom}(b) = \varnothing$. Hence by (Well Update) and (Well Concur) we deduce $u.\ell \Leftarrow \varsigma(x)b' : T$ for either T. Part (2) follows by symmetry.

(Struct Let) We have $\mathit{let}\ x{=}a\ \mathit{in}\ b \equiv \mathit{let}\ x{=}a'\ \mathit{in}\ b'$ obtained from $a \equiv a'$ and $b \equiv b'$. For part (1), if $\mathit{let}\ x{=}a\ \mathit{in}\ b : T$ then by Lemma 10(2) we get $a : \mathit{Exp}$, $b : \mathit{Exp}$ and $\mathit{dom}(b) = \varnothing$. The induction hypothesis applied to $a \equiv a'$ gives $a' : \mathit{Exp}$ and Lemma 9(4) gives $\mathit{dom}(a) = \mathit{dom}(a')$. The induction hypothesis applied to $b \equiv b'$ gives $b' : \mathit{Exp}$ and Lemma 9(4) gives $\mathit{dom}(b') = \mathit{dom}(b) = \varnothing$. Rules (Well Let) and (Well Concur) give $\mathit{let}\ x{=}a'\ \mathit{in}\ b' : T$ for either T. Part (2) follows by symmetry.

(Struct Res) We have $(\nu p)a \equiv (\nu p)a'$ obtained from $a \equiv a'$. For part (1), if $(\nu p)a : T$ then by Lemma 10(3), $a : T$ and $p \in \mathit{dom}(a)$. The induction hypothesis applied to $a \equiv a'$ gives $a' : T$ and Lemma 9(4) gives $\mathit{dom}(a') = \mathit{dom}(a)$. Rule (Well Res) gives $(\nu p)a' : T$ since $p \in \mathit{dom}(a')$. Part (2) follows by symmetry.

(Struct Par) We have $a \Rsh b \equiv a' \Rsh b'$ obtained from $a \equiv a'$ and $b \equiv b'$. For part (1), if $a \Rsh b : T$ then by Lemma 10 we have $a : \mathit{Proc}$, $b : T$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. The induction hypothesis applied to $a \equiv a'$ gives $a' : \mathit{Proc}$ and Lemma 9(4) gives $\mathit{dom}(a') = \mathit{dom}(a)$. The induction hypothesis applied to $b \equiv b'$ gives $b' : T$ and Lemma 9(4) gives $\mathit{dom}(b) = \mathit{dom}(b')$. Now, $\mathit{dom}(a') \cap \mathit{dom}(b') = \mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. Hence by (Well Par), $a' \Rsh b' : T$. Part (2) follows by symmetry.

(Struct Object) We have $p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i \in 1..n}] \equiv p \mapsto [\ell_i = \varsigma(x_i)b'_i{}^{\,i \in 1..n}]$ obtained from $b_i \equiv b'_i$ and $\mathit{dom}(b_i) = \varnothing$ for all $i \in 1..n$. For part (1), if $p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i \in 1..n}] : T$ then by Lemma 10(4) we have $T = \mathit{Proc}$, $\mathit{dom}(b_i) = \varnothing$ and $b_i : \mathit{Exp}$ for each $i \in 1..n$. The induction hypothesis applied to $b_i \equiv b'_i$ gives $b'_i : \mathit{Exp}$ for each $i \in 1..n$, and Lemma 9(4) gives $\mathit{dom}(b_i) = \varnothing$ for each $i \in 1..n$.

By (Well Object), $p \mapsto [\ell_i = \varsigma(x_i)b'_i{}^{\,i\in 1..n}] : \mathit{Proc}$, and $\mathit{dom}(p \mapsto [\ell_i = \varsigma(x_i)b'_i{}^{\,i\in 1..n}]) = \{p\} = \mathit{dom}(p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i\in 1..n}])$.

Part (2) follows by symmetry.

(Struct Par Assoc) We have $(a \Rsh b) \Rsh c \equiv a \Rsh (b \Rsh c)$. For part (1), if $(a \Rsh b) \Rsh c : T$ then by Lemma 10(1) we have $(a \Rsh b) : \mathit{Proc}$, $c : T$ and $\mathit{dom}(a \Rsh b) \cap \mathit{dom}(c) = \varnothing$. Similarly, Lemma 10(1) applied to $(a \Rsh b) : \mathit{Proc}$ gives $a : \mathit{Proc}$, $b : \mathit{Proc}$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. Since $\mathit{dom}(a \Rsh b) = \mathit{dom}(a) \cup \mathit{dom}(b)$ we have that the sets $\mathit{dom}(a)$, $\mathit{dom}(b)$ and $\mathit{dom}(c)$ are pairwise disjoint. Rule (Well Par) applied to $b : \mathit{Proc}$, $c : T$ and $\mathit{dom}(b) \cap \mathit{dom}(c) = \varnothing$ gives $b \Rsh c : T$. Rule (Well Par) applied to $a : \mathit{Proc}$, $(b \Rsh c) : T$ and $\mathit{dom}(a) \cap \mathit{dom}(b \Rsh c) = \varnothing$ gives $a \Rsh (b \Rsh c) : T$. Part (2) follows by a similar argument.

(Struct Par Comm) We have $(a \Rsh b) \Rsh c \equiv (b \Rsh a) \Rsh c$. For part (1), if $(a \Rsh b) \Rsh c : T$ we argue as in the (Struct Par Assoc) case, to deduce $a : \mathit{Proc}$, $b : \mathit{Proc}$, $c : T$ and that the sets $\mathit{dom}(a)$, $\mathit{dom}(b)$ and $\mathit{dom}(c)$ are pairwise disjoint. Applying (Well Par) twice we deduce $(b \Rsh a) \Rsh c : T$. Part (2) of the proposition follows by symmetry.

(Struct Res Res) We have $(\nu p)(\nu q)a \equiv (\nu q)(\nu p)a$. Assume for part (1) $(\nu p)(\nu q)a : T$. By renaming bound variables, we can assume without loss of generality that $p \neq q$. Applying Lemma 10(3) we get $a : T$ and $\{p, q\} \subseteq \mathit{dom}(a)$. Since $p \neq q$, $q \in \mathit{dom}(a)$ and $p \in (\mathit{dom}(a) - \{q\})$. Hence by (Well Res) (applied twice), $(\nu q)(\nu p)a : T$. Part (2) follows by symmetry.

(Struct Par 1) We have $(\nu p)(a \Rsh b) \equiv a \Rsh (\nu p)b$ obtained from $p \notin \mathit{fn}(a)$. For part (1), assume $(\nu p)(a \Rsh b) : T$. Parts (1) and (3) of Lemma 10 applied to this judgment yield $a : \mathit{Proc}$, $b : T$, $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$ and $p \in \mathit{dom}(a) \cup \mathit{dom}(b)$. Lemma 9(1) and $p \notin \mathit{fn}(a)$ imply $p \notin \mathit{dom}(a)$. Since $p \in \mathit{dom}(a) \cup \mathit{dom}(b)$ we must have $p \in \mathit{dom}(b)$. By (Well Res), $(\nu p)b : T$. Now, $\mathit{dom}(a) \cap \mathit{dom}((\nu p)b) = \mathit{dom}(a) \cap (\mathit{dom}(b) - \{p\}) = \varnothing$, since $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. Hence (Well Par) gives $a \Rsh (\nu p)b : T$.

Part (2) follows similarly. If $a \Rsh (\nu p)b : T$ then by Lemma 10, $a : \mathit{Proc}$, $b : T$, $p \in \mathit{dom}(b)$ and $\mathit{dom}(a) \cap (\mathit{dom}(b) - \{p\}) = \varnothing$. Hence $(\nu p)(a \Rsh b) : T$.

(Struct Par 2) Similar to (Struct Par 1).

(Struct Let Assoc) If $y \notin \mathit{fn}(c)$ we have $\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c \equiv \mathit{let}\ y{=}a\ \mathit{in}\ (\mathit{let}\ x{=}b\ \mathit{in}\ c)$.

For part (1), assume $\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c : T$. Lemma 10(2) gives $\mathit{let}\ y{=}a\ \mathit{in}\ b : \mathit{Exp}$, $c : \mathit{Exp}$ and $\mathit{dom}(c) = \varnothing$. Applying Lemma 10(2) to $\mathit{let}\ y{=}a\ \mathit{in}\ b : \mathit{Exp}$ gives $a : \mathit{Exp}$, $b : \mathit{Exp}$ and $\mathit{dom}(b) = \varnothing$. From the definition of dom we have $\mathit{dom}(\mathit{let}\ x{=}b\ \mathit{in}\ c) = \mathit{dom}(b) = \varnothing$. Now we can deduce from (Well Let) and (Well Concur) that for either T, $\mathit{let}\ y{=}a\ \mathit{in}\ \mathit{let}\ x{=}b\ \mathit{in}\ c : T$.

For part (2), we assume $\mathit{let}\ y{=}a\ \mathit{in}\ \mathit{let}\ x{=}b\ \mathit{in}\ c : T$. As before, we deduce $a : \mathit{Exp}$, $b : \mathit{Exp}$, $c : \mathit{Exp}$, $\mathit{dom}(b) = \varnothing$ and $\mathit{dom}(c) = \varnothing$. Hence from (Well Let), $\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c : T$.

(Struct Res Let) We have $(\nu p)\mathit{let}\ x{=}a\ \mathit{in}\ b \equiv \mathit{let}\ x{=}(\nu p)a\ \mathit{in}\ b$ obtained from $p \notin \mathit{fn}(b)$. For part (1), we assume $(\nu p)\mathit{let}\ x{=}a\ \mathit{in}\ b : T$. From Lemma 10(2) and (3) we deduce $a : \mathit{Exp}$, $b : \mathit{Exp}$, $\mathit{dom}(b) = \varnothing$ and $p \in \mathit{dom}(a)$. Finally, rules (Well Let), (Well Res) and (Well Concur) give $\mathit{let}\ x{=}(\nu p)a\ \mathit{in}\ b : T$ for either T.

For part (2), we assume $\mathit{let}\ x{=}(\nu p)a\ \mathit{in}\ b : T$. Similarly to before, we deduce $a : \mathit{Exp}$, $b : \mathit{Exp}$, $p \in \mathit{dom}(a)$ and $\mathit{dom}(b) = \varnothing$. Hence we get $(\nu p)\mathit{let}\ x{=}a\ \mathit{in}\ b : T$.

(Struct Par Let) We have $a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c \equiv \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c$. For part (1), we assume $a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c : T$. By Lemma 10(1) and (2) we get $a : \mathit{Proc}$, $b : \mathit{Exp}$, $c : \mathit{Exp}$, $\mathit{dom}(c) = \varnothing$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. Rule (Well Par) implies $a \Rsh b : \mathit{Exp}$, and rule (Well Let) and (Well Concur) gives $\mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c : T$ for either T.

For part (2), we assume $\mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c : T$. Much as before, we deduce $a : \mathit{Proc}$, $b : \mathit{Exp}$, $c : \mathit{Exp}$, $\mathit{dom}(c) = \varnothing$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. From these we deduce $a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c : T$. □

Lemma 12 *If $a : T$ and $a \to b$ then $b : T$.*

Proof We prove this by induction on the derivation of $a \to b$. We consider each of the rules which may derive $a \to b$ in turn:

(Red Select) We have $(p \mapsto d) \Rsh p.\ell_j \to (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\}$ where $d = [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ and $j \in 1..n$. If $(p \mapsto d) \Rsh p.\ell_j : T$ then by Lemma 10(1), $p \mapsto d : \mathit{Proc}$ and $p.\ell_j : T$. Lemma 10(4) tells us $b_i : \mathit{Exp}$ and $\mathit{dom}(b_i) = \varnothing$ for each $i \in 1..n$. Lemma 9(6) applied to $b_j : \mathit{Exp}$ gives $b_j\{\!\{x_j \leftarrow p\}\!\} : \mathit{Exp}$. By (Well Concur), $b_j\{\!\{x_j \leftarrow p\}\!\} : T$ for either T. It is easy to see that $\mathit{dom}(b_j\{\!\{x_j \leftarrow p\}\!\}) = \varnothing$, since $\mathit{dom}(b_j) = \varnothing$. Hence, $(p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\} : T$.

(Red Update), (Red Clone), (Red Acquire), (Red Release) In each case, the proof is similar to that of (Red Select).

(Red Let Result) We have $let\ x{=}p\ in\ b \to b\{\!\{x \leftarrow p\}\!\}$. If $let\ x{=}p\ in\ b : T$ then by Lemma 10(2), $p : Exp$, $b : Exp$ and $dom(b) = \varnothing$. By Lemma 9(6), $b\{\!\{x \leftarrow p\}\!\} : Exp$, and it is easy to see that if $dom(b) = \varnothing$ then $dom(b\{\!\{x \leftarrow p\}\!\}) = \varnothing$.

(Red Res) We have $(\nu p)a \to (\nu p)a'$ obtained from $a \to a'$. If $(\nu p)a : T$ then by Lemma 10(3), $a : T$, and $p \in dom(a)$. By induction, $a' : T$ and Lemma 9(5) implies $dom(a') = dom(a)$, so $p \in dom(a')$. Hence $(\nu p)a' : T$.

(Red Par 1) We have $a \Rsh b \to a' \Rsh b$ from $a \to a'$. If $a \Rsh b : T$ then by Lemma 10(1) we have $a : Proc$, $b : T$ and $dom(a) \cap dom(b) = \varnothing$. Hence by induction, $a' : Proc$, and Lemma 9(5) gives $dom(a) = dom(a')$. Rule (Well Par) gives $a' \Rsh b : T$.

(Red Par 2) Similar to (Red Par 1).

(Red Let) We have $let\ x{=}a\ in\ b \to let\ x{=}a'\ in\ b$ obtained from $a \to a'$. If $let\ x{=}a\ in\ b : T$ then by Lemma 10(2), $a : Exp$, $b : Exp$ and $dom(b) = \varnothing$. By induction, we have $a' : Exp$ and $dom(a) = dom(a')$. Hence by (Well Let), and (Well Concur), $let\ x{=}a'\ in\ b : T$ for either T.

(Red Struct) We have $a \to b$ obtained from $a \equiv a'$, $a' \to b'$ and $b' \equiv b$. If $a : T$ then by Lemma 11, $a' : T$. The induction hypothesis applied to $a' \to b'$ gives $b' : T$. Finally, by Lemma 11 again, $b : T$. □

Proof of Proposition 1

(1) *If $a : T$ and $a \equiv b$ then $b : T$ and $dom(a) = dom(b)$.*

(2) *If $a : T$ and $a \to b$ then $b : T$ and $dom(a) = dom(b)$.*

Proof Combine Lemmas 9, 11, and 12. □

We can generalise this proposition to hold for the T-indexed forms of structural congruence and reduction:

Proposition 13

(1) *If $a : T$ and $a \overset{T}{\equiv} b$ then $b : T$ and $dom(a) = dom(b)$.*

(2) *If $a : T$ and $a \xrightarrow{T} b$ then $b : T$ and $dom(a) = dom(b)$.*

Proof Since Proposition 1 already covers the case $T = \mathit{Exp}$ we need only consider the case for $T = \mathit{Proc}$.

For part (1) we need to show $a : \mathit{Proc}$ and $a \overset{\mathit{Proc}}{\equiv} b$ implies $b : \mathit{Proc}$ and $\mathit{dom}(a) = \mathit{dom}(b)$. Now, $a \overset{\mathit{Proc}}{\equiv} b$ means $a \Rsh p \equiv b \Rsh p$ for some p. If $a : \mathit{Proc}$ then $a \Rsh p : \mathit{Exp}$. Applying Lemma 11 to $a \Rsh p \equiv b \Rsh p$, we get $b \Rsh p : \mathit{Exp}$ and $\mathit{dom}(b) = \mathit{dom}(b \Rsh p) = \mathit{dom}(a \Rsh p) = \mathit{dom}(a)$. Lemma 10(1) applied to $b \Rsh p : \mathit{Exp}$ gives $b : \mathit{Proc}$ as required.

For part (2) we must prove that if $a \overset{\mathit{Proc}}{\to} b$ and $a : \mathit{Proc}$ then $b : \mathit{Proc}$ and $\mathit{dom}(a) = \mathit{dom}(b)$. We recall that $a \overset{\mathit{Proc}}{\to} b$ means $a \overset{\mathit{Proc}}{\equiv} a' \to b' \overset{\mathit{Proc}}{\equiv} b$. Assume $a : \mathit{Proc}$. From part (1), $a' : \mathit{Proc}$ and $\mathit{dom}(a') = \mathit{dom}(a)$. Applying Lemma 12 to $a' \to b'$ gives $b' : \mathit{Proc}$ and $\mathit{dom}(b') = \mathit{dom}(a') = \mathit{dom}(a)$. Finally, by part (1) again, $b : \mathit{Proc}$ and $\mathit{dom}(b) = \mathit{dom}(b') = \mathit{dom}(a)$. □

A.3 Reformulating the Semantics of Section 4.2

When proving Theorem 1 in the next section, it is convenient ot have reformulated the structured operational semantics rules of section 4.2. We factor out of each of the rules the part that extracts a fragment of store and a single thread, and make this a new rule, (SOS′ Config). This makes the rules defining the SOS reductions closer to those specifying the reduction relation. We refer to the rules given here as the SOS′ rules and those of section 4.2 as SOS rules to disambiguate between the two rule sets.

Lemma 14 shows that the two presentations of the structural operational semantics are equivalent.

We define the relation $a \xrightarrow{\mathit{SOS}'} b$ as follows:

Alternative structural operational semantics

(SOS′ Select) (where $\{\vec{p}\} \cap \mathit{fn}(p \mapsto d) = \varnothing$)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i\ {}^{i \in 1..n}] \quad j \in 1..n \quad \mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\} = (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle}{\langle p \mapsto d \parallel p.\ell_j \rangle \xrightarrow{\mathit{SOS}'} (\nu\vec{p})\langle p \mapsto d, \sigma' \parallel \rho' \rangle}$$

(SOS′ Update)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i\ {}^{i \in 1..n}] \quad d' = [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i\ {}^{i \in (1..n)-\{j\}}] \quad j \in 1..n}{\langle p \mapsto d \parallel p.\ell_j \Leftarrow \varsigma(x)b \rangle \xrightarrow{\mathit{SOS}'} \langle p \mapsto d' \parallel p \rangle}$$

(SOS′ Clone) (where $q \notin \mathit{fn}(p \mapsto d)$)

$$\frac{d = [\ell_i = \varsigma(x_i)b_i\ {}^{i \in 1..n}]}{\langle p \mapsto d \parallel \mathit{clone}(p) \rangle \xrightarrow{\mathit{SOS}'} (\nu q)\langle p \mapsto d, q \mapsto d \parallel q \rangle}$$

(SOS′ Acquire)

$$\frac{}{\langle p \mapsto \mathit{unlocked} \parallel \mathit{acquire}(p)\rangle \xrightarrow{SOS'} \langle p \mapsto \mathit{locked} \parallel p\rangle}$$

(SOS′ Release)

$$\frac{d \in \{\mathit{locked}, \mathit{unlocked}\}}{\langle p \mapsto d \parallel \mathit{release}(p)\rangle \xrightarrow{SOS'} \langle p \mapsto \mathit{unlocked} \parallel p\rangle}$$

(SOS′ Let Result)

$$\frac{}{\langle \varnothing \parallel \mathit{let}\ x{=}p\ \mathit{in}\ b\rangle \xrightarrow{SOS'} \mathcal{N}(b)\{\!\{x \leftarrow p\}\!\}}$$

(SOS′ Let) (where $\{\vec{p}\} \cap \mathit{fn}(b) = \varnothing$)

$$\frac{\langle \sigma \parallel t\rangle \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma' \parallel \rho', t'\rangle \quad \mathit{length}(\sigma) \leq 1}{\langle \sigma \parallel \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma' \parallel \rho', \mathit{let}\ x{=}t'\ \mathit{in}\ b\rangle}$$

(SOS′ Config) (where $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_3, \rho_1, \rho_2) = \varnothing$)

$$\frac{\langle \sigma_2 \parallel t\rangle \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle \quad \mathit{length}(\sigma_2) \leq 1}{\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2\rangle \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_2\rangle}$$

(SOS′ Res)

$$\frac{a \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma \parallel \rho\rangle}{(\nu p)a \xrightarrow{SOS'} (\nu p)(\nu\vec{p})\langle \sigma \parallel \rho\rangle}$$

(SOS′ Norm)

$$\frac{\mathcal{N}(a) \xrightarrow{SOS} (\nu\vec{p})\langle \sigma \parallel \rho\rangle}{a \xrightarrow{SOS'} (\nu\vec{p})\langle \sigma \parallel \rho\rangle}$$

Lemma 14 *For all a and b, $a \xrightarrow{SOS} b$ if and only if $a \xrightarrow{SOS'} b$.*

Proof The proof of equivalence will take the following four steps:

(1) If $\langle \sigma \parallel t\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle$ then $\sigma = \sigma_1, \sigma_2, \sigma_3$, $\sigma' = \sigma_1, \sigma_2', \sigma_3$ where $\langle \sigma_2 \parallel t\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma_2 \parallel \rho'\rangle$, $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_3) = \varnothing$ and $\mathit{length}(\sigma_2) \leq 1$.

(2) If $a \xrightarrow{SOS} b$ then $a \xrightarrow{SOS'} b$.

(3) If $\langle \sigma \parallel \rho\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle$ and $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_2, \rho_1, \rho_2) = \varnothing$ then $\langle \sigma_1, \sigma, \sigma_2 \parallel \rho_1, \rho, \rho_2\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_2 \parallel \rho_1, \rho', \rho_2\rangle$.

(4) If $a \xrightarrow{SOS'} b$ then $a \xrightarrow{SOS} b$.

The proofs of (1–4) are as follows:

(1) An easy induction on the derivation of $a \xrightarrow{SOS} b$.

(2) From (1), we note that we can rewrite the derivation of any $a \xrightarrow{SOS} b$ judgment so that any instance of (SOS Let) is of the form: $\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, \mathit{let}\ x{=}t\ \mathit{in}\ b, \rho_2 \rangle \xrightarrow{SOS} (\nu \vec{p})\langle \sigma_1, \sigma_2', \sigma_3 \parallel \rho_1, \rho', \mathit{let}\ x{=}t\,'\mathit{in}\ b, \rho_2 \rangle$ derived from $\langle \sigma_2 \parallel t \rangle \xrightarrow{SOS} (\nu \vec{p})\langle \sigma_2' \parallel \rho', t' \rangle$ where $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_2, \rho_1, b, \rho_2) = \varnothing$ and $\mathit{length}(\sigma_2) \leq 1$.

We can now easily prove (3) by induction on the (rewritten) derivation of $a \xrightarrow{SOS} b$. In all of the non-trivial cases, the SOS rule can be derived from the corresponding SOS′ rule and (SOS′ Config). Hence if $a \xrightarrow{SOS} b$ then $a \xrightarrow{SOS'} b$.

(3) We prove (3) by induction on the derivation of $\langle \sigma \parallel \rho \rangle \xrightarrow{SOS} (\nu \vec{p})\langle \sigma' \parallel \rho' \rangle$. The proof is straightforward in every case.

(4) We need to show that $a \xrightarrow{SOS'} b$ implies $a \xrightarrow{SOS} b$. We prove this by induction on the derivation of $a \xrightarrow{SOS'} b$. All the cases are easy with the exception of (SOS′ Config), which follows from the induction hypothesis and part (3). □

Now we have shown that $\xrightarrow{SOS}$ and $\xrightarrow{SOS'}$ are the same relation we use the notation $a \xrightarrow{SOS} b$ for both in the remainder of this appendix, and all proofs about $a \xrightarrow{SOS} b$ relation depend on the SOS′ rules given in this section.

A.4 Proof of Theorem 1

In this section we prove Theorem 1 and Proposition 4 of Section 4.2 which provide a correspondence between the reduction semantics and the structural operational semantics. We use Theorem 1 in the next section to prove facts from Section 5.

An important result in the proof of Theorem 1 is a lemma, Lemma 22, which states that structural congruence preserves SOS reductions. The proof of Lemma 22 relies on an explicit characterisation of structural congruence on configurations provided by Lemmas 19 and 20. The main result of the section is Proposition 26. Theorem 1 is an immediate corollary.

We first prove some facts about normalisation.

Lemma 15

(1) *If $a : T$ then $\mathcal{N}(a)$ exists.*

(2) *If $a : \mathit{Exp}$ then $\mathcal{N}(a) = (\nu\vec{q})\langle\sigma \parallel \rho, t\rangle$ for some $\vec{q}, \sigma, \rho$ and t.*

Proof These two facts can be proved simultaneously by induction on the derivation of $a : T$. □

Lemma 16 *For all $a : T$, $a \overset{T}{\equiv} \mathcal{N}(a)$.*

Proof There are two cases, when $T = \mathit{Exp}$ and when $T = \mathit{Proc}$. Both cases follow by induction on the structure of a.

When $T = \mathit{Exp}$, the only difficult case is when $a = b \upharpoonright c$. Then $\mathcal{N}(b \upharpoonright c) = (\nu\vec{p})(\nu\vec{q})\langle\sigma, \sigma' \parallel \rho, \rho'\rangle$ where $\mathcal{N}(b) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$, $\mathcal{N}(c) = (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle$, $\{\vec{p}\} \cap (\mathit{fn}(\sigma') \cup \mathit{fn}(\rho')) = \{\vec{q}\} \cap (\mathit{fn}(\sigma) \cup \mathit{fn}(\rho)) = \varnothing$. From Lemma 10(1) we have that $c : \mathit{Exp}$ and $b : \mathit{Proc}$ and hence by Lemma 15, $\rho' \neq \varnothing$. Then we compute:

$$\begin{array}{lll}
 & (\nu\vec{p})(\nu\vec{q})\langle\sigma, \sigma' \parallel \rho, \rho'\rangle & \\
\equiv & (\nu\vec{p})(\nu\vec{q})(\langle\sigma \parallel \rho\rangle \upharpoonright \langle\sigma \parallel \rho'\rangle) & \text{by (Struct Par Comm)} \\
\equiv & (\nu\vec{p})(\langle\sigma \parallel \rho\rangle \upharpoonright (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle) & \text{by (Struct Par 2)} \\
\equiv & ((\nu\vec{p})\langle\sigma \parallel \rho\rangle) \upharpoonright ((\nu\vec{q})\langle\sigma' \parallel \rho'\rangle) & \text{by (Struct Par 1)} \\
= & \mathcal{N}(b) \upharpoonright \mathcal{N}(c) &
\end{array}$$

Hence $\mathcal{N}(b \upharpoonright c) \equiv \mathcal{N}(b) \upharpoonright \mathcal{N}(c)$ and since by induction we have $\mathcal{N}(b) \overset{\mathit{Proc}}{\equiv} b$ and $\mathcal{N}(c) \equiv c$, by Lemma 8, $b \upharpoonright c \equiv \mathcal{N}(b) \upharpoonright \mathcal{N}(c) \equiv \mathcal{N}(b \upharpoonright c)$.

When $T = \mathit{Proc}$, again the only difficult case is when $a = b \upharpoonright c$. Then, we have by Lemma 10(1) that $b : \mathit{Proc}$ and $c : \mathit{Proc}$. We pick a fresh r, and compute much as before:

$$\begin{array}{lll}
 & \mathcal{N}(b \upharpoonright c) \upharpoonright r & \\
= & (\nu\vec{p})(\nu\vec{q})\langle\sigma, \sigma' \parallel \rho, \rho', r\rangle & \\
\equiv & (\nu\vec{p})(\nu\vec{q})(\langle\sigma \parallel \rho\rangle \upharpoonright \langle\sigma' \parallel \rho'\rangle \upharpoonright r) & \text{by (Struct Par Comm)} \\
\equiv & ((\nu\vec{p})(\nu\vec{q})(\langle\sigma \parallel \rho\rangle \upharpoonright \langle\sigma' \parallel \rho'\rangle)) \upharpoonright r & \text{by (Struct Par 1)} \\
\equiv & ((\nu\vec{p})(\langle\sigma \parallel \rho\rangle \upharpoonright (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle)) \upharpoonright r & \text{by (Struct Par 2)} \\
\equiv & ((\nu\vec{p})\langle\sigma \parallel \rho\rangle) \upharpoonright ((\nu\vec{q})\langle\sigma' \parallel \rho'\rangle) \upharpoonright r & \text{by (Struct Par 1)} \\
= & \mathcal{N}(a) \upharpoonright \mathcal{N}(b) \upharpoonright r &
\end{array}$$

Hence by Lemma 8, $b \upharpoonright c \overset{\mathit{Proc}}{\equiv} \mathcal{N}(b) \upharpoonright \mathcal{N}(c) \overset{\mathit{Proc}}{\equiv} \mathcal{N}(b \upharpoonright c)$. □

We can now prove two lemmas stated in section 4.2:

Proof of Lemma 2 *If $a : \mathit{Exp}$ then $\mathcal{N}(a) \equiv a$.*

Proof This is an immediate corollary of Lemma 16. □

The normalisation function is the identity on configurations:

Proof of Lemma 3 $\mathcal{N}((\nu\vec{p})\langle\sigma \parallel \rho\rangle) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$.

Proof By inspection of the normalisation function. □

Lemma 17 *If $t : \mathit{Proc}$ then $t : \mathit{Exp}$. If $a \xrightarrow{SOS} a'$ and $a : \mathit{Proc}$ then $\mathcal{N}(a) : \mathit{Exp}$.*

Proof The first part follows by inspection of the (Well) rules. For the second, we note that every (SOS$'$) reduction involves a thread, so $\mathcal{N}(a)$ has at least one thread. Hence, $\mathcal{N}(a)$ has a right-most thread, say $\mathcal{N}(a) = (\nu\vec{p})(a' \upharpoonright t)$. Since $\mathcal{N}(a) : \mathit{Proc}$ we have $a' : \mathit{Proc}$ and $t : \mathit{Proc}$ by Lemma 10. Applying the first part of this lemma we deduce $t : \mathit{Exp}$, and hence $\mathcal{N}(a) : \mathit{Exp}$ by (Well Par) and (Well Res). □

To state the following lemmas, we extend structural congruence to sequences of terms. Let the relations $a_i{}^{i\in 1..n} \overset{T}{\equiv_s} b_i{}^{i\in 1..n}$ for $T \in \{\mathit{Proc}, \mathit{Exp}\}$ be inductively defined as follows:

Structural Congruence on Sequences

(Seq *Exp*)

$$\frac{a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} a'_i{}^{i\in 1..n} \qquad b \equiv b'}{a_i{}^{i\in 1..n}, b \overset{\mathit{Exp}}{\equiv_s} a'_i{}^{i\in 1..n}, b'}$$

(Seq *Proc* Swap)

$$\frac{}{a_i{}^{i\in 1..n}, b_i{}^{i\in 1..m} \overset{\mathit{Proc}}{\equiv_s} b_i{}^{i\in 1..m}, a_i{}^{i\in 1..n}}$$

(Seq *Proc* Concat)

$$\frac{a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} a'_i{}^{i\in 1..n} \qquad b_i{}^{i\in 1..m} \overset{\mathit{Proc}}{\equiv_s} b'_i{}^{i\in 1..m}}{a_i{}^{i\in 1..n}, b_i{}^{i\in 1..m} \overset{\mathit{Proc}}{\equiv_s} a'_i{}^{i\in 1..n}, b'_i{}^{i\in 1..m}}$$

(Seq *Proc* $\equiv$)

$$\frac{a_i \equiv a'_i \quad \forall i \in 1..n}{a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} a'_i{}^{i\in 1..n}}$$

(Seq *Proc* Trans)

$$\frac{a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} b_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} c_i{}^{i\in 1..n}}{a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} c_i{}^{i\in 1..n}}$$

It is easy to see that both of these relations are symmetric, reflexive and transitive. We also have the following important property of the $\overset{\mathit{Proc}}{\equiv_s}$ relation:

Lemma 18 *If $a_i{}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv_s} b_i{}^{i\in 1..n}$ then for all $j \in 1..n$ there is $k \in 1..n$ such that $a_j \equiv b_k$ and $a_i{}^{i\in 1..n-\{j\}} \overset{\mathit{Proc}}{\equiv_s} b_i{}^{i\in 1..n-\{k\}}$.*

Proof By induction on the derivation of $a_i{}^{i \in 1..n} \overset{Proc}{\equiv}_s b_i{}^{i \in 1..n}$. □

Lemma 19 *If $a \equiv b$, $a : T$ and $b : T$ then $\mathcal{N}(a) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_j{}^{j \in 1..m}\rangle$ and $\mathcal{N}(b) = (\nu\vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_j{}^{j \in 1..m}\rangle$ where:*

(1) *The $\vec{p}$ are pairwise distinct, the $\vec{q}$ are pairwise distinct, and $\{\vec{p}\} = \{\vec{q}\}$.*

(2) $p_i \mapsto d_i{}^{i \in 1..n} \overset{Proc}{\equiv}_s q_i \mapsto d'_i{}^{i \in 1..n}$.

(3) $t_i{}^{i \in 1..m} \overset{T}{\equiv}_s t'_i{}^{i \in 1..m}$.

Conversely, if $a, b : Exp$ and $\mathcal{N}(a), \mathcal{N}(b)$ satisfy properties (1–3) then $a \equiv b$.

Proof We prove this by induction on the derivation of $a \equiv b$. We consider each rule that may derive $a \equiv b$ in turn. In many cases, it is necessary to show that subterms are well-typed; we can use Lemma 10 for this, but we omit the details for clarity.

(Struct Refl) We have $a \equiv a$. The result is trivial.

(Struct Symm) We have $a \equiv b$ derived from $b \equiv a$. The induction hypothesis applied to $b \equiv a$ gives $\mathcal{N}(b) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_j{}^{j \in 1..m}\rangle$ and $\mathcal{N}(a) = (\nu\vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_j{}^{j \in 1..m}\rangle$ satisfying properties (1–3). Since $\overset{Exp}{\equiv}_s$ and $\overset{Proc}{\equiv}_s$ are symmetric, the result follows easily.

(Struct Trans) We have $a \equiv c$ derived from $a \equiv b$ and $b \equiv c$. The result follows from the transitivity of $\overset{Exp}{\equiv}_s$ and $\overset{Proc}{\equiv}_s$.

(Struct Update) We have $u.\ell \Leftarrow \varsigma(x)b \equiv u.\ell \Leftarrow \varsigma(x)b'$ obtained from $b \equiv b'$. Since $\mathcal{N}(u.\ell \Leftarrow \varsigma(x)b) = \langle \varnothing \parallel u.\ell \Leftarrow \varsigma(x)b\rangle$ and $\mathcal{N}(u.\ell \Leftarrow \varsigma(x)b') = \langle \varnothing \parallel u.\ell \Leftarrow \varsigma(x)b'\rangle$ we see that (1) is trivial, (2) is immediate from (Seq *Proc* $\equiv$) and (3) follows from (Struct Update), (Seq *Proc* $\equiv$) and (Seq *Exp*).

(Struct Let) We have *let* x=a *in* $b \equiv$ *let* x=a' *in* b' obtained from $a \equiv a'$ and $b \equiv b'$. Applying the induction hypothesis to $a \equiv a'$ (noting that $a, a' : Exp$) gives $\mathcal{N}(a) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_i{}^{i \in 1..m}\rangle$ and $\mathcal{N}(a') = (\nu\vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_i{}^{i \in 1..m}\rangle$ satisfying properties (1–3). Now:

$$\mathcal{N}(let\ x{=}a\ in\ b) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_i{}^{i \in 1..m-1}, let\ x{=}t_m\ in\ b\rangle$$

Similarly:

$$\mathcal{N}(let\ x{=}a'\ in\ b') = (\nu\vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_i{}^{i \in 1..m-1}, let\ x{=}t'_m\ in\ b\rangle$$

We obtain properties (1) and (2) from the induction hypothesis applied to $a \equiv a'$. For (3), we may deduce from $t_i{}^{i \in 1..m} \overset{\mathit{Exp}}{\equiv}_s t'_i{}^{i \in 1..m}$ that $t_m \equiv t'_m$ and $t_i{}^{i \in 1..m-1} \overset{\mathit{Proc}}{\equiv}_s t'_i{}^{i \in 1..m-1}$. Then, from (Struct Let) $\mathit{let}\ x{=}t_m\ \mathit{in}\ b \equiv \mathit{let}\ x{=}t'_m\ \mathit{in}\ b'$, so rule (Seq *Exp*) gives us property (3).

(Struct Res) Here $(\nu p)a \equiv (\nu p)a'$ obtained from $a \equiv a'$. By induction we have:

$$\begin{array}{rcl} \mathcal{N}(a) & = & (\nu \vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_i{}^{i \in 1..m}\rangle \\ \mathcal{N}(b) & = & (\nu \vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_i{}^{i \in 1..m}\rangle \end{array}$$

satisfying (1–3).

Now, $\mathcal{N}((\nu p)a) = (\nu p)\mathcal{N}(a)$ and $\mathcal{N}((\nu p)b) = (\nu p)\mathcal{N}(b)$. By renaming the bound name p, we may assume $p \notin \{\vec{p}, \vec{q}\}$. So $p, \vec{p}$ and $p, \vec{q}$ satisfy (1), since $\{\vec{p}\} = \{\vec{q}\}$ by induction. Properties (2) and (3) follow from the induction hypothesis.

(Struct Par) Here $a \Rsh b \equiv a' \Rsh b'$ obtained from $a \equiv a'$ and $b \equiv b'$. For simplicity, we consider only the $T = \mathit{Exp}$ case in detail; the $T = \mathit{Proc}$ case is similar. We have $a, a' : \mathit{Proc}$ and $b, b' : \mathit{Exp}$. Applying the induction hypothesis to $a \equiv a'$ and $b \equiv b'$ gives:

$$\begin{array}{rcl} \mathcal{N}(a) & = & (\nu \vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_i{}^{i \in 1..m}\rangle \\ \mathcal{N}(a') & = & (\nu \vec{q})\langle q_i \mapsto d'_i{}^{i \in 1..n} \parallel t'_i{}^{i \in 1..m}\rangle \\ \mathcal{N}(b) & = & (\nu \vec{r})\langle r_i \mapsto d''_i{}^{i \in 1..n'} \parallel t''_i{}^{i \in 1..m'}\rangle \\ \mathcal{N}(b') & = & (\nu \vec{s})\langle s_i \mapsto d'''_i{}^{i \in 1..n'} \parallel t'''_i{}^{i \in 1..m'}\rangle \end{array}$$

satisfying (1–3). By renaming bound variables, we may assume $\{\vec{p}\} \cap \{\vec{r}\} = \varnothing$. From the definition of the normalisation function, we compute $\mathcal{N}(a \Rsh b)$ and $\mathcal{N}(a' \Rsh b')$:

$$\begin{array}{rcl} \mathcal{N}(a \Rsh b) & = & (\nu \vec{p}, \vec{r})\langle p_i \mapsto d_i{}^{i \in 1..n}, r_i \mapsto d''_i{}^{i \in 1..n'} \parallel t_i{}^{i \in 1..m}, t''_i{}^{i \in 1..m'}\rangle \\ \mathcal{N}(a' \Rsh b') & = & (\nu \vec{q}, \vec{s})\langle q_i \mapsto d'_i{}^{i \in 1..n}, s_i \mapsto d'''_i{}^{i \in 1..n'} \parallel t'_i{}^{i \in 1..m}, t'''_i{}^{i \in 1..m'}\rangle \end{array}$$

First we note that (1) holds. By induction, $\{\vec{p}\} = \{\vec{q}\}$ and $\{\vec{r}\} = \{\vec{s}\}$ so $\{\vec{p}, \vec{r}\} = \{\vec{q}, \vec{s}\}$.

We deduce (2) from $p_i \mapsto d_i{}^{i \in 1..n} \overset{\mathit{Proc}}{\equiv}_s q_i \mapsto d'_i{}^{i \in 1..n}$, $r_i \mapsto d''_i{}^{i \in 1..n'} \overset{\mathit{Proc}}{\equiv}_s s_i \mapsto d'''_i{}^{i \in 1..n'}$ and (Seq *Proc* Concat).

For property (3), we note that from $t''_i{}^{i \in 1..m'} \overset{\mathit{Exp}}{\equiv}_s t'''_i{}^{i \in 1..m'}$ we can deduce $t''_i{}^{i \in 1..m'-1} \overset{\mathit{Proc}}{\equiv}_s t'''_i{}^{i \in 1..m'-1}$ and $t''_{m'} \equiv t'''_{m'}$. Then by (Seq *Proc* Concat) and (Seq *Exp*) with $t_i{}^{i \in 1..m} \overset{\mathit{Proc}}{\equiv}_s t'_i{}^{i \in 1..m}$ we can derive $t_i{}^{i \in 1..n}, t''_i{}^{i \in 1..n'} \overset{\mathit{Exp}}{\equiv}_s t'_i{}^{i \in 1..n}, t'''_i{}^{i \in 1..n'}$ as required.

(Struct Object) Here $p \mapsto [\ell_i = \varsigma(x_i)b_i \;^{i \in 1..n}] \equiv p \mapsto [\ell_i = \varsigma(x_i)b'_i \;^{i \in 1..n}]$ obtained from $b_i \equiv b'_i$ for each $i \in 1..n$. We have:

$$\begin{aligned}\mathcal{N}(p \mapsto [\ell_i = \varsigma(x_i)b_i \;^{i \in 1..n}]) &= \langle p \mapsto [\ell_i = \varsigma(x_i)b_i \;^{i \in 1..n} \parallel \varnothing \rangle \\ \mathcal{N}(p \mapsto [\ell_i = \varsigma(x_i)b'_i \;^{i \in 1..n}]) &= \langle p \mapsto [\ell_i = \varsigma(x_i)b'_i \;^{i \in 1..n}] \parallel \varnothing \rangle\end{aligned}$$

We note that we must have $T = \mathit{Proc}$, since neither side of the congruence can have type *Exp*. Properties (1) and (3) are trivial, and (2) can be derived from (Seq *Proc* $\equiv$) and (Struct Object).

(Struct Par Assoc) We have $(a \upharpoonright b) \upharpoonright c \equiv a \upharpoonright (b \upharpoonright c)$. This case is easy, since $\mathcal{N}((a \upharpoonright b) \upharpoonright c) = \mathcal{N}(a \upharpoonright (b \upharpoonright c))$.

(Struct Par Comm) We have $(a \upharpoonright b) \upharpoonright c \equiv (b \upharpoonright a) \upharpoonright c$.

Suppose:

$$\begin{aligned}\mathcal{N}(a) &= (\nu\vec{p})\langle p_i \mapsto d_i \;^{i \in 1..n} \parallel t_i \;^{i \in 1..m}\rangle \\ \mathcal{N}(b) &= (\nu\vec{q})\langle q_i \mapsto d'_i \;^{i \in 1..n'} \parallel t'_i \;^{i \in 1..m'}\rangle \\ \mathcal{N}(c) &= (\nu\vec{r})\langle r_i \mapsto d''_i \;^{i \in 1..n''} \parallel t''_i \;^{i \in 1..m''}\rangle\end{aligned}$$

Then:

$$\begin{aligned}&\mathcal{N}((a \upharpoonright b) \upharpoonright c) = (\nu\vec{p}, \vec{q}, \vec{r})\langle \\ &\quad p_i \mapsto d_i \;^{i \in 1..n}, q_i \mapsto d'_i \;^{i \in 1..n'}, r_i \mapsto d''_i \;^{i \in 1..n''} \parallel \\ &\quad t_i \;^{i \in 1..m}, t'_i \;^{i \in 1..m'}, t''_i \;^{i \in 1..m''}\rangle\end{aligned}$$

Similarly:

$$\begin{aligned}&\mathcal{N}((b \upharpoonright a) \upharpoonright c) = (\nu\vec{q}, \vec{p}, \vec{r})\langle \\ &\quad q_i \mapsto d'_i \;^{i \in 1..n'}, p_i \mapsto d_i \;^{i \in 1..n}, r_i \mapsto d''_i \;^{i \in 1..n''} \parallel \\ &\quad t'_i \;^{i \in 1..m'}, t_i \;^{i \in 1..m}, t''_i \;^{i \in 1..m''}\rangle\end{aligned}$$

By renaming the bound variables, we may assume the names in $\{\vec{p}, \vec{q}, \vec{r}\}$ are pairwise distinct and hence we get property (1). Properties (2) and (3) follow straightforwardly from (Seq *Proc* Concat), (Seq *Proc* Swap) and (Seq *Exp*).

(Struct Par 1) Here $(\nu p)(a \upharpoonright b) \equiv a \upharpoonright (\nu p)b$. The result is immediate, since $\mathcal{N}((\nu p)(a \upharpoonright b)) = \mathcal{N}(a \upharpoonright (\nu p)b)$.

(Struct Par 2) Similar to (Struct Par 1).

(Struct Let Assoc) If $y \notin \mathit{fv}(c)$ we have here $\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c \equiv \mathit{let}\ y{=}a\ \mathit{in}\ (\mathit{let}\ x{=}b\ \mathit{in}\ c)$. Suppose $\mathcal{N}(a) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i \in 1..n} \parallel t_i{}^{i \in 1..m}\rangle$. Then:

$$\begin{aligned}&\mathcal{N}(\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c) = \\ &\qquad (\nu\vec{p})\langle p_i \mapsto d_i \;^{i \in 1..n} \parallel t_i \;^{i \in 1..m-1}, \mathit{let}\ x{=}(\mathit{let}\ y{=}t_m\ \mathit{in}\ b)\ \mathit{in}\ c\rangle\end{aligned}$$

Similarly:

$$\begin{aligned}&\mathcal{N}(\mathit{let}\ y{=}a\ \mathit{in}\ (\mathit{let}\ x{=}b\ \mathit{in}\ c)) = \\ &\qquad (\nu\vec{p})\langle p_i \mapsto d_i\ {}^{i\in 1..n} \parallel t_i\ {}^{i\in 1..m-1}, \mathit{let}\ y{=}t_m\ \mathit{in}\ (\mathit{let}\ x{=}b\ \mathit{in}\ c)\rangle\end{aligned}$$

Properties (1) and (2) are trivial. Property (3) follows much as in the (Struct Let) case.

(Struct Par Let) Here $a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c \equiv \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c$. Again, the result is easy, since $\mathcal{N}(a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c) = \mathcal{N}(\mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c)$.

This completes the proof of the first part of the lemma.

The converse of the lemma, that is, $a \equiv b$ if $a : \mathit{Exp}$ and $\mathcal{N}(a)$ and $\mathcal{N}(b)$ satisfy (1–3) holds by inspection of the (Struct) rules — we may use (Struct Res Res) to permute restrictions, and (Struct Par Comm) to permute the store and threads. □

As a corollary to Lemma 19 we have the following lemma:

Lemma 20 *If* $a : \mathit{Proc}$, $b : \mathit{Proc}$ *and* $a \overset{\mathit{Proc}}{\equiv} b$ *then* $\mathcal{N}(a) = (\nu\vec{p})\langle p_i \mapsto d_i{}^{i\in 1..n} \parallel t_j\ {}^{j\in 1..m}\rangle$ *and* $\mathcal{N}(b) = (\nu\vec{q})\langle q_i \mapsto d'_i\ {}^{i\in 1..n} \parallel t'_j\ {}^{j\in 1..m}\rangle$ *where:*

(1) *The names* $\vec{p}$ *are pairwise distinct, the names* $\vec{q}$ *are pairwise distinct and* $\{\vec{p}\} = \{\vec{q}\}$*.*

(2) $p_i \mapsto d_i\ {}^{i\in 1..n} \overset{\mathit{Proc}}{\equiv}_s q_i \mapsto d'_i\ {}^{i\in 1..n}$.

(3) $t_j\ {}^{j\in 1..m} \overset{\mathit{Proc}}{\equiv}_s t'_j\ {}^{j\in 1..m}$.

Conversely, if $a, b : \mathit{Proc}$ *and* $\mathcal{N}(a), \mathcal{N}(b)$ *satisfy properties (1–3) then* $a \overset{\mathit{Proc}}{\equiv} b$*.*

Proof This follows easily from the previous lemma, because $a \overset{\mathit{Proc}}{\equiv} b$ means that there is a fresh p with $a \Rsh p \equiv b \Rsh p$. If $a, b : \mathit{Proc}$ then $a \Rsh p, b \Rsh p : \mathit{Exp}$. Applying the previous lemma to $a \Rsh p$ and $b \Rsh p$ gives the conditions (1–3) above.

For the converse, we note that if $\mathcal{N}(a)$ and $\mathcal{N}(b)$ satisfy (1–3) then $\mathcal{N}(a \Rsh p)$ and $\mathcal{N}(b \Rsh p)$ (where p is fresh) satisfy (1–3) of the previous lemma. Hence, $a \Rsh p \equiv b \Rsh p$ and by the definition of structural congruence for *Proc* terms, $a \overset{\mathit{Proc}}{\equiv} b$. □

Lemma 21

(1) *If* $a \xrightarrow{SOS} b$ *then* $a \upharpoonright p \xrightarrow{SOS} b'$ *where* $b' \equiv b \upharpoonright p$.

(2) *If* $a \upharpoonright p \xrightarrow{SOS} b'$ *then* $a \xrightarrow{SOS} b$ *where* $b' \equiv b \upharpoonright p$.

Proof

(1) If $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$ then $\mathcal{N}(a \upharpoonright p) = (\nu\vec{p})\langle\sigma \parallel \rho, p\rangle$ where $p \notin \{\vec{p}\}$. If $a \xrightarrow{SOS} b = (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle$ then by (SOS′ Config) we deduce $a \upharpoonright p \xrightarrow{SOS} (\nu\vec{q})\langle\sigma' \parallel \rho', p\rangle \equiv (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle \upharpoonright p$.

(2) Suppose $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$. Then $\mathcal{N}(a \upharpoonright p) = (\nu\vec{p})\langle\sigma \parallel \rho, p\rangle$ where $p \notin \{\vec{p}\}$. By inspection of the (SOS′) rules, we see that the only derivations that can occur are via (SOS′ Res) to remove the restrictions $\vec{p}$, then (SOS′ Config) choosing a thread from ρ (and possibly a denomination from σ), say deriving $(\nu\vec{p})\langle\sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2, p\rangle \xrightarrow{SOS} b'$ (where $b' = (\nu\vec{p})(\nu\vec{q})\langle\sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_2, p\rangle$) from $\langle\sigma_2 \parallel t\rangle \xrightarrow{SOS} (\nu\vec{q})\langle\sigma' \parallel \rho'\rangle$ where $\sigma = \sigma_1, \sigma_2, \sigma_3$ and $\rho = \rho_1, t, \rho_2$. But now we can derive from (SOS′ Config) and (SOS′ Res), $(\nu\vec{p})\langle\sigma \parallel \rho\rangle \xrightarrow{SOS} b$, (where $b = (\nu\vec{p})(\nu\vec{q})\langle\sigma_1, \sigma', \sigma_2 \parallel \rho_1, \rho', \rho_2\rangle$). Finally we note $b \upharpoonright p \equiv b'$. □

Lemma 22 *If* $a : T$, $a \xrightarrow{SOS} b$ *and* $a' \overset{T}{\equiv} a$ *then* $a' \xrightarrow{SOS} b'$ *where* $b' \overset{T}{\equiv} b$.

Proof We first prove the case when $T = \mathit{Exp}$ by induction on the derivation of $a \xrightarrow{SOS} b$. All the cases apart from (SOS′ Config) are similar, so we consider the case (SOS′ Config) and the case (SOS′ Select) as a representative of the other cases.

(SOS′ Select) Here $\langle p \mapsto d \parallel p.\ell_j\rangle \xrightarrow{SOS} (\nu\vec{p})\langle p \mapsto d, \sigma' \parallel \rho'\rangle$ where $d = [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$, $j \in 1..n$, $\mathcal{N}(b_j)\{\!\{p \leftarrow x_j\}\!\} = (\nu\vec{p})\langle\sigma' \parallel \rho'\rangle$ and $\{\vec{p}\} \cap \mathit{fn}(p \mapsto d) = \varnothing$. If $a' \equiv \langle p \mapsto d \parallel p.\ell_j\rangle$ then by Lemma 19 $\mathcal{N}(a') = \langle p \mapsto d' \parallel p.\ell_j\rangle$ where $p \mapsto d \equiv p \mapsto d'$. Hence $d' = [\ell_i = \varsigma(x_i)b'_i\ ^{i \in 1..n}]$ where $b_i \equiv b'_i$ for each $i \in 1..n$. Then by (SOS′ Select), $\langle p \mapsto d' \parallel p.\ell_j\rangle \xrightarrow{SOS} (\nu\vec{q})\langle p \mapsto d', \sigma'' \parallel \rho''\rangle$ where $\mathcal{N}(b'_j)\{\!\{x_j \leftarrow p\}\!\} = (\nu\vec{q})\langle\sigma'' \parallel \rho''\rangle$, and $\{\vec{q}\} \cap \mathit{fn}(p \mapsto d') = \varnothing$. Since $b_j \equiv b'_j$ we have $\mathcal{N}(b_j) \equiv \mathcal{N}(b'_j)$ By Lemma 8(1) we have $\mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\} \equiv \mathcal{N}(b'_j)\{\!\{x_j \leftarrow p\}\!\}$, that is, $(\nu\vec{p})\langle\sigma' \parallel \rho'\rangle \equiv (\nu\vec{p})\langle\sigma'' \parallel \rho''\rangle$. By (Struct Par), $p \mapsto d \upharpoonright (\nu\vec{p})\langle\sigma' \parallel \rho'\rangle \equiv p \mapsto d' \upharpoonright (\nu\vec{q})\langle\sigma'' \parallel \rho''\rangle$. Finally by (Struct Par 1) (applied several times), $(\nu\vec{p})\langle p \mapsto d, \sigma' \parallel \rho'\rangle \equiv (\nu\vec{q})\langle p \mapsto d', \sigma'' \parallel \rho''\rangle$.

(SOS$'$ Config) We have $\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_2 \rangle$ where $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_3, \rho_1, \rho_2) = \varnothing$, $\mathit{length}(\sigma_2) \leq 1$ obtained from $\langle \sigma_2 \parallel t \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$. By Lemma 19 we deduce $\mathcal{N}(b) = \langle \sigma'' \parallel \rho'' \rangle$ where $\sigma'' \overset{\mathit{Proc}}{\equiv_s} \sigma_1, \sigma_2, \sigma_3$ and $\rho'' \overset{\mathit{Exp}}{\equiv_s} \rho_1, t, \rho_2$. Either by Lemma 18 (if $\mathit{length}(\sigma_2) = 1$) or trivially (if $\mathit{length}(\sigma_2) = 0$), we can decompose σ'' as $\sigma''_1, \sigma''_2, \sigma''_3$ where $\sigma''_2 \overset{\mathit{Proc}}{\equiv_s} \sigma_2$ and $\sigma''_1, \sigma''_3 \overset{\mathit{Proc}}{\equiv_s} \sigma_1, \sigma_3$. By Lemma 18 and analysis of the derivation of the judgment $\rho'' \overset{\mathit{Exp}}{\equiv_s} \rho_1, t, \rho_2$ we can decompose ρ'' as ρ''_1, t', ρ''_2 where:

- If $\rho_2 = \varnothing$ then $t' \equiv t$, $\rho''_1 \overset{\mathit{Proc}}{\equiv_s} \rho_1$ and $\rho''_2 = \varnothing$.
- If $\rho_2 \neq \varnothing$ then $t' \equiv t$ and $\rho''_1, \rho''_2 \overset{\mathit{Exp}}{\equiv_s} \rho_1, \rho_2$.

Then $\langle \sigma_2 \parallel t \rangle \equiv \langle \sigma''_2 \parallel t' \rangle$ so by the induction hypothesis applied to $\langle \sigma_2 \parallel t \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$ we have $\langle \sigma''_2 \parallel t' \rangle \xrightarrow{SOS} b'$ where $b' \equiv (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$. By Lemma 19 we have $b' = (\nu\vec{q})\langle \sigma''' \parallel \rho''' \rangle$ where the names $\vec{p}$ are pairwise distinct, the names $\vec{q}$ are pairwise distinct, $\{\vec{p}\} = \{\vec{q}\}$, $\sigma' \overset{\mathit{Proc}}{\equiv_s} \sigma'''$ and $\rho' \overset{\mathit{Exp}}{\equiv_s} \rho'''$. Applying rule (SOS$'$ Config) to the reduction $\langle \sigma''_2 \parallel t' \rangle \xrightarrow{SOS} b'$ we get $\langle \sigma''_1, \sigma''_2, \sigma''_3 \parallel \rho''_1, t', \rho''_2 \rangle \xrightarrow{SOS} (\nu\vec{q})\langle \sigma''_1, \sigma''', \sigma''_3 \parallel \rho''_1, \rho''', \rho''_2 \rangle$ where $\{\vec{q}\} \cap \mathit{fn}(\sigma''_1, \sigma''_3, \rho''_1, \rho''_2) = \varnothing$. We can now deduce $\sigma''_1, \sigma''', \sigma''_3 \overset{\mathit{Proc}}{\equiv_s} \sigma_1, \sigma', \sigma_3$ from (Seq *Proc* Concat). Similarly, we can deduce $\rho''_1, t', \rho''_2, \overset{\mathit{Exp}}{\equiv_s} \rho_1, t, \rho_2$ from (Seq *Exp*), (Seq *Proc* Concat) and the relations between $\rho''_1, \rho_1, \rho''_2$ and ρ_2 above. This suffices for the result by the converse of Lemma 19.

When $T = \mathit{Proc}$ we apply Lemma 21(1) to get that $a \Rsh p \xrightarrow{SOS} a''$ where $a'' \equiv a' \Rsh p$. We apply the $T = \mathit{Exp}$ case to $a \Rsh p \xrightarrow{SOS} a''$ to obtain a b'' with $b \Rsh p \xrightarrow{SOS} b''$ and $b'' \equiv a''$. By Lemma 21(2) we have $b \xrightarrow{SOS} b'$ where $b' \Rsh p \equiv b''$. Now, $b' \Rsh p \equiv b'' \equiv a'' \equiv a' \Rsh p$. Hence $b' \overset{\mathit{Proc}}{\equiv} a'$ as required. □

Lemma 23 *If $(\nu p)a \xrightarrow{SOS} b$ then $b = (\nu p)a'$ with $a \xrightarrow{SOS} a'$.*

Proof We prove this by induction on the derivation of $(\nu p)a \xrightarrow{SOS} b$. The only two rules that can apply are (SOS$'$ Res) and (SOS$'$ Norm). In the former, the result is immediate. In the latter, the result follows from noting that $\mathcal{N}((\nu p)a) = (\nu p)\mathcal{N}(a)$, and then the induction hypothesis applies. □

We now have a lemma which makes explicit the intuition that every reduction involves only a single thread.

Lemma 24 *If* $\langle \sigma \parallel t_1, \ldots, t_n \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$ *then there is* $i \in 1..n$ *with* $\langle \sigma \parallel t_i \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho \rangle$*,* $\rho' = t_1, \ldots, t_{i-1}, \rho, t_{i+1}, \ldots t_n$ *and* $\{\vec{p}\} \cap fn(t_j) = \varnothing$ *for every* $j \in 1..n - \{i\}$*.*

Proof We prove this by induction on the derivation of $\langle \sigma \parallel t_1, \ldots, t_n \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$. For rules (SOS′ Select), (SOS′ Update), (SOS′ Clone), (SOS′ Acquire), (SOS′ Release), (SOS′ Let Result) and (SOS′ Let) the result is trivial, since only one thread reacts in the rule. Rule (SOS′ Res) is not applicable, since the term $\langle \sigma \parallel t_1, \ldots, t_n \rangle$ is not restricted. Rule (SOS′ Norm) is trivial, since $\mathcal{N}(a) = a$ on configurations. In rule (SOS′ Config) we derive $\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_3 \rangle$ from $\langle \sigma_2 \parallel t \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$. The result now follows, since we let $t_1, \ldots, t_{i-1} = \rho_1$, $t_i = t$ and $t_{i+1}, \ldots, t_n = \rho_2$. □

We generalise the previous lemma:

Lemma 25 *If* $\langle \sigma \parallel \rho_1, \rho_2 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho \rangle$ *then either* $\langle \sigma \parallel \rho_1 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$*,* $\rho = \rho', \rho_2$ *and* $\{\vec{p}\} \cap fn(\rho_2) = \varnothing$ *or* $\langle \sigma \parallel \rho_2 \rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho' \rangle$*,* $\rho = \rho_1, \rho'$ *and* $\{\vec{p}\} \cap fn(\rho_1) = \varnothing$*.*

Proof This is an easy corollary of Lemma 24. □

Proposition 26 *For all* $a, b : T$*,* $a \xrightarrow{T} b$ *if and only if* $a \xrightarrow{SOS}\overset{T}{\equiv} b$*.*

Proof To prove $a, b : T$, $a \xrightarrow{T} b$ implies $a \xrightarrow{SOS}\overset{T}{\equiv} b$ we adopt the following proof strategy. We first show that $a : T$ and $a \to b$ implies $a \xrightarrow{SOS}\overset{Exp}{\equiv} b$. After proving this, we deduce $a : Proc$ and $a \xrightarrow{Proc} b$ implies $a \xrightarrow{SOS}\overset{Proc}{\equiv} b$.

We first prove that $a : T$ and $a \to b$ implies $a \xrightarrow{SOS}\equiv b$ by induction on the derivation of $a \to b$. We consider each rule which may derive $a \to b$ in turn:

(Red Select) Here $(p \mapsto d) \Rsh p.\ell_j \to (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\}$ where $d = [\ell_i = \varsigma(x_i)b_i\ ^{i \in 1..n}]$ and $j \in 1..n$. Now, $\mathcal{N}((p \mapsto d) \Rsh p.\ell_j) = \langle p \mapsto d \parallel p.\ell_j \rangle$. From rule (SOS′ Select), $\langle p \mapsto d \parallel p.\ell_j \rangle \xrightarrow{SOS} (\nu\vec{q})\langle p \mapsto d, \sigma \parallel \rho \rangle$ where $(\nu\vec{q})\langle \sigma \parallel \rho \rangle = \mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\}$ and $\{\vec{q}\} \cap fn(p \mapsto d) = \varnothing$. From Lemma 10(4) applied to $(p \mapsto d) \Rsh p.\ell_j : T$ we have $b_j : Exp$. Lemma 16 implies $\mathcal{N}(b_j) \equiv b_j$. Lemma 8(1) implies $\mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\} \equiv b_j\{\!\{x_j \leftarrow p\}\!\}$, that is $(\nu\vec{q})\langle \sigma \parallel \rho \rangle \equiv b_j\{\!\{x_j \leftarrow p\}\!\}$. We can assume $\{\vec{q}\} \cap \{\vec{p}, p\} = \varnothing$ by renaming the bound $\vec{q}$ if necessary. We compute:

$$\begin{array}{lll} & (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\} & \\ \equiv & (p \mapsto d) \Rsh (\nu\vec{q})\langle \sigma\{\!\{x_j \leftarrow p\}\!\} \parallel \rho\{\!\{x_j \leftarrow p\}\!\} \rangle & \text{by (Struct Par)} \\ \equiv & (\nu\vec{q})\langle p \mapsto d, \sigma\{\!\{x_j \leftarrow p\}\!\} \parallel \rho\{\!\{x_j \leftarrow p\}\!\} \rangle & \text{by (Struct Par 2)} \end{array}$$

(Red Update) Here $(p \mapsto d) \upharpoonright p.\ell_j \Leftarrow \varsigma(x)b \to (p \mapsto d') \upharpoonright p$ where $d = [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}]$, $d' = [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}]$ and $j \in 1..n$. Here the result is immediate, since $\mathcal{N}((p \mapsto d) \upharpoonright p.\ell_j \Leftarrow \varsigma(x)b) = \langle p \mapsto d \parallel p.\ell_j \Leftarrow \varsigma(x)b\rangle$ and by rule (SOS′ Update), $\langle p \mapsto d \parallel p.\ell_j \Leftarrow \varsigma(x)b\rangle \xrightarrow{SOS} \langle p \mapsto d' \parallel p\rangle = (p \mapsto d') \upharpoonright p$.

(Red Clone) Here $(p \mapsto d) \upharpoonright \mathit{clone}(p) \to (p \mapsto d) \upharpoonright (\nu q)((q \mapsto d) \upharpoonright q)$ where d is an object, and $q \notin \mathit{fn}(p \mapsto d)$. Now, $\mathcal{N}((p \mapsto d) \upharpoonright \mathit{clone}(p)) = \langle p \mapsto d \parallel \mathit{clone}(p)\rangle$ and by rule (SOS′ Clone), $\langle p \mapsto d \parallel \mathit{clone}(p)\rangle \xrightarrow{SOS} (\nu q)\langle p \mapsto d, q \mapsto d \parallel q\rangle$. By (Struct Par 2), $(\nu q)\langle p \mapsto d, q \mapsto d \parallel q\rangle \equiv (p \mapsto d) \upharpoonright (\nu q)((q \mapsto d) \upharpoonright q)$.

(Red Let Result) Here $\mathit{let}\ x{=}p\ \mathit{in}\ a \to a\{\!\{x \leftarrow p\}\!\}$. From the definition of normalisation, $\mathcal{N}(\mathit{let}\ x{=}p\ \mathit{in}\ a) = \langle \varnothing \parallel \mathit{let}\ x{=}p\ \mathit{in}\ a\rangle$. By rule (SOS′ Let Result), $\langle \varnothing \parallel \mathit{let}\ x{=}p\ \mathit{in}\ a\rangle \xrightarrow{SOS} (\nu\vec{q})\langle \sigma\{\!\{x \leftarrow p\}\!\} \parallel \rho\{\!\{x \leftarrow p\}\!\}\rangle$ where $\mathcal{N}(a) = (\nu\vec{q})\langle \sigma \parallel \rho\rangle$ and $p \notin \{\vec{q}\}$. By Lemma 8(1) $a\{\!\{x \leftarrow p\}\!\} \equiv (\nu\vec{q})\langle \sigma \parallel \rho\rangle\{\!\{x \leftarrow p\}\!\}$ and since $p \notin \{\vec{q}\}$, $(\nu\vec{q})\langle \sigma \parallel \rho\rangle\{\!\{x \leftarrow p\}\!\} = (\nu\vec{q})\langle \sigma\{\!\{x \leftarrow p\}\!\} \parallel \rho\{\!\{x \leftarrow p\}\!\}\rangle$.

(Red Res) Here $(\nu p)a \to (\nu p)b$ obtained from $a \to b$. Lemma 10(3) means $a : T$, so we may apply the induction hypothesis to $a \to b$ to get $a \xrightarrow{SOS} b' \equiv b$. Now by (SOS′ Res), $(\nu p)a \xrightarrow{SOS} (\nu p)b'$ and by rule (Struct Res), $(\nu p)b' \equiv (\nu p)b$.

(Red Par 1) Here $a \upharpoonright b \to a' \upharpoonright b$ obtained from $a \to a'$. Lemma 10(1) applied to $a \upharpoonright b : T$ means $a : \mathit{Proc}$. Hence we may apply the induction hypothesis to $a \to a'$ to get $a \xrightarrow{SOS} a_1 \equiv a'$. Suppose $\mathcal{N}(a) = (\nu\vec{p})\langle \sigma_a \parallel \rho_a\rangle$ and $\mathcal{N}(b) = (\nu\vec{q})\langle \sigma_b \parallel \rho_b\rangle$. Then $\mathcal{N}(a \upharpoonright b) = (\nu\vec{p})(\nu\vec{q})\langle \sigma_a, \sigma_b \parallel \rho_a, \rho_b\rangle$. Since $a \xrightarrow{SOS} a_1$ we know $\mathcal{N}(a) = (\nu\vec{p})\langle \sigma_a \parallel \rho_a\rangle \xrightarrow{SOS} a_1$ Applying Lemma 23 we get $\langle \sigma_a \parallel \rho_a\rangle \xrightarrow{SOS} (\nu\vec{r})\langle \sigma' \parallel \rho'\rangle$ where $a_1 = (\nu\vec{p})(\nu\vec{r})\langle \sigma' \parallel \rho'\rangle$. Hence we can apply rule (SOS′ Config) to get:

$$\langle \sigma_a, \sigma_b \parallel \rho_a, \rho_b\rangle \xrightarrow{SOS} (\nu\vec{r})\langle \sigma', \sigma_b \parallel \rho', \rho_b\rangle$$

Hence by rule (SOS′ Res):

$$(\nu\vec{p})(\nu\vec{q})\langle \sigma_a, \sigma_b \parallel \rho_a, \rho_b\rangle \xrightarrow{SOS} (\nu\vec{p})(\nu\vec{q})(\nu\vec{r})\langle \sigma', \sigma_b \parallel \rho', \rho_b\rangle$$

We now compute:

$$\begin{array}{lll}
 & (\nu\vec{p})(\nu\vec{q})(\nu\vec{r})\langle \sigma', \sigma_b \parallel \rho', \rho_b\rangle & \\
\equiv & (\nu\vec{p},\vec{r})(\nu\vec{q})\langle \sigma', \sigma_b \parallel \rho', \rho_b\rangle & \text{by (Struct Res Res)} \\
\equiv & (\nu\vec{p},\vec{r})\langle \sigma' \parallel \rho'\rangle \upharpoonright (\nu\vec{q})\langle \sigma_b \parallel \rho_b\rangle & \text{by (Struct Par 1 and 2)} \\
\equiv & a' \upharpoonright b & \text{by (Struct Par)}
\end{array}$$

(Red Par 2) Similar to (Red Par 1).

(Red Let) Here $\mathit{let}\ x{=}a\ \mathit{in}\ b \to \mathit{let}\ x{=}a'\ \mathit{in}\ b$ obtained from $a \to a'$. Lemma 10(2) applied to $\mathit{let}\ x{=}a\ \mathit{in}\ b : T$ gives $a : \mathit{Exp}$. Hence we may apply the induction hypothesis to $a \to a'$ to get $a \xrightarrow{SOS} a_1 \equiv a'$. Let $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$. If $a \xrightarrow{SOS} a_1$ then $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle \xrightarrow{SOS} a_1$. We apply Lemma 23 as in the (Red Par 1) case to get $\langle\sigma \parallel \rho\rangle \xrightarrow{SOS} (\nu\vec{r})\langle\sigma' \parallel \rho'\rangle$ where $a_1 = (\nu\vec{p},\vec{r})\langle\sigma' \parallel \rho'\rangle$. Since $a, a' : \mathit{Exp}$ we have by Lemma 15, $\rho = \rho_1, t$ and $\rho' = \rho_1', t'$. We apply Lemma 25 to $\langle\sigma \parallel \rho_1, t\rangle \xrightarrow{SOS} (\nu\vec{r})\langle\sigma' \parallel \rho'\rangle$ to deduce one of:

(1) $\langle\sigma \parallel \rho_1\rangle \xrightarrow{SOS} (\nu\vec{r})\langle\sigma' \parallel \rho_1'\rangle$ and $\rho' = \rho_1', t$

(2) $\langle\sigma \parallel t\rangle \xrightarrow{SOS} (\nu\vec{r})\langle\sigma' \parallel \rho_2', t'\rangle$ and $\rho' = \rho_1, \rho_2', t'$.

In case 1 we compute:

$$\begin{array}{lll}
 & \mathcal{N}(\mathit{let}\ x{=}a\ \mathit{in}\ b) & \\
= & (\nu\vec{p})\langle\sigma \parallel \rho_1, \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle & \\
\xrightarrow{SOS} & (\nu\vec{p})(\nu\vec{r})\langle\sigma' \parallel \rho_1', \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle & \text{by (SOS}'\text{ Config)} \\
\equiv & (\nu\vec{p},\vec{r})\mathit{let}\ x{=}\langle\sigma' \parallel \rho_1', t\rangle\ \mathit{in}\ b & \text{by (Struct Par Let)} \\
\equiv & \mathit{let}\ x{=}(\nu\vec{p},\vec{r})\langle\sigma' \parallel \rho_1', t\rangle\ \mathit{in}\ b & \text{by (Struct Res Let)} \\
= & \mathit{let}\ x{=}a_1\ \mathit{in}\ b & \\
\equiv & \mathit{let}\ x{=}a'\ \mathit{in}\ b & \text{by (Struct Let)}
\end{array}$$

In case 2 we compute:

$$\begin{array}{lll}
 & \mathcal{N}(\mathit{let}\ x{=}a\ \mathit{in}\ b) & \\
= & (\nu\vec{p})\langle\sigma \parallel \rho_1, \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle & \\
\xrightarrow{SOS} & (\nu\vec{p})(\nu\vec{r})\langle\sigma' \parallel \rho_1, \rho_2', \mathit{let}\ x{=}t'\ \mathit{in}\ b\rangle & \text{by (SOS}'\text{ Let)} \\
\equiv & (\nu\vec{p},\vec{r})\mathit{let}\ x{=}\langle\sigma' \parallel \rho'\rangle\ \mathit{in}\ b & \text{by (Struct Par Let)} \\
\equiv & \mathit{let}\ x{=}(\nu\vec{p},\vec{r})\langle\sigma' \parallel \rho'\rangle\ \mathit{in}\ b & \text{by (Struct Res Let)} \\
= & \mathit{let}\ x{=}a_1\ \mathit{in}\ b & \\
\equiv & \mathit{let}\ x{=}a'\ \mathit{in}\ b & \text{by (Struct Let)}
\end{array}$$

(Red Struct) Here $a \to b$ obtained from $a \equiv a'$, $a' \to b'$ and $b' \equiv b$. If $a : T$ then by Proposition 13, $a' : T$, $b' : T$ and $b : T$. Hence we may apply the induction hypothesis to $a' \to b'$ to get $a' \xrightarrow{SOS}\equiv b'$. We apply Lemma 22 to get that $a \xrightarrow{SOS} b''$ where $b'' \equiv b'$. Hence, $a \xrightarrow{SOS} b'' \equiv b' \equiv b$ and by (Struct Trans), $a \xrightarrow{SOS} b'' \equiv b$.

(Red Acquire) Here $(p \mapsto \mathit{unlocked}) \Rsh \mathit{acquire}(p) \to (p \mapsto \mathit{locked}) \Rsh p$. The result follows immediately in this case from the two observations:

$$\mathcal{N}((p \mapsto \mathit{unlocked}) \Rsh \mathit{acquire}(p)) = \langle p \mapsto \mathit{unlocked} \parallel \mathit{acquire}(p)\rangle$$
$$\langle p \mapsto \mathit{unlocked} \parallel \mathit{acquire}(p)\rangle \xrightarrow{SOS} \langle p \mapsto \mathit{locked} \parallel p\rangle = (p \mapsto \mathit{unlocked}) \Rsh p$$

(Red Release) Similar to (Red Acquire).

This completes the proof of $a : T$ and $a \to b$ implies $a \overset{SOS}{\longrightarrow}\equiv b$. It remains to show that if $a : \mathit{Proc}$ and $a \overset{Proc}{\to} b$ then $a \xrightarrow{SOS} b' \overset{Proc}{\equiv} b$. If $a \overset{Proc}{\to} b$ then $a \overset{Proc}{\equiv} a_1 \to b_1 \overset{Proc}{\equiv} b$. By the result just proven, $a_1 \to b_1$ means $a_1 \overset{SOS}{\longrightarrow}\equiv b_1$. Lemma 22 gives $a \xrightarrow{SOS} b_1' \overset{Proc}{\equiv} b_1'' \equiv b_1$. So $a \xrightarrow{SOS} b_1' \overset{Proc}{\equiv} b_1'' \equiv b_1 \overset{Proc}{\equiv} b$. Since $\equiv \subset \overset{Proc}{\equiv}$ and $\overset{Proc}{\equiv}$ is transitive, $a \xrightarrow{SOS} \overset{Proc}{\equiv} b$.

This completes the proof of forwards implication, that $a : T$ and $a \xrightarrow{T} b$ implies $a \overset{SOS\,T}{\longrightarrow\equiv} b$.

For the other half of this theorem, we prove that $a : T$ and $a \xrightarrow{SOS} b$ implies $a \overset{T}{\to} b$ by induction on the derivation of $a \xrightarrow{SOS} b$. The rules (SOS′ Update), (SOS′ Acquire) and (SOS′ Release) are special cases of the respective (Red) rules, so we consider the other (SOS′) rules which may derive $a \xrightarrow{SOS} b$ in turn. We note that in cases (SOS′ Select), (SOS′ Clone) and (SOS′ Let Result) we show that $a \to b$; this is sufficient since $\to \subset \overset{Proc}{\to}$.

(SOS′ Select) Here $\langle p \mapsto d \parallel p.\ell_j\rangle \xrightarrow{SOS} (\nu\vec{p})\langle p \mapsto d, \sigma \parallel \rho\rangle$ where $d = [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}]$, $j \in 1..n$, $\mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\} = \langle\sigma \parallel \rho\rangle$ and $\{\vec{q}\} \cap \mathit{fn}(p \mapsto d) = \varnothing$. By (Red Select), $\langle p \mapsto d \parallel p.\ell_j\rangle \to (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\}$.

Lemma 10(1) and (4) give $b_j : \mathit{Exp}$, so $\mathcal{N}(b_j) \equiv b_j$. Lemma 8(1) gives $\mathcal{N}(b_j)\{\!\{x_j \leftarrow p\}\!\} \equiv b_j\{\!\{x_j \leftarrow p\}\!\}$. Hence $(p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\} \equiv (\nu\vec{p})\langle p \mapsto d, \sigma \parallel \rho\rangle$ by (Struct Par 2) and (Struct Par).

(SOS′ Clone) Here $\langle p \mapsto d \parallel \mathit{clone}(p)\rangle \xrightarrow{SOS} (\nu q)\langle p \mapsto d, q \mapsto d \parallel q\rangle$ where d is an object and $q \notin \mathit{fn}(p \mapsto d)$. Now, $(p \mapsto d) \Rsh \mathit{clone}(p) \to (p \mapsto d) \Rsh (\nu q)(q \mapsto d \Rsh q) \equiv (\nu q)((p \mapsto d) \Rsh (q \mapsto d) \Rsh q)$ as required.

(SOS′ Let Result) Here $\langle\varnothing \parallel \mathit{let}\ x{=}p\ \mathit{in}\ b\rangle \xrightarrow{SOS} (\nu\vec{p})\langle\sigma\{\!\{x \leftarrow p\}\!\} \parallel \rho\{\!\{x \leftarrow p\}\!\}\rangle$ where $\mathcal{N}(b) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$ and $p \notin \{\vec{p}\}$. Now by (Red Let Result), $\langle\varnothing \parallel \mathit{let}\ x{=}p\ \mathit{in}\ b\rangle \to b\{\!\{x \leftarrow p\}\!\}$. Since $\mathit{let}\ x{=}p\ \mathit{in}\ b : T$ we have by Lemma 10 $b : \mathit{Exp}$. Hence, by Lemma 16, $b \equiv \mathcal{N}(b) = (\nu\vec{p})\langle\sigma \parallel \rho\rangle$. By Lemma 8(1), $b\{\!\{x \leftarrow p\}\!\} \equiv (\nu\vec{p})\langle\sigma\{\!\{x \leftarrow p\}\!\} \parallel \rho\{\!\{x \leftarrow p\}\!\}\rangle$ (since $p \notin \{\vec{p}\}$).

(SOS′ Let) Here $\langle \varnothing \parallel \mathit{let}\ x{=}t\ \mathit{in}\ b\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho', \mathit{let}\ x{=}t'\ \mathit{in}\ b\rangle$ obtained from $\langle \sigma \parallel t\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho', t'\rangle$ where $\{vecp\} \cap \mathit{fn}(b) = \varnothing$. Since $\mathit{let}\ x{=}t\ \mathit{in}\ b : T$ we have from Lemma 10(2) that $t : \mathit{Exp}$. Hence by induction, $t \rightarrow (\nu\vec{p})\langle \sigma' \parallel \rho', t'\rangle$. By rule (Red Let), $\mathit{let}\ x{=}t\ \mathit{in}\ b \rightarrow \mathit{let}\ x{=}(\nu\vec{p})\langle \sigma' \parallel \rho', t'\rangle\ \mathit{in}\ b$ and by (Struct Par Let) and (Struct Res Let), $\mathit{let}\ x{=}(\nu\vec{p})\langle \sigma' \parallel \rho', t'\rangle\ \mathit{in}\ b \equiv (\nu\vec{p})\langle \sigma' \parallel \rho', \mathit{let}\ x{=}t'\ \mathit{in}\ b\rangle$.

(SOS′ Config) Here $\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_2\rangle$ obtained from $\langle \sigma_2 \parallel t\rangle \xrightarrow{SOS} (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle$ where $\{\vec{p}\} \cap \mathit{fn}(\sigma_1, \sigma_3, \rho_1, \rho_2) = \varnothing$. By induction we have $\langle \sigma_2 \parallel t\rangle \xrightarrow{T} b'$ where $b' = (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle$. Applying rules (Red Par Comm), (Red Par 1) and (Red Par 2) we can deduce $\langle \sigma_1, \sigma_2, \sigma_3 \parallel \rho_1, t, \rho_2\rangle \xrightarrow{T} \sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright b' \upharpoonright \rho_2$. Rule (Struct Par) gives $\sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright b' \upharpoonright \rho_2 \equiv \sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle \upharpoonright \rho_2$. Rules (Struct Par 1) and (Struct Par 2) give $\sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright (\nu\vec{p})\langle \sigma' \parallel \rho'\rangle \upharpoonright \rho_2 \equiv (\nu\vec{p})(\sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright \sigma' \upharpoonright \rho' \upharpoonright \rho_2)$. Finally, (Struct Par Comm) gives $(\nu\vec{p})(\sigma_1 \upharpoonright \sigma_3 \upharpoonright \rho_1 \upharpoonright \sigma' \upharpoonright \rho' \upharpoonright \rho_2) \equiv (\nu\vec{p})\langle \sigma_1, \sigma', \sigma_3 \parallel \rho_1, \rho', \rho_2\rangle$. Since $\equiv \subseteq \overset{T}{\equiv}$ for $T = \mathit{Exp}$ or Proc and $\overset{T}{\equiv}$ is transitive, we are done.

(SOS′ Res) Here $(\nu p)a \xrightarrow{SOS} (\nu p)b$ obtained from $a \xrightarrow{SOS} b$. Since $(\nu p)a : T$, by Lemma 10(3), $a : T$ so by induction, $a \xrightarrow{T} b$. If $T = \mathit{Exp}$, then by (Red Res), $(\nu p)a \rightarrow (\nu p)b$. Otherwise, $a \overset{Proc}{\equiv} a' \rightarrow b' \overset{Proc}{\equiv} b$. By (Red Res), $(\nu p)a' \rightarrow (\nu p)b'$. Lemma 8 implies $(\nu p)a \overset{Proc}{\equiv} (\nu p)a'$ and $(\nu p)b' \overset{Proc}{\equiv} (\nu p)b'$. Hence, $(\nu p)a \overset{Proc}{\rightarrow} (\nu p)b$ as required.

(SOS′ Norm) Here $a \xrightarrow{SOS} b$ obtained from $\mathcal{N}(a) \xrightarrow{SOS} b$. Since $a : T$ we have $a \overset{T}{\equiv} \mathcal{N}(a)$. The induction hypothesis applied to $\mathcal{N}(a) \xrightarrow{SOS} b$ gives $\mathcal{N}(a) \xrightarrow{T} b$. Since $a \overset{T}{\equiv} \mathcal{N}(a) \xrightarrow{T} b$ we have $a \xrightarrow{T} b$ as required. □

Proof of Theorem 1 *For all $a, b : \mathit{Exp}$, $a \rightarrow b$ if and only $a \xrightarrow{SOS}\equiv b$.*

Proof An immediate corollary of Proposition 26. □

A.5 Proof of Theorem 2

In this section we prove some facts about the single-threaded fragment of the language defined in Section 5. In particular, we prove Lemma 5, Theorem 2, and Proposition 6.

We begin with a lemma similar to Lemma 10 for the deterministic type system, $a :^1 A$:

Lemma 27

(1) *If* $a \Rsh b :^1 T$ *then* $a :^1 \mathit{Proc}$, $b :^1 T$ *and* $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$.

(2) *If* $\mathit{let}\ x{=}a\ \mathit{in}\ b :^1 T$ *then* $T = \mathit{Exp}$, $a : \mathit{Exp}$, $b : \mathit{Exp}$ *and* $\mathit{dom}(b) = \varnothing$.

(3) *If* $(\nu p)a :^1 T$ *then* $a :^1 T$ *and* $p \in \mathit{dom}(a)$.

(4) *If* $p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}] : T$ *then* $T = \mathit{Proc}$, $b_i : \mathit{Exp}$ *and* $\mathit{dom}(b_i) = \varnothing$ *for all* $i \in 1..n$.

(5) *If* $u.\ell \Leftarrow \varsigma(x)b :^1 T$ *then* $T = \mathit{Exp}$, $b : \mathit{Exp}$ *and* $\mathit{dom}(b) = \varnothing$.

Proof Each of these statements follows by an easy induction on the derivation of the typing derivation. □

Lemma 28 *If* $t :^1 T$ *then* $T = \mathit{Exp}$.

Proof By inspection of the typing rules. □

Proof of Lemma 5

(1) *If* $a :^1 T$ *and* $a \equiv b$ *then* $b :^1 T$.

(2) *If* $a :^1 T$ *and* $a \to b$ *then* $b :^1 T$.

(3) *For all* $a, b :^1 \mathit{Exp}$, $a \to b$ *if and only* $a \xrightarrow{SOS}\equiv b$.

(4) *If* $a :^1 \mathit{Proc}$ *then* $\mathcal{N}(a)$ *takes the form* $(\nu\vec{p})\langle\sigma \parallel \varnothing\rangle$.

(5) *If* $a :^1 \mathit{Exp}$ *then* $\mathcal{N}(a)$ *takes the form* $(\nu\vec{p})\langle\sigma \parallel t\rangle$.

Proof

(1), (2) These may be proved by similar inductions to those found in the proof of Proposition 1.

(3) Since $a :^1 \mathit{Exp}$ implies $a : \mathit{Exp}$, we have from Theorem 1 that $a \to b$ if and only if $a \xrightarrow{SOS}\equiv b$.

(4) Suppose $a :^1 \mathit{Proc}$. Therefore we may derive $a : \mathit{Proc}$. By Lemma 15, $\mathcal{N}(a)$ exists. Suppose that $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel t_1, \ldots, t_n\rangle$ for names $\vec{p}$, store σ, and threads t_1, …, t_n. By Lemma 16, $a \equiv (\nu\vec{p})\langle\sigma \parallel t_1, \ldots, t_n\rangle$. By part (1), $a :^1 \mathit{Proc}$ implies $(\nu\vec{p})\langle\sigma \parallel t_1, \ldots, t_n\rangle :^1 \mathit{Proc}$. By Lemma 27, $t_1, \ldots, t_n :^1 \mathit{Proc}$. By Lemma 27, it must be that $n = 0$.

(5) Suppose $a :^1 \mathit{Exp}$. Much as in the previous case, we have that $\mathcal{N}(a) = (\nu\vec{p})\langle\sigma \parallel t_1, \ldots, t_n\rangle$, and that $(\nu\vec{p})\langle\sigma \parallel t_1, \ldots, t_n\rangle :^1 \mathit{Exp}$. By Lemma 27, it must be that $n = 1$. □

Proof of Theorem 2 *Suppose $a :^1 \mathit{Exp}$. If $a \to a'$ and $a \to a''$ then $a' \equiv a''$.*

Proof By applying Lemma 5, we can normalise a to $\mathcal{N}(a)$ and consider reductions in the SOS′ semantics rather than the reduction semantics. It is a simple induction to prove that SOS′ reductions are unique for terms in the single-threaded fragment, because, by Lemma 5, single-threaded terms have only one thread in their configuration. This fact suffices to prove this theorem, because if $a \to a'$ and $a \to a''$ then $a \xrightarrow{SOS} b \equiv a'$ and $a \xrightarrow{SOS} b' \equiv a''$. But by the fact that SOS′ reductions are unique, $b = b'$ and $a'' \equiv a'$. □

Proof of Proposition 6 *If a represents a term of* **impς**, *we can derive $a :^1 \mathit{Exp}$.*

Proof This can be proved by induction on the structure of a. □

A.6 Proof of Theorem 3

In this section we prove the subject reduction result of Section 5, Theorem 3.

Lemma 29

(1) *If $E \vdash A$ then $E \vdash \diamond$.*

(2) *If $E \vdash a : A$ then $E \vdash A$.*

(3) *If $E \vdash A{<:}B$ then $E \vdash A$ and $E \vdash B$.*

Proof By inductions on derivations. □

Proof of Lemma 7 *If $E \vdash a : A$ and $E \vdash A{<:}T$ then $a : T$.*

Proof By induction on the derivation of $E \vdash a : A$.

(Val Subsumption) Here $E \vdash a : B$ obtains from $E \vdash a : A$ and $E \vdash A{<:}B$. If $E \vdash B{<:}T$ then $E \vdash A{<:}T$ by (Sub Trans). Hence by induction, $a : T$.

(Val Object) Here $E \vdash p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i\in 1..n}] : \mathit{Proc}$ obtains from $E, x_i : A \vdash b_i : B_i$ with $E = E_1, p : A, E_2$ and $\mathit{dom}(b_i) = \varnothing$ for $i \in 1..n$, where $A = [\ell_i : B_i{}^{\,i\in 1..n}]$. By Lemma 29, $E \vdash \diamond$. By (Val u) $E \vdash p : A$, and hence $E \vdash A$. From (Type Object) we get $E \vdash B_i{<:}\mathit{Exp}$ for all $i \in 1..n$. Hence by induction, $b_i : \mathit{Exp}$ and from (Well Object), $p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i\in 1..n}] : \mathit{Proc}$ as required.

(Val Let) Here $E \vdash \mathit{let}\ x{=}a\ \mathit{in}\ b : B$ obtains from $E \vdash a : A$ and $E, x : A \vdash b : B$, $\mathit{dom}(b) = \varnothing$, $E \vdash A{<:}\mathit{Exp}$ and $E \vdash B{<:}\mathit{Exp}$. By induction $a : \mathit{Exp}$ and $b : \mathit{Exp}$ and hence by (Well Let), $\mathit{let}\ x{=}a\ \mathit{in}\ b : \mathit{Exp}$.

The cases for the other rules are similar. □

Lemma 30

(1) *If $E \vdash a \upharpoonright b : A$ then $E \vdash a : \mathit{Proc}$, $E \vdash b : A$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$.*

(2) *If $E \vdash \mathit{let}\ x{=}a\ \mathit{in}\ b : B$ then there are A, B' with $E \vdash A{<:}\mathit{Exp}$, $E \vdash B'{<:}\mathit{Exp}$ such that $E \vdash a : A$, $E, x : A \vdash b : B'$, $\mathit{dom}(b) = \varnothing$ and $E \vdash B'{<:}B$.*

(3) *If $E \vdash (\nu p)b : B$ then there is an A such that $E, p : A \vdash b : B$ and $p \in \mathit{dom}(b)$.*

(4) *If $E \vdash u : A$ then $E = E_1, u : A', E_2$ where $E \vdash A'{<:}A$.*

(5) *If $E \vdash u.\ell_j \Leftarrow \varsigma(x)b : A$ then there is an $A' = [\ell_i : B_i{}^{\,i\in 1..n}]$ such that $E \vdash u : A'$, $j \in 1..n$, $\mathit{dom}(b) = \varnothing$, $E, x : A' \vdash b : B_j$ and $E \vdash A'{<:}A$.*

(6) *If $E \vdash p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{\,i\in 1..n}] : A$ then $A = \mathit{Proc}$ and there is an $A' = [\ell_i : B_i{}^{\,i\in 1...n}]$ such that $E = E_1, p : A', E_2$, $\mathit{dom}(b_i) = \varnothing$ and $E, x_i : A' \vdash b_i : B_i$ for all $i \in 1..n$.*

(7) *If $E \vdash \mathit{acquire}(u) : A$ then $E \vdash u : \mathit{Mutex}$.*

(8) *If $E \vdash \mathit{release}(u) : A$ then $E \vdash u : \mathit{Mutex}$.*

(9) *If $E \vdash \mathit{clone}(u) : A$ then $E \vdash u : A$.*

(10) *If $E \vdash u.\ell_j : A$ then there is an $A' = [\ell_i : B_i{}^{\,i\in 1..n}]$ such that $E \vdash u : A'$, $j \in 1..n$ and $E \vdash B_j{<:}A$.*

Proof Each of these facts can be proved by an induction on the appropriate judgment. □

We use the notation $E \vdash \mathcal{J}$ for an arbitrary judgment, where $\mathcal{J}$ stands for the fragments $\diamond$, A, $a : A$ and $A{<:}B$.

We need the following standard lemmas in the proof of subject reduction:

Lemma 31 *If $E, x : B, E' \vdash \mathcal{J}$ and $E \vdash v : B$ then $E, E' \vdash \mathcal{J}\{\!\{x \leftarrow v\}\!\}$.*

Proof By induction on the derivation of $E, x : B, E' \vdash \mathcal{J}$. □

Lemma 32 *If $E, x : D, E' \vdash \mathcal{J}$ and $E \vdash D'{<:}D$ then $E, x : D', E' \vdash \mathcal{J}$.*

Proof This is by an induction on the derivation of $E, x : D, E' \vdash \mathcal{J}$. □

We note that this lemma is not valid if we generalise it to allow bound weakening for names as well as variables in the environment. For example, $\varnothing, p : [] \vdash (p \mapsto []) : \mathit{Proc}$ holds, but the weakened judgment $\varnothing, p : [\ell : []] \vdash (p \mapsto []) : \mathit{Proc}$ does not.

Lemma 33 *If $E, u : A, E' \vdash \mathcal{J}$ and $u \notin \mathit{fn}(\mathcal{J}) \cup \mathit{fv}(\mathcal{J})$ then $E, E' \vdash \mathcal{J}$.*

Proof By induction on the derivation of $E, u : A, E' \vdash \mathcal{J}$. □

Lemma 34 *If $E, E' \vdash \mathcal{J}$ and $u \notin \mathit{dom}(E, E')$ and $E \vdash A$ then $E, u : A, E' \vdash \mathcal{J}$.*

Proof By induction on the derivation of $E, E' \vdash \mathcal{J}$. □

Lemma 35 *If $E, u : A, v : B, E' \vdash a : C$ then $E, v : B, u : A, E' \vdash a : C$.*

Proof By induction on the derivation of $E, u : A, v : B, E' \vdash \mathcal{J}$. □

We show that structural congruence preserves typings:

Lemma 36 *If $E \vdash a : A$ and $a \equiv b$ then $E \vdash b : A$.*

Proof We first symmetrise the lemma:

(1) If $E \vdash a : A$ and $a \equiv b$ then $E \vdash b : A$.

(2) If $E \vdash b : A$ and $a \equiv b$ then $E \vdash a : A$.

We prove this by induction on the derivation of $a \equiv b$. We consider each of the rules which may derive $a \equiv b$ in turn:

(Struct Refl) We have $a \equiv a$, and the result is trivial.

(Struct Symm) We have $a \equiv b$ obtained from $b \equiv a$. Because of the symmetrised form of the lemma, the result is trivial.

(Struct Trans) We have $a \equiv c$ obtained from $a \equiv b$ and $b \equiv c$. If $E \vdash a : A$ then the induction hypothesis applied to $a \equiv b$ gives $E \vdash b : A$. The induction hypothesis applied to $b \equiv c$ gives $E \vdash c : A$. Conversely, if $E \vdash c : A$ then we deduce $E \vdash b : A$ $b \equiv c$. Similarly, we deduce $E \vdash a : A$ from $a \equiv b$.

(Struct Update) We have $u.\ell_j \Leftarrow \varsigma(x)b \equiv u.\ell_j \Leftarrow \varsigma(x)b'$ obtained from $b \equiv b'$. If $E \vdash u.\ell_j \Leftarrow \varsigma(x)b : A$ then by Lemma 30(5) we have an $A' = [\ell_i : B_i{}^{i \in 1..n}]$ with $j \in 1..n$, $E, x : A' \vdash b : B_j$, $E \vdash u : A'$, $dom(b) = \varnothing$ and $E \vdash A' <: A$. The induction hypothesis applied to $b \equiv b'$ gives $E, x : A' \vdash b' : B_j$. By Lemma 9(4), $dom(b') = \varnothing$. Hence by (Val Update) $E \vdash u.\ell_j \Leftarrow \varsigma(x)b' : A'$. Since $E \vdash A' <: A$ we have by (Val Subsumption) $E \vdash u.\ell_j \Leftarrow \varsigma(x)b' : A$ as required. Part (2) follows by symmetry.

(Struct Let) We have $let\ x{=}a\ in\ b \equiv let\ x{=}a'\ in\ b'$ obtained from $a \equiv a'$ and $b \equiv b'$. If $E \vdash let\ x{=}a\ in\ b : B$ then by Lemma 30(2) we have A, B' with $E \vdash A <: Exp$ and $E \vdash B' <: Exp$ such that $E \vdash a : A$, $E, x : A \vdash b : B'$, $dom(b) = \varnothing$ and $E \vdash B' <: B$. The induction hypothesis applied to $a \equiv a'$ gives $E \vdash a' : A$. The induction hypothesis applied to $b \equiv b'$ gives $E, x : A \vdash b' : B'$. Hence by (Val Let) and (Val Subsumption), $E \vdash let\ x{=}a'\ in\ b' : B$. Part (2) follows by symmetry.

(Struct Res) We have $(\nu p)a \equiv (\nu p)a'$ obtained from $a \equiv a'$. If $E \vdash (\nu p)a : A$ then by Lemma 30(3), $E, p : B \vdash a : A$ for some B, and $p \in dom(a)$. The induction hypothesis applied to $a \equiv a'$ gives $E, p : B \vdash a' : A$. Lemma 9(4) gives $dom(a') = dom(a)$, and so $p \in dom(a')$. Hence, by (Val Res), $E \vdash (\nu p)a' : A$. Part (2) follows by symmetry.

(Struct Par) We have $a \Rsh b \equiv a' \Rsh b'$ obtained from $a \equiv a'$ and $b \equiv b'$. If $E \vdash a \Rsh b : A$ then by Lemma 30(1) we have $E \vdash a : Proc$, $E \vdash b : A$ and $dom(a) \cap dom(b) = \varnothing$. The induction hypothesis applied to $a \equiv a'$ gives $E \vdash a' : Proc$. The induction hypothesis applied to $b \equiv b'$ gives $E \vdash b' : A$. By Lemma 9(4), $dom(a') = dom(a)$ and $dom(b') = dom(b)$. Hence by (Well Par), $E \vdash a' \Rsh b' : A$. Part (2) follows by symmetry.

(Struct Object) We have $p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}] \equiv p \mapsto [\ell_i = \varsigma(x_i)b'_i{}^{i \in 1..n}]$ obtained from $b_i \equiv b'_i$ and $dom(b_i) = \varnothing$ for all $i \in 1..n$. If $E \vdash p \mapsto [\ell_i = \varsigma(x_i)b_i{}^{i \in 1..n}] : A$ then by Lemma 30(6) we have $A = Proc$ and there is an $A' = [\ell_i : B_i{}^{i \in 1..n}]$ such that $E = E_1, p : A', E_2$,

$dom(b_i) = \varnothing$ and $E, x_i : A' \vdash b_i : B_i$ for each $i \in 1..n$. The induction hypothesis applied to $b_i \equiv b'_i$ gives $E, x_i : A' \vdash b'_i : B_i$ for $i \in 1..n$. Lemma 9(4) gives $dom(b_i) = \varnothing$ for each $i \in 1..n$. Hence by (Val Object), $E \vdash p \mapsto [\ell_i = \varsigma(x_i)b'_i{}^{i \in 1..n}] : Proc$ as required.

(Struct Par Assoc) We have $(a \upharpoonright b) \upharpoonright c \equiv a \upharpoonright (b \upharpoonright c)$. If $E \vdash (a \upharpoonright b) \upharpoonright c : A$ then by Lemma 30(1) we have $E \vdash (a \upharpoonright b) : Proc$, $E \vdash c : A$ and $dom(a \upharpoonright b) \cap dom(c) = \varnothing$. Similarly, Lemma 30(1) applied to $E \vdash (a \upharpoonright b) : Proc$ gives $E \vdash a : Proc$, $E \vdash b : Proc$ and $dom(a) \cap dom(b) = \varnothing$. Since $dom(a \upharpoonright b) = dom(a) \cup dom(b)$ we have that the sets $dom(a)$, $dom(b)$ and $dom(c)$ are pairwise disjoint. Rule (Val Par) applied to $E \vdash b : Proc$, $E \vdash c : A$ and $dom(b) \cap dom(c) = \varnothing$ gives $E \vdash b \upharpoonright c : A$. Rule (Val Par) applied to $E \vdash a : Proc$, $E \vdash (b \upharpoonright c) : A$ and $dom(a) \cap dom(b \upharpoonright c) = \varnothing$ gives $E \vdash a \upharpoonright (b \upharpoonright c) : A$. Part (2) follows by a similar argument.

(Struct Par Comm) We have $(a \upharpoonright b) \upharpoonright c \equiv (b \upharpoonright a) \upharpoonright c$. If $E \vdash (a \upharpoonright b) \upharpoonright c : A$ we argue much as in the (Struct Par Assoc) case, to deduce $E \vdash a : Proc$, $E \vdash b : Proc$, $E \vdash c : A$ and that the sets $dom(a)$, $dom(b)$ and $dom(c)$ are pairwise disjoint. Applying (Val Par) twice we deduce $E \vdash (b \upharpoonright a) \upharpoonright c : A$. Part (2) follows by symmetry.

(Struct Res Res) We have $(\nu p)(\nu q)a \equiv (\nu q)(\nu p)a$.

Suppose $E \vdash (\nu p)(\nu q)a : A$. By renaming bound variables, we can assume without loss of generality that $p \neq q$. Applying Lemma 30(3) we get $E, p : B, q : C \vdash a : A$ for some B and C and $\{p, q\} \subseteq dom(a)$. By Lemma 35, $E, q : C, p : B \vdash a : A$. Hence by (Val Res), $E, q : C \vdash (\nu p)a : A$. Applying (Val Res) again, $E \vdash (\nu q)(\nu p)a : A$. Part (2) follows by symmetry.

(Struct Par 1) We have $(\nu p)(a \upharpoonright b) \equiv a \upharpoonright (\nu p)b$ obtained from $p \notin fn(a)$. Assume $E \vdash (\nu p)(a \upharpoonright b) : A$. Lemma 10(1) and (4) applied to this judgment gives (for some B), $E, p : B \vdash a : Proc$, $E, p : B \vdash b : A$, $dom(a) \cap dom(b) = \varnothing$ and $p \in dom(a) \cup dom(b)$. Lemma 9(1) and $p \notin fn(a)$ imply $p \notin dom(a)$. Since $p \in dom(a) \cup dom(b)$ we must have $p \in dom(b)$. By (Val Res), $E \vdash (\nu p)b : A$. Since $p \notin fn(a)$, $E \vdash a : Proc$ by Lemma 33. Now, $dom(a) \cap dom((\nu p)b) = dom(a) \cap (dom(b) - \{p\}) = \varnothing$, since $dom(a) \cap dom(b) = \varnothing$. Hence (Val Par) gives $E \vdash a \upharpoonright (\nu p)b : A$.

Part (2) follows similarly. If $E \vdash a \upharpoonright (\nu p)b : A$ then by Lemma 30, $E \vdash a : Proc$, $E, p : B \vdash b : A$, $p \in dom(b)$ and $dom(a) \cap (dom(b) - \{p\}) = \varnothing$ for some B. By Lemma 34, $E, p : B \vdash a : Proc$ since $p \notin fn(a)$.

Hence by (Val Par), $E, p : B \vdash a \Rsh b : A$. Finally, (Val Res) gives $E \vdash (\nu p)(a \Rsh b) : A$.

(Struct Par 2) Similar to (Struct Par 1).

(Struct Let Assoc) If $y \notin \mathit{fn}(c)$ we have $\mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c \equiv \mathit{let}\ y{=}a\ \mathit{in}\ \mathit{let}\ x{=}b\ \mathit{in}\ c$. Assume $E \vdash \mathit{let}\ x{=}(\mathit{let}\ y{=}a\ \mathit{in}\ b)\ \mathit{in}\ c : C$. Lemma 30(2) gives B, C' with $E \vdash B{<:}\mathit{Exp}$, $E \vdash C'{<:}\mathit{Exp}$, $E \vdash \mathit{let}\ y{=}a\ \mathit{in}\ b : B$, $E, x : B \vdash c : C'$, $\mathit{dom}(c) = \varnothing$ and $E \vdash C'{<:}C$. Applying Lemma 30(2) to $E \vdash \mathit{let}\ y{=}a\ \mathit{in}\ b : B$ gives us A, B' with $E \vdash A{<:}\mathit{Exp}$, $E \vdash B'{<:}\mathit{Exp}$, $E \vdash a : A$, $E, y : A \vdash b : B'$, $\mathit{dom}(b) = \varnothing$ and $E \vdash B'{<:}B$. By (Val Subsumption) we have $E, y : A \vdash b : B$. Since $y \notin \mathit{fn}(c)$ we have $E, y : A, x : B \vdash c : C'$ from Lemma 34. By (Val Let) $E, y : A \vdash \mathit{let}\ x{=}b\ \mathit{in}\ c : C'$. Applying (Val Let) again, $E \vdash \mathit{let}\ y{=}a\ \mathit{in}\ \mathit{let}\ x{=}b\ \mathit{in}\ c : C'$. Finally by (Val Subsumption), $E \vdash \mathit{let}\ y{=}a\ \mathit{in}\ \mathit{let}\ x{=}b\ \mathit{in}\ c : C$.

Part (2) follows much as Part (1).

(Struct Res Let) We have $(\nu p)\mathit{let}\ x{=}a\ \mathit{in}\ b \equiv \mathit{let}\ x{=}(\nu p)a\ \mathit{in}\ b$ obtained from $p \notin \mathit{fn}(b)$. For Part (1), we assume $E \vdash (\nu p)\mathit{let}\ x{=}a\ \mathit{in}\ b : B$. Lemma 30(2) and (3) gives A and C such that $E, p : C \vdash a : A$, $E \vdash A{<:}\mathit{Exp}$, $E, p : C, x : A \vdash b : B$, $\mathit{dom}(b) = \varnothing$ and $p \in \mathit{dom}(a)$. From (Val Res) we deduce $E \vdash (\nu p)a : A$. Since $p \notin \mathit{fn}(b)$, $E, x : A \vdash b : B$ by Lemma 33. Hence by (Val Let), $E \vdash \mathit{let}\ x{=}(\nu p)a\ \mathit{in}\ b : B$.

Part (2) follows much as Part (1).

(Struct Par Let) We have $a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c \equiv \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c$. For Part (1), we assume $E \vdash a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c : C$. By Lemma 30(1) and (2) we get $E \vdash a : \mathit{Proc}$, $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$ and B, C' such that $E \vdash B{<:}\mathit{Exp}$, $E \vdash C'{<:}\mathit{Exp}$, $E \vdash b : B$, $E, x : B \vdash c : C'$, $\mathit{dom}(c) = \varnothing$ and $E \vdash C'{<:}C$. Rule (Val Par) implies $E \vdash a \Rsh b : B$ and rule (Val Let) gives $E \vdash \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c : C'$. Finally, (Val Subsumption) allows us to infer $E \vdash \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c : C$.

For Part (2), we assume $E \vdash \mathit{let}\ x{=}(a \Rsh b)\ \mathit{in}\ c : C$. Much as before, we deduce that $E \vdash a : \mathit{Proc}$, $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$ and there are B, C' such that $E \vdash B{<:}\mathit{Exp}$, $E \vdash C'{<:}\mathit{Exp}$, $E \vdash b : B$, $E, x : B \vdash c : C'$, $\mathit{dom}(c) = \varnothing$ and $E \vdash C'{<:}C$. From (Val Par) and (Val Let) we deduce $E \vdash a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c : C'$ and from (Val Subsumption), $E \vdash a \Rsh \mathit{let}\ x{=}b\ \mathit{in}\ c : C$. □

We show that reduction preserves typings:

Lemma 37 *If $E \vdash a : A$ and $a \rightarrow b$ then $E \vdash b : A$.*

Proof We prove this by induction on the derivation of $a \rightarrow b$. We consider each of the rules which may derive $a \rightarrow b$ in turn:

(Red Select) We have $(p \mapsto d) \Rsh p.\ell_j \rightarrow (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\}$ where $d = [\ell_i = \varsigma(x_i)b_i\,^{i \in 1..n}]$ and $j \in 1..n$. If $E \vdash (p \mapsto d) \Rsh p.\ell_j : A$ then by Lemma 30(1), $E \vdash p \mapsto d : \mathit{Proc}$ and $E \vdash p.\ell_j : A$. Lemma 30(6) applied to $E \vdash p \mapsto d : \mathit{Proc}$ gives a $B = [\ell_i : B_i\,^{i \in 1..n}]$ such that $E = E_1, p : B, E_2$, $\mathit{dom}(b_i) = \varnothing$ and $E, x_i : B \vdash b_i : B_i$ for each $i \in 1..n$. Lemma 30(4) applied to $E \vdash p.\ell_j : A$ gives a $B' = [\ell_i : B'_i\,^{i \in 1..n'}]$ such that $E \vdash p : B'$, $j \in 1..n'$ and $E \vdash B'_j{<:}A$. From Lemma 31 applied to $E_1, p : B, E_2, x_i : B \vdash b_j : B_j$ we get $E_1, p : B, E_2 \vdash b_j\{\!\{x_j \leftarrow p\}\!\} : B_j$. From Lemma 30(4) we deduce $E \vdash B{<:}B'$ and hence from (Sub Object) that $n' < n$ and $B'_j = B_j$. Hence by (Val Subsumption) $E \vdash b_j\{\!\{x_j \leftarrow p\}\!\} : A$. Finally, from (Val Par) we deduce $E \vdash (p \mapsto d) \Rsh b_j\{\!\{x_j \leftarrow p\}\!\} : A$.

(Red Update) We have $(p \mapsto d) \Rsh p.\ell_j \Leftarrow \varsigma(x)b \rightarrow (p \mapsto d') \Rsh p$ where $d = [\ell_i = \varsigma(x_i)b_i\,^{i \in 1..n}]$ and $d' = [\ell_j = \varsigma(x)b, \ell_i = \varsigma(x_i)b_i\,^{i \in 1..n-\{j\}}]$. If $E \vdash (p \mapsto d) \Rsh p.\ell_j \Leftarrow \varsigma(x)b : A$ then by Lemma 30(1) and (6) we get a $B = [\ell_i : B_i\,^{i \in 1..n}]$ such that $E = E_1, p : B, E_2$, $\mathit{dom}(b_i) = \varnothing$ and $E, x_i : B \vdash b_i : B_i$ for each $i \in 1..n$. From Lemma 30(1), (4) and (5) we get a $B' = [\ell_i : B'_i\,^{i \in 1..n'}]$ such that $E \vdash p : B'$, $j \in 1..n'$, $\mathit{dom}(b) = \varnothing$, $E, x : B' \vdash b : B'_j$, $E \vdash B'{<:}A$ and $E \vdash B{<:}B'$. From $E \vdash B{<:}B'$ and (Sub Object) we see that $n' < n$ and $B_j = B'_j$.

From Lemma 32 applied to $E, x : B' \vdash b : B_j$ and $E \vdash B{<:}B'$ we can deduce $E, x : B \vdash b : B_j$. From this with $E = E_1, p : B, E_2$ and $E, x_i : B \vdash b_i : B_i$ for $i \in 1..n - \{j\}$ we can deduce $E \vdash (p \mapsto d') : \mathit{Proc}$. The judgment $E \vdash p : A$ follows from (Val u) and (Val Subsumption). Finally we deduce $E \vdash (p \mapsto d') \Rsh p : A$ from (Val Par).

(Red Clone) We have $(p \mapsto d) \Rsh \mathit{clone}(p) \rightarrow (p \mapsto d) \Rsh (\nu q)((q \mapsto d) \Rsh q)$, where $d = [\ell_i = \varsigma(x_i)b_i\,^{i \in 1..n}]$ and $q \notin \mathit{fn}(p \mapsto d)$. Suppose $E \vdash (p \mapsto d) \Rsh \mathit{clone}(p) : B$. Then by Lemma 30(1, 6, 9) we deduce $E = E_1, p : A, E_2$, $E \vdash A{<:}B$ and $E, x : A \vdash b_i : B_i$ for $i \in 1..n$, where $A = [\ell_i : B_i\,^{i \in 1..n}]$. From Lemma 34 we can deduce $E, q : A, x_i : A \vdash b_i : B_i$ for each $i \in 1..n$ and hence from (Val Object) we derive $E, q : A \vdash q \mapsto d : \mathit{Proc}$. From (Val u), (Val Par) and (Val Subsumption) we get $E, q : A \vdash (q \mapsto d) \Rsh q : B$. Hence by (Val Res), $E \vdash (\nu q)((q \mapsto d) \Rsh q) : B$. Finally by (Val Par), $E \vdash (p \mapsto d) \Rsh (\nu q)((q \mapsto d) \Rsh q) : B$ as required.

(Red Let Result) We have $\mathit{let}\ x{=}p\ \mathit{in}\ b \to b\{\!\!\{x \leftarrow p\}\!\!\}$. If we have $E \vdash \mathit{let}\ x{=}p\ \mathit{in}\ b : B$ then by Lemma 30(2), there are A, B' such that $E \vdash A{<:}\mathit{Exp}$, $E \vdash B'{<:}\mathit{Exp}$, $E \vdash p : A$, $E, x : A \vdash b : B'$, $\mathit{dom}(b) = \varnothing$ and $E \vdash B'{<:}B$. By Lemma 31, $E \vdash b\{\!\!\{x \leftarrow p\}\!\!\} : B'$ since $E \vdash p : A$ and $E, x : A \vdash b : B'$. Finally by (Val Subsumption), $E \vdash b\{\!\!\{x \leftarrow p\}\!\!\} : B$.

(Red Res) We have $(\nu p)a \to (\nu p)a'$ obtained from $a \to a'$. If $E \vdash (\nu p)a : A$ then by Lemma 30(3) there is B such that $E, p : B \vdash a : A$, and $p \in \mathit{dom}(a)$. By induction, $E \vdash a' : A$. Lemma 9(5) gives $\mathit{dom}(a') = \mathit{dom}(a)$, so $p \in \mathit{dom}(a')$. Hence $E \vdash (\nu p)a' : A$.

(Red Par 1) We have $a \Rsh b \to a' \Rsh b$ obtained from $a \to a'$. If $E \vdash a \Rsh b : A$ then by Lemma 30(1) we have $E \vdash a : \mathit{Proc}$, $E \vdash b : A$ and $\mathit{dom}(a) \cap \mathit{dom}(b) = \varnothing$. Hence by induction, $E \vdash a' : \mathit{Proc}$. Lemma 9(5) gives $\mathit{dom}(a') = \mathit{dom}(a)$. Rule (Val Par) gives $E \vdash a' \Rsh b : A$.

(Red Par 2) Similar to (Red Par 1).

(Red Let) We have $\mathit{let}\ x{=}a\ \mathit{in}\ b \to \mathit{let}\ x{=}a'\ \mathit{in}\ b$ obtained from $a \to a'$. If $E \vdash \mathit{let}\ x{=}a\ \mathit{in}\ b : B$ then by Lemma 30(2), there are A, B' such that $E \vdash A{<:}\mathit{Exp}$, $E \vdash B'{<:}\mathit{Exp}$, $E \vdash a : A$, $E, x : A \vdash b : B'$, $\mathit{dom}(b) = \varnothing$ and $E \vdash B'{<:}B$. By induction, we have $E \vdash a' : A$. Hence by (Val Let), $E \vdash \mathit{let}\ x{=}a'\ \mathit{in}\ b : B'$. By (Val Subsumption) we deduce $E \vdash \mathit{let}\ x{=}a'\ \mathit{in}\ b : B$.

(Red Struct) We have $a \to b$ obtained from $a \equiv a'$, $a' \to b'$ and $b' \to b$. If $E \vdash a : A$ then by Lemma 36, $E \vdash a' : A$. The induction hypothesis applied to $a' \to b'$ gives $E \vdash b' : A$. Finally, applying Lemma 36 again, $E \vdash b : A$. □

Proof of Theorem 3

(1) *If* $E \vdash a : A$ *and* $a \equiv b$ *then* $E \vdash b : A$.

(2) *If* $E \vdash a : A$ *and* $a \to b$ *then* $E \vdash b : A$.

Proof Part (1) follows immediately from Lemma 36. Part (2) follows immediately from Lemma 37. □

A proposition analogous to Theorem 3 holds for *Proc*-indexed structural congruence and reduction:

Proposition 38 *Suppose $E \vdash a : \mathit{Proc}$. Then:*

(1) *If $a \overset{\mathit{Proc}}{\equiv} b$ then $E \vdash b : \mathit{Proc}$.*

(2) *If $a \overset{\mathit{Proc}}{\rightarrow} b$ then $E \vdash b : \mathit{Proc}$.*

Proof

(1) If $a \overset{\mathit{Proc}}{\equiv} b$ then there is a fresh p with $a \Rsh p \equiv b \Rsh p$. By (Val Par) and Lemma 34, $E, p : [] \vdash a \Rsh p : []$. (Our choice of the type $[]$ for the name p is somewhat arbitrary.) By Lemma 36, $E, p : [] \vdash b \Rsh p : []$. By Lemma 30(1), $E, p : [] \vdash b : \mathit{Proc}$. Since $p \notin \mathit{fn}(b)$, $E \vdash b : \mathit{Proc}$ by Lemma 33.

(2) If $a \overset{\mathit{Proc}}{\rightarrow} b$ then $a \overset{\mathit{Proc}}{\equiv} a' \rightarrow b' \overset{\mathit{Proc}}{\equiv} b$. From Part (2), $E \vdash a' : \mathit{Proc}$. From Part(3), $E \vdash b' : \mathit{Proc}$. Finally, Part (2) gives $E \vdash b : \mathit{Proc}$. □