A theory of bisimulation for a fragment of concurrent ML with local names

Alan Jeffrey
CTI, DePaul University
243 South Wabash Ave
Chicago IL 60604, USA
ajeffrey@cs.depaul.edu

Julian Rathke
COGS, University of Sussex
Brighton BN1 9QH, UK
julianr@cogs.susx.ac.uk

Revised manuscript for TCS, July 2003

Abstract

Concurrent ML is an extension of Standard ML with π-calculus-like primitives for multi-threaded programming. CML has a reduction semantics, but to date there has been no labelled transition system semantics provided for the entire language. In this paper, we present a labelled transition semantics for a fragment of CML called μνCML which includes features not covered before: dynamically generated local channels and thread identifiers. We show that weak bisimilarity for μνCML is a congruence, and coincides with barbed bisimulation congruence. We also provide a variant of Sangiorgi's normal bisimulation for μνCML, and show that this too coincides with bisimilarity.

Keywords: concurrency, higher-order functions, bisimulation, local names.

1 Introduction

Concurrent threads of program execution have now become part of a standard toolkit available in modern object-oriented programming languages such as C++ and Java. Their use can ease design of systems, improve readability of code and improve performance. In order to take advantage of the flexibility offered by concurrent programming, combined with the expressive programming style offered by functional languages, Reppy [26] defined and implemented Concurrent ML (CML), an extension of Standard ML [28] with concurrency primitives based on Milner's CCS [19] and π-calculus [20, 21].

In keeping with practice for ML, Reppy [26] and Berry, Milner and Turner [5] provided CML with a reduction semantics, based on the SML language definition [28]. Such a semantics allows the proof of many important properties of the language such as subject reduction, but does not directly support a notion of program equivalence. An understanding of program equivalence for any programming language is important both in practical terms, for instance, for rewrites in compiler optimizations, and for more theoretical concerns such as expressivity of certain useful, but inessential, language features.

Typically a notion of equivalence for a programming language will be defined by identifying some basic observable property of programs, such as termination or production of a particular value, and then ask that equivalent programs display the same observable properties in all program contexts. It is this latter requirement that makes reasoning about such equivalence appear to be a cumbersome task. By modelling the programming language it is often possible to characterise the program equivalence in a direct manner which avoids such quantification over contexts of the language and can simply reasoning about program equivalence significantly. We choose to study a notion of equivalence based upon barbed bisimulation [22] appropriate for a concurrent programming language.

Initial efforts towards characterising program equivalence for CML were made by Ferreira, Hennessy and Jeffrey [6] who provided a labelled transition system semantics for a fragment of CML, and showed that the resulting theory of bisimulation was a congruence. This result is significant in the sense that equivalences established using this semantics could be used to justify equational rewriting for program fragments. The sublanguage of CML covered, however, was more restrictive than Reppy's reduction semantics, and in particular did not treat two important features of the language: channel generation, and thread identifiers. In this paper, we show how these features can also be modelled cleanly using labelled transition semantics.

Channel generation is an important primitive of CML: it allows new communication channels to be created dynamically, and for their local scope to be controlled in the style of the π-calculus. Much of the power of CML rests on local name generation, for example it is used in Reppy's coding of recursion into λ_{cv}. Many languages, such as Fournet *et al.*'s join calculus [7, 9], Boudol's blue calculus [3], Thomsen's CHOCS [33] and Sangiorgi's higher-order π-calculus [29] include encodings of the λ-calculus which rely upon local name generation. This paper provides the first direct characterization of program equivalence for the λ-calculus together with π-style concurrency.

Our richer fragment of the language CML also contains thread identifiers. In particular, the type for the spawn primitive which creates new threads is given by

$$\mathsf{spawn} : (\mathsf{unit} \to \sigma) \to \sigma\,\mathsf{thread}$$

That is, spawn takes an inactive thread and sets it running concurrently with the new thread. The result of this command is the identifier name of the new thread, which can then be used to block waiting for another thread to terminate, using:

$$\mathsf{join} : \sigma\,\mathsf{thread} \to \sigma$$

Calling $\mathsf{join}\,t$ causes the current thread to wait for t to terminate with some value v, which is then returned. Ferreira, Hennessy and Jeffrey's treatment of CML ignored thread identifiers entirely. Their type for thread spawning was simply:

$$\mathsf{spawn} : (\mathsf{unit} \to \mathsf{unit}) \to \mathsf{unit}$$

Thread identifiers have received less attention in the literature than channels, largely because they can be easily expressed in terms of channels. In this paper, we provide a semantics for thread identifiers, partly because they already exist in CML, but also because they simplify the labelled transition system semantics considerably. This use of thread identifiers is similar to the use of function definitions in Fournet *et al.*'s join calculus, and object pointers in Gordon and Hankin's [11] concurrent object calculus, but with the important difference that thread identifiers can name actively executing code rather than just functions or objects. They can also be seen as a restricted form of Cardelli and Gordon's [4] ambients, where the ambient tree is flat and names are used linearly.

Characterising equivalence for higher-order languages featuring locally declared names is known to be a difficult task [25]. The problems arise particularly for bisimulation based approaches when it comes to demonstrating that bisimilarity forms a congruence. The authors made initial steps towards the current labelled transition semantics for local names in [15]. We proposed there a novel transition system which incorporated a notion of privacy as a means of studying locality in the small sequential language ν-calculus of Pitts and Stark, [24] with the intention of developing useful proof techniques for establishing congruence of bisimilarity. We discovered that the pure interaction of higher-order functions and locally declared names generates a subtle and complex notion of equivalence which can be simplified greatly by the addition of side-effecting computation. The present work can also be seen as a case in support of this argument by investigating the problem in the concurrent setting. In fact we adopt the same technique for modelling local names in this paper; unfortunately however, the proof techniques for establishing congruence of bisimilarity are very different.

In the previous paper we used a variant of a proof technique known as Howe's method [14, 10]. Unfortunately, this is not available to us here as Howe's method relies heavily upon the syntactic structure of terms. The use of a sophisticated structural equivalence such as the parallel composition operator forming a commutative monoid is in direct conflict with Howe's proof technique. If one were to attempt a reduction semantics for the current language without recourse to a structural equivalence then one would find that, in order to establish the notion of substitutivity required for the Howe relation, we require congruence with respect to parallel composition; this is of course too difficult to show directly. Instead of employing Howe's method we adapt Sangiorgi's [29] trigger semantics from the higher-order π-calculus. The approach we develop here generalizes the trigger encodings and their corresponding correctness proofs to accommodate the functional setting. This is achieved by introducing a hierarchy of operational semantics based on type-order and establishing correctness throughout the hierarchy. We also make use of a 'bisimulation up to' technique [30] based on confluent reduction to simplify and structure the correctness proofs.

In this paper, we consider congruence of *configurations* (that is top-level programs consisting of communicating threads) rather than *terms* (that is program fragments). In particular, this allows us to assume that configurations do not have any free variables, and so we do not have to deal with channel aliasing. We leave a treatment of such features for future work.

This paper originally appeared as an extended abstract in [16]; in this paper we provide more details, including complete proofs.

The remainder of the paper is organized as follows: in the next section we present the fragment of CML which contains the features of interest to us and define a type system, reduction semantics and notion of observational equivalence for the language. In Section 3 we describe our labelled transition system semantics and offer justification for our transition rules by demonstrating a contextuality result. Bisimulation equivalence for our language is also presented here. Section 4 is given over to establishing that bisimilarity is a congruence. We follow this up with a much tractable labelled transition semantics for which bisimulation equivalence coincides with the equivalence of Section 3. Finally we conclude with some closing remarks about related and future work.

2 A core fragment of CML

We examine the language $\mu\nu$CML, which is a subset of Reppy's [26] concurrent functional language CML, given by extending the simply-typed λ-calculus with primitives for thread creation and inter-

$$\frac{}{\Gamma;\Delta,n:\sigma\vdash n:\sigma} \qquad \frac{\Gamma;\Delta\vdash e:\sigma_1 \qquad \Gamma,x:\sigma_1;\Delta\vdash t:\sigma_2}{\Gamma;\Delta\vdash \mathsf{let}\, x=e\,\mathsf{in}\, t:\sigma_2}$$

$$\frac{\Gamma;\Delta\vdash v_1:\sigma\,\mathsf{thread} \qquad \Gamma;\Delta\vdash v_2:\sigma\,\mathsf{thread}}{\Gamma;\Delta\vdash v_1=v_2:\mathsf{bool}} \qquad \frac{\Gamma;\Delta\vdash v_1:\sigma\,\mathsf{chan} \qquad \Gamma;\Delta\vdash v_2:\sigma\,\mathsf{chan}}{\Gamma;\Delta\vdash v_1=v_2:\mathsf{bool}}$$

$$\frac{\Gamma;\Delta\vdash v_1:B \qquad \Gamma;\Delta\vdash v_2:B}{\Gamma;\Delta\vdash v_1=v_2:\mathsf{bool}} \qquad \frac{\Gamma;\Delta\vdash v:\mathsf{bool} \qquad \Gamma;\Delta\vdash t_1:\sigma \qquad \Gamma;\Delta\vdash t_2:\sigma}{\Gamma;\Delta\vdash \mathsf{if}\, v\,\mathsf{then}\, t_1\,\mathsf{else}\, t_2:\sigma}$$

$$\frac{\Gamma;\Delta\vdash v:\mathsf{unit}}{\Gamma;\Delta\vdash \mathsf{chan}\, v:\sigma\,\mathsf{chan}} \qquad \frac{\Gamma;\Delta\vdash v:\sigma\,\mathsf{chan}*\sigma}{\Gamma;\Delta\vdash \mathsf{send}\, v:\mathsf{unit}} \qquad \frac{\Gamma;\Delta\vdash v:\sigma\,\mathsf{chan}}{\Gamma;\Delta\vdash \mathsf{recv}\, v:\sigma}$$

$$\frac{\Gamma;\Delta\vdash v:\sigma\,\mathsf{thread}}{\Gamma;\Delta\vdash \mathsf{join}\, v:\sigma} \qquad \frac{\Gamma;\Delta\vdash v:\mathsf{unit}\to\sigma}{\Gamma;\Delta\vdash \mathsf{spawn}\, v:\sigma\,\mathsf{thread}}$$

Figure 1: Thread type inference rules (not showing the usual simply typed λ-calculus rules)

$$\frac{;\Delta,n:\sigma\,\mathsf{thread}\vdash t:\sigma}{\Delta,n:\sigma\,\mathsf{thread}\vdash n[t]} \qquad \frac{}{\Delta\vdash \mathbf{0}} \qquad \frac{\Delta\vdash C_1 \qquad \Delta\vdash C_2}{\Delta\vdash C_1\,\|\,C_2} \qquad \frac{\Delta,n:\sigma\vdash C}{\Delta\vdash \nu n:\sigma.\,C}$$

Figure 2: Configuration type inference rules

thread communication. Threads can communicate in two ways: by π-calculus-style synchronous channels, or by waiting for a thread to terminate.

$\mu\nu$CML contains many of the features of Ferreira, Hennessy and Jeffrey's [6] μCML, but is missing the event type and its associated functions. We believe that adding Ferreira, Hennessy and Jeffrey's treatment of the event type back into $\mu\nu$CML would pose few technical problems.

The grammar for $\mu\nu$CML types is obtained by extending that for simply-typed lambda calculus with type constructors for thread identifiers and channel identifiers. We assume a grammar B for base types, including at least the unit type unit and the boolean type bool. The grammar of types is given:

$$\sigma ::= B \mid \sigma*\sigma \mid \sigma\to\sigma \mid \sigma\,\mathsf{chan} \mid \sigma\,\mathsf{thread}$$

Since we are using a call-by-value reduction semantics, we need a grammar for values. We assume an infinite set of variables x and names n, and some base values b including at least $()$, true and false. The grammar of $\mu\nu$CML values is given:

$$v ::= b \mid (v,v) \mid \lambda x:\sigma.\,t \mid n \mid x$$

Threads take the form $\mathsf{let}\, x_1=e_1\,\mathsf{in}\cdots\mathsf{let}\, x_n=e_n\,\mathsf{in}\, v$ and consist of a stack of expressions $e_1,\ldots,e_n$ to be evaluated, followed by a return value v. The grammar of $\mu\nu$CML threads is given:

$$t ::= v \mid \mathsf{let}\, x=e\,\mathsf{in}\, t$$

$$\begin{array}{rcll} \mathbf{0} \parallel C & \equiv & C & \\ (C_1 \parallel C_2) \parallel C_3 & \equiv & C_1 \parallel (C_2 \parallel C_3) & \\ C_1 \parallel C_2 & \equiv & C_2 \parallel C_1 & \\ C_1 \parallel \nu n \,.\, C_2 & \equiv & \nu n \,.\, (C_1 \parallel C_2) & (n \notin C_1) \\ \nu n \,.\, \nu n' \,.\, C & \equiv & \nu n' \,.\, \nu n \,.\, C & \end{array}$$

Figure 3: Axioms for structural congruence $C \equiv C'$

An expression consists of the usual simply-typed λ-calculus with booleans, together with primitives for multi-threaded computation:

- $\mathsf{chan}\,()$ creates a new channel identifier.
- $\mathsf{send}\,(c, v)$ sends value v along channel c to a matching expression $\mathsf{recv}\,c$, which returns v.
- $\mathsf{spawn}\,v$ creates a new named thread, which executes $v()$, and returns the thread identifier.
- $\mathsf{join}\,i$ blocks waiting for the thread with identifier i to terminate with value v, which is then returned (this is similar to Reppy's [26] *joinVal* function).

The grammar of $\mu\nu$CML expressions is given:

$$\begin{array}{rl} e ::= & t \mid \mathsf{fst}\,v \mid \mathsf{snd}\,v \mid v\,v \mid \mathsf{if}\,v\,\mathsf{then}\,t\,\mathsf{else}\,t \mid v = v \mid \\ & \mathsf{send}\,v \mid \mathsf{recv}\,v \mid \mathsf{chan}\,v \mid \mathsf{join}\,v \mid \mathsf{spawn}\,v \end{array}$$

The use of values rather than expressions in many of the above terms may appear to be rather restrictive however, in light of the fact that we are using call-by-value reduction we can use simple syntactic sugar to recover many terms such as

$$\mathsf{fst}\,e \equiv \mathsf{let}\,x = e\,\mathsf{in}\,\mathsf{fst}\,x.$$

We will also make use of a sequential composition operator defined by

$$e; e' \equiv \mathsf{let}\,x = e\,\mathsf{in}\,e'$$

and a parallel composition operator defined by

$$e' \mid e \equiv \mathsf{spawn}\,(\lambda x \,.\, e'); e$$

where x does not occur free in e'. The restricted syntax makes the operational semantics much simpler to present: for example we do not need to introduce 'evaluation contexts' or work explicitly with nested let-blocks, since these are not allowed in the syntax. This restricted syntax is inspired by Moggi's [23] computational λ-calculus, although we are treating the separation of values and terms syntactically rather than by type.

Note that we do not include recursive functions explicitly in this syntax, since they can be coded using threads:

$$\mu f.\lambda x.e = \mathsf{let}\,c = \mathsf{chan}\,()\,\mathsf{in}\,\mathsf{let}\,f = (\lambda x \,.\, (\mathsf{let}\,g = \mathsf{recv}\,c\,\mathsf{in}\,(\mathsf{send}\,(c, g) \mid g(x))))\,\mathsf{in}\,(\mathsf{send}\,(c, \lambda x.e) \mid f)$$

We consider all terms up to α-conversion given by the consistent renaming of bound names and variables.

The type inference rules for threads are given in Figure 1. The type judgements are of the form:

$$\Gamma;\Delta \vdash t : \sigma$$

where Γ is the type context for free variables (of the form $x : \sigma$) and Δ the type context for free names (of the form $n : \sigma\,\mathsf{chan}$ or $n : \sigma\,\mathsf{thread}$).

In order to present the reduction semantics for $\mu\nu$CML it will be useful to describe the configurations of evaluation. The basic unit of a configuration is a named thread. These can be combined using $\|$ to express concurrency and the configuration $\mathbf{0}$ represents the empty configuration and forms a unit for $\|$. We use the scoping operator $\nu n : \sigma\,.\,[\cdot]$ to delimit the portion of the configuration in which the identifier n is deemed to exist. The grammar for configurations is as follows:

$$C ::= \mathbf{0} \mid C \,\|\, C \mid \nu n : \sigma\,.\,C \mid n[t]$$

Let the thread names of a configuration be defined:

$$\begin{array}{rclcrcl} tn(\mathbf{0}) & = & \emptyset & \qquad & tn(C_1 \,\|\, C_2) & = & tn(C_1) \cup tn(C_2) \\ tn(n[t]) & = & \{n\} & & tn(\nu n\,.\,C) & = & tn(C) \setminus \{n\} \end{array}$$

We will only consider configurations in which threads are named uniquely, that is:

> In any configuration $C_1 \,\|\, C_2$, the thread names of C_1 and C_2 are disjoint.

We present the type inference rules for configurations in Figure 2. Judgements are of the form:

$$\Delta \vdash C$$

A *context*, $\mathcal{C}[\cdot_\Delta]$, is a configuration which contains a single occurrence of a typed indexed 'hole' $[\cdot_\Delta]$ which is well-typed according to the rules in Figure 2 along with

$$\Delta, \Delta' \vdash [\cdot_\Delta]$$

Placement of a configuration $\Delta \vdash C$ in a context $\Delta' \vdash \mathcal{C}[\cdot_\Delta]$ is standard and respects well-typedness. There is an evident structural congruence on configurations given in Figure 3 which should be familiar to readers from the π-calculus [20]. As we can see then, a configuration is simply a collection of named threads running concurrently, with shared private names and, up to structural congruence, can be written in the form

$$\nu \vec{m} : \vec{\sigma}\,.\,(n_1[t_1] \,\|\, \ldots \,\|\, n_k[t_k])$$

We will, on occasion, write $n[e]$ as shorthand for the configuration $n[\mathsf{let}\, x = e \,\mathsf{in}\, x]$.

Let the reduction relation $C \to C'$ be the least precongruence on configurations which includes the axioms in Figure 4 and satisfies: if $C \equiv C' \to C'' \equiv C'''$ then $C \to C'''$. We have split the reduction axioms into confluent rules $C \xrightarrow{\beta} C'$ and one non-confluent communication rule $C \xrightarrow{\tau} C'$. We will often annotate the reduction $C \to C'$ with β or τ to indicate which of the basic axioms were used to infer $C \to C'$. Let $\Rightarrow$ be the reflexive transitive closure of $\to$.

$$
\begin{array}{rcll}
n[\mathsf{let}\,x = v\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[t[v/x]] & \\
n[\mathsf{let}\,x_2 = (\mathsf{let}\,x_1 = e_1\,\mathsf{in}\,t_2)\,\mathsf{in}\,t_3] & \xrightarrow{\beta} & n[\mathsf{let}\,x_1 = e_1\,\mathsf{in}\,(\mathsf{let}\,x_2 = t_2\,\mathsf{in}\,t_3)] & (x_1 \notin t_3) \\
n[\mathsf{let}\,x = \mathsf{fst}\,(v_1, v_2)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = v_1\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = \mathsf{snd}\,(v_1, v_2)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = v_2\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = (\lambda x_1\,.\,t_1)\,v_1\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = (\mathsf{let}\,x_1 = v_1\,\mathsf{in}\,t_1)\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = (\mathsf{if}\,\mathsf{true}\,\mathsf{then}\,t_1\,\mathsf{else}\,t_2)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = t_1\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = (\mathsf{if}\,\mathsf{false}\,\mathsf{then}\,t_1\,\mathsf{else}\,t_2)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = t_2\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = (v = v)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = \mathsf{true}\,\mathsf{in}\,t] & \\
n[\mathsf{let}\,x = (v_1 = v_2)\,\mathsf{in}\,t] & \xrightarrow{\beta} & n[\mathsf{let}\,x = \mathsf{false}\,\mathsf{in}\,t] & (v_1 \neq v_2) \\
n[\mathsf{let}\,x = \mathsf{chan}\,()\,\mathsf{in}\,t] & \xrightarrow{\beta} & \nu n'\,.\,n[\mathsf{let}\,x = n'\,\mathsf{in}\,t] & (n' \text{ fresh}) \\
n_1[\mathsf{let}\,x = \mathsf{join}\,n_2\,\mathsf{in}\,t] \parallel n_2[v] & \xrightarrow{\beta} & n_1[\mathsf{let}\,x = v\,\mathsf{in}\,t] \parallel n_2[v] & \\
n[\mathsf{let}\,x = \mathsf{spawn}\,v\,\mathsf{in}\,t] & \xrightarrow{\beta} & \nu n'\,.\,n[\mathsf{let}\,x = n'\,\mathsf{in}\,t] \parallel n'[v\,()] & (n' \text{ fresh}) \\
\\
n_1[\mathsf{let}\,x_1 = \mathsf{send}\,(n, v)\,\mathsf{in}\,t_1] \parallel n_2[\mathsf{let}\,x_2 = \mathsf{recv}\,n\,\mathsf{in}\,t_2] & \xrightarrow{\tau} & n_1[\mathsf{let}\,x_1 = ()\,\mathsf{in}\,t_1] \parallel n_2[\mathsf{let}\,x_2 = v\,\mathsf{in}\,t_2] &
\end{array}
$$

Figure 4: Axioms for reduction precongruence $C \to C'$

2.1 Barbed equivalence, $\approx^b$

We present a notion of observational equivalence for our language following [13], which is a variant of the barbed bisimulation equivalence proposed in [22]. In both approaches a basic observable behaviour, called a barb, is identified. Bisimulation relations are defined so that equivalent processes simulate reduction transitions (coinductively) and barbs. The crucial difference however lies in how the resulting equivalence is demanded to be a congruence. In the approach we follow we ask that each witness for the coinductively defined equivalence be a congruence whereas the approach prescribed in [22] asks solely that the equivalence itself be a congruence. We adopt the former approach because it appears more suitable to our setting in which certain 'asynchronous' transitions are present in the labelled transition semantics of the next section. The interested reader can see [8] for a thorough discussion regarding these two approaches.

In this paper, we take boolean barbs as primitives, and observe threads which terminate with the result true. We believe that the results in this paper are independent of this choice of guard.

We define a *type-indexed relation* $\mathcal{R}$ to be a family of relations $\mathcal{R}_\Delta$ on typed configurations $\Delta \vdash C$. We will often write $\Delta \vDash C_1\ \mathcal{R}\ C_2$ whenever $(\Delta \vdash C_1)\ \mathcal{R}_\Delta\ (\Delta \vdash C_2)$. Furthermore, we call a type-indexed relation $\mathcal{R}$ on configurations *contextual* if it satisfies:

$$\Delta \vDash C_1\ \mathcal{R}\ C_2 \quad \text{implies} \quad \Delta' \vDash \mathcal{C}[C_1]\ \mathcal{R}\ \mathcal{C}[C_2]$$

for all contexts

$$\Delta' \vdash \mathcal{C}[\cdot_\Delta]$$

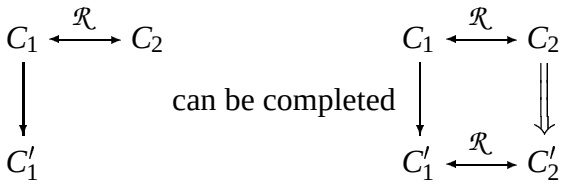
$\mathcal{R}$ is *barbed* if it satisfies:

$$C_1 \mathcal{R} C_2 \quad \text{implies} \quad \forall n \,.\, C_1 \Downarrow_n \text{ iff } C_2 \Downarrow_n$$

where:

$$C \Downarrow_n \quad \text{iff} \quad \exists C' \,.\, C \Rightarrow n[\mathsf{true}] \,\|\, C'$$

$\mathcal{R}$ is *reduction closed* if it satisfies:



and a similar symmetric condition.

Then, *barbed equivalence*, $\approx^b$ is defined to be the largest reduction-closed barbed contextual relation on well-typed configurations. It is routine to show that barbed equivalence is a congruence, since we have required it to be contextual.

Barbed equivalence is a natural definition for a bisimulation-like equivalence, but it is very difficult to reason about, since its definition includes a quantification over all contexts. In the remainder of this paper, we shall present a labelled transition system semantics for $\mu\nu$CML and a coinductive presentation of barbed equivalence based on this.

3 Operational semantics and bisimulation equivalence

3.1 Labelled transition semantics

We make our first steps towards characterizing barbed equivalence using a labelled transition system semantics. We adopt the approach we advocated in [15] by designing a semantics such that:

- Bisimulation can be defined in the standard way, following Gordon [10] and Bernstein's [1] approach to bisimulation for higher-order languages. This contrasts with the higher-order bisimulation used by Thomsen [33] and Ferreira, Hennessy and Jeffrey [6] in which a non-standard notion of bisimulation is proposed whereby processes which emit other processes are compared such that the emitted values must be related independently of the residual processes. Sangiorgi [29] showed this latter approach to be inadequate for higher-order statically scoped languages with name generation.

- Labels are *contextual* in the sense that each labelled transition represents a small program fragment which induces an appropriate reduction. This notion of contextual label has been investigated in depth by Sewell [31] and Leifer and Milner [18].

Our labelled transition system is defined as a relation between well-typed configurations. The rules are presented in Figure 5.In addition to these transitions with labels ranged over by γ, the labelled transition system relation also contains the reduction relation $\rightarrow$ of the previous section, suitably labelled with β and τ: that is $(\Delta \vdash C) \xrightarrow{\beta} (\Delta \vdash C')$ holds whenever $C \xrightarrow{\beta} C'$ holds (and similarly for τ).

$$
\begin{array}{rcl}
(\Delta \vdash n[v]) & \xrightarrow{n\Downarrow} & (\Delta \vdash n[v]) \\
(\Delta \vdash C) & \xrightarrow{\mathsf{weak}.n:\sigma} & (\Delta, n:\sigma \vdash C) \\
(\Delta \vdash n[b]) & \xrightarrow{n.b} & (\Delta \vdash n[b]) \\
(\Delta \vdash n[n'']) & \xrightarrow{n.n''} & (\Delta \vdash n[n'']) \\
(\Delta \vdash n[(v_1, v_2)]) & \xrightarrow{n.\mathsf{fst}.n'} & (\Delta, n' \vdash n[(v_1, v_2)] \parallel n'[v_1]) \\
(\Delta \vdash n[(v_1, v_2)]) & \xrightarrow{n.\mathsf{snd}.n'} & (\Delta, n' \vdash n[(v_1, v_2)] \parallel n'[v_2]) \\
(\Delta \vdash n[\lambda x_1 : \sigma_1 . t_1]) & \xrightarrow{n.@v_1.n'} & (\Delta, n' \vdash n[\lambda x_1 : \sigma_1 . t_1] \parallel n'[\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, t_1]) \quad \text{if } (\Delta \vdash v_1 : \sigma_1) \\
(\Delta \vdash n[\mathsf{let}\, x = \mathsf{send}\,(n'', v)\,\mathsf{in}\, t]) & \xrightarrow{\mathsf{send}(n'').n'} & (\Delta, n' \vdash n[\mathsf{let}\, x = ()\,\mathsf{in}\, t] \parallel n'[v]) \\
(\Delta \vdash n[\mathsf{let}\, x = \mathsf{recv}\, n''\,\mathsf{in}\, t]) & \xrightarrow{\mathsf{recv}(n'',v)} & (\Delta \vdash n[\mathsf{let}\, x = v\,\mathsf{in}\, t]) \qquad \text{if } (\Delta \vdash n'' : \sigma\,\mathsf{chan} \text{ and } \Delta \vdash v : \sigma) \\
(\Delta \vdash C) & \xrightarrow{\mathsf{join}(v).n} & (\Delta \vdash n[v] \parallel C) \qquad \text{if } (\Delta \vdash n : \sigma\,\mathsf{thread} \text{ and } \Delta \vdash v : \sigma)
\end{array}
$$

$$
\frac{(\Delta \vdash C_1) \xrightarrow{\gamma} (\Delta' \vdash C_1')}{(\Delta \vdash C_1 \parallel C_2) \xrightarrow{\gamma} (\Delta' \vdash C_1' \parallel C_2)} \qquad \frac{(\Delta \vdash C_2) \xrightarrow{\gamma} (\Delta' \vdash C_2')}{(\Delta \vdash C_1 \parallel C_2) \xrightarrow{\gamma} (\Delta' \vdash C_1 \parallel C_2')}
$$

$$
\frac{(\Delta, n:\sigma \vdash C) \xrightarrow{\gamma} (\Delta', n:\sigma \vdash C')}{(\Delta \vdash \nu n:\sigma . C) \xrightarrow{\gamma} (\Delta' \vdash \nu n:\sigma . C')}[n \notin \gamma] \qquad \frac{(\Delta, n':\sigma \vdash C) \xrightarrow{n.n'} (\Delta, n':\sigma \vdash C')}{(\Delta \vdash \nu n':\sigma . C) \xrightarrow{n.\nu n'} (\Delta, n':\sigma \vdash C')}[n \neq n']
$$

Figure 5: Labelled transition system semantics

Let α range over γ, τ and β transitions. We define $(\Delta \vdash C) \stackrel{\alpha}{\Longrightarrow} (\Delta' \vdash C')$ as $(\Delta \vdash C) \Rightarrow \xrightarrow{\alpha} \Rightarrow (\Delta' \vdash C')$ and $(\Delta \vdash C) \stackrel{\hat{\alpha}}{\Longrightarrow} (\Delta' \vdash C')$ as $(\Delta \vdash C) \Rightarrow (\Delta' \vdash C')$ when α is β or τ and $(\Delta \vdash C) \stackrel{\alpha}{\Longrightarrow} (\Delta' \vdash C')$ otherwise.

The labels used take various forms. Many are prepended with an identifier, for example, $\xrightarrow{n.b}$; this signifies which named thread we are currently investigating. Some are followed by another identifier, for example, $\xrightarrow{n.\mathsf{fst}.n'}$ indicates that we can observe that thread n has converged to a pair of values and we may take the first component of this pair and test with it in a new thread named n'. Because the only way in which an observer may interact with thread n is by means of a non-destructive join synchronisation we notice that the thread under examination will be unaffected by the test thus allowing subsequent tests upon this pair of values to be performed. This obviates the need for explicit copying transitions to allow repeated testing, *cf.* [15]. The transitions for modelling the communication primitives are not addressed using a thread identifier because the origin of a communication is not an observable property in this language. Similarly, the transition labelled join simply allocates a value to the named thread, irrespective of any term under investigation. It should be clear that such transitions are necessary in order to distinguish, say,

$$n[\mathsf{let}\, x = \mathsf{join}\, n' \,\mathsf{in}\, \mathsf{true}] \not\approx n[\mathsf{let}\, x = \mathsf{join}\, n' \,\mathsf{in}\, \mathsf{false}]$$

However, it is not observable in this language whether a thread is currently waiting on another to terminate. This bears similarity to the situation of the asynchronous π-calculus [2, 12] and the transitions we use are akin to those for *input receptivity* [12]. It is observable whether a particular thread has terminated though and we use the transitions labelled $n \Downarrow$ to allow this. The use of the free name context allows us to model the static scoping discipline present in CML. The intention is that the names in Δ are global and thus known to the observer. The observer is also at liberty to invent fresh names of their own, modelled by the use of the $\mathsf{weak} \,.\, n : \sigma$ transitions. The transition rules we use are essentially those of [15] and the side-conditions in the two rules for inferring transitions under $\nu n.$ contexts ensure privacy and freshness respectively. We only consider transitions between configurations in which threads are named uniquely.

We can now prove some crucial properties of the reduction semantics and labelled transition semantics for $\mu\nu$CML, including subject reduction:

Proposition 3.1 (Subject Reduction) *If* $\Delta \vdash C$ *and* $(\Delta \vdash C) \xrightarrow{\alpha} (\Delta' \vdash C')$ *then* $\Delta' \vdash C'$

Proof: Straightforward induction. □

Proposition 3.2 *If* $C \equiv C'$ *and* $(\Delta \vdash C') \xrightarrow{\alpha} (\Delta' \vdash C'')$ *and* $C'' \equiv C'''$ *then* $(\Delta \vdash C) \xrightarrow{\alpha} (\Delta' \vdash C''')$ *also.*

Proof: Straightforward induction. □

The above proposition states the labelled transition semantics is still well-defined if we consider the transition relation to be defined on structural equivalence classes of well-typed configurations. This allows us to work freely with transitions up to $\equiv$ yet still perform rule induction over the judgements.

It is not too hard to see that the reductions we identified as being β-reductions are in fact confluent. They are not only confluent with respect to other reductions, but in fact with respect to labelled transitions:

Proposition 3.3 *The following diagram can be completed:*

$$\begin{array}{ccc} (\Delta \vdash C) & \xrightarrow{\beta} & (\Delta \vdash C') \\ {\scriptstyle\alpha}\downarrow & & \\ (\Delta' \vdash C'') & & \end{array} \qquad \begin{array}{ccc} (\Delta \vdash C) & \xrightarrow{\beta} & (\Delta \vdash C') \\ {\scriptstyle\alpha}\downarrow & & {\scriptstyle\alpha}\downarrow \\ (\Delta' \vdash C'') & \xrightarrow{\beta} & (\Delta' \vdash C''') \end{array}$$

as

or $C' \equiv C''$ *if* α *is* β.

Proof: Firstly we can easily establish that all β-reductions are, up to structural equivalence, of the form

$$\nu\Delta_0 \,.\, (C_1 \,\|\, C_2) \xrightarrow{\beta} \nu\Delta_0 \,.\, (C_1' \,\|\, C_2)$$

where $C_1 \xrightarrow{\beta} C_1'$ is an instance of a β-reduction axiom.

Now, suppose (wlog) that $(\Delta \vdash \nu\Delta_0 \,.\, (C_1 \,\|\, C_2)) \xrightarrow{\alpha} (\Delta' \vdash C'')$ also and C' is $\nu\Delta_0 \,.\, (C_1' \,\|\, C_2)$. Given this, it is then easy to see by inspecting the reduction and transition axioms that, for $\alpha \neq \beta$, save for the case in which the join synchronisation β-reduction occurs, it must be that $(\Delta \vdash \nu\Delta_0 \,.\, C_2) \xrightarrow{\alpha} (\Delta' \vdash \nu\Delta_0' \,.\, C_2')$

for appropriate Δ_0'. So C'' must be of the form $\nu\Delta_0' \,.\, (C_1 \parallel C_2')$ and if we let C''' be $\nu\Delta_0' \,.\, (C_1' \parallel C_2')$ we are done.

If, however, the β-reduction actually arises as an instance of the join axiom:

$$\nu\Delta_0 \,.\, (n_1[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t] \parallel n_2[v] \parallel C_2) \xrightarrow{\beta} \nu\Delta_0 \,.\, (n_1[\mathsf{let}\, x = v\, \mathsf{in}\, t] \parallel n_2[v] \parallel C_2)$$

then we notice that α may be derived not only from C_2 but also from $n_2[v]$. In this situation we also see that all observations, α, deriving from this value have the general form

$$\Delta \vdash \nu\Delta_0 \,.\, (n_1[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t] \parallel n_2[v] \parallel C_2) \xrightarrow{\alpha} \Delta' \vdash \nu\Delta_0' \,.\, (n_1[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t] \parallel n_2[v] \parallel C_2')$$

thus $n_2[v]$ is again residual in the target term and the α transition cannot preclude the β-reduction.

It only to remains to investigate the case in which the α transition is actually a β-reduction. Clearly, this could be exactly the same β-reduction in C_1 (then $C' \equiv C''$), or it be a different reduction originating entirely in C_2, in which case the two clearly commute. Alternatively, it could be an overlapping instance of the join axiom

$$\begin{aligned}&\nu\Delta_0 \,.\, (n_1[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t] \parallel n_2[v] \parallel n_3[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t'] \parallel C_2)\\ &\quad\xrightarrow{\beta} \nu\Delta_0 \,.\, (n_1[\mathsf{let}\, x = \mathsf{join}\, n_2\, \mathsf{in}\, t] \parallel n_2[v] \parallel n_3[\mathsf{let}\, x = v\, \mathsf{in}\, t'] \parallel C_2)\end{aligned}$$

Again, we notice that the $n_2[v]$ is residual in the target term and this allows the two β-reductions to commute. □

We now make good on our claim that the labelled transitions presented above actually correspond to small reduction-inducing contexts of the language. Barbs play an important role here as we use them to establish whether a context has successfully induced a reduction. We list below the actual terms used to build a context for each label. We write C_γ^Δ to mean the configuration corresponding to the label γ used on a term with free names in Δ with a barb indicating success at a special fresh location l. We also report, at another fresh location m, the names of any fresh identifiers which are created during a transition labelled γ. This is a technical convenience whose use will become apparent in proving bisimilarity complete for barbed equivalence. Given this configuration we build a context for γ simply by placing it in parallel with the hole, that is $C_\gamma^\Delta \parallel [\cdot]$.

$$\begin{aligned}
C^\Delta_{n\Downarrow} &= m[()] \parallel l[\mathsf{join}\, n; \mathsf{true}]\\
C^\Delta_{\mathsf{weak}.n:\sigma} &= \nu n : \sigma \,.\, (m[n] \parallel l[\mathsf{true}])\\
C^\Delta_{n.b} &= m[()] \parallel l[b = \mathsf{join}\, n]\\
C^\Delta_{n.n'} &= m[()] \parallel l[n' = \mathsf{join}\, n]\\
C^\Delta_{n.\nu n'} &= m[\mathsf{join}\, n] \parallel l[(\mathsf{join}\, n) \notin \Delta]\\
C^\Delta_{n.\mathsf{fst}.n'} &= \nu n' \,.\, (m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{fst}\, (\mathsf{join}\, n)])\\
C^\Delta_{n.\mathsf{snd}.n'} &= \nu n' \,.\, (m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{snd}\, (\mathsf{join}\, n)])\\
C^\Delta_{n.@v.n'} &= \nu n' \,.\, (m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{join}\, n\, v])\\
C^\Delta_{\mathsf{recv}(n,v)} &= m[()] \parallel l[\mathsf{send}\, (n, v); \mathsf{true}]\\
C^\Delta_{\mathsf{send}(n).n'} &= \nu n' \,.\, (m[n'] \parallel l[\mathsf{join}\, n'; \mathsf{true}] \parallel n'[\mathsf{recv}\, n])\\
C^\Delta_{\mathsf{join}(v).n} &= m[()] \parallel l[\mathsf{true}] \parallel n[v]
\end{aligned}$$

where we use obvious syntax sugar such as $n \notin \Delta$ and abuse notation by writing Δ to also mean the tuple of names declared in the environment Δ.

Proposition 3.4 *(Contextuality)*

(i) *If* $(\Delta \vdash C) \xrightarrow{\gamma} (\Delta, \Delta' \vdash C')$ *then* $C^{\Delta}_{\gamma} \parallel C \Rightarrow \nu\Delta' \,.\, (m[\Delta'] \parallel l[\mathsf{true}] \parallel C')$

(ii) *If* $C^{\Delta}_{\gamma} \parallel C \Rightarrow l[\mathsf{true}] \parallel C''$ *then* $(\Delta \vdash C) \stackrel{\gamma}{\Longrightarrow} (\Delta, \Delta' \vdash C')$ *and* $C'' \xrightarrow{\beta}{}^{*} \nu\Delta' \,.\, (m[\Delta'] \parallel C')$.

Proof: Part (i) We proceed by induction on the derivation of $(\Delta \vdash C) \xrightarrow{\gamma} (\Delta, \Delta' \vdash C')$. Examples of the base case are:

- Suppose C is $n[n']$ and γ is $n \,.\, n'$ so that Δ' is empty and C' is C.
  We know that $C^{\Delta}_{n.n'}$ is $m[()] \parallel l[n' = \mathsf{join}\, n]$ and
  $$(m[()] \parallel l[\mathsf{join}\, n = n'] \parallel n[n']) \Rightarrow (m[()] \parallel l[n' = n'] \parallel n[n']) \Rightarrow (m[()] \parallel l[\mathsf{true}] \parallel n[n'])$$
  as required.

- Suppose C is $n[\lambda x \,.\, t]$ and γ is $n \,.\, @v \,.\, n'$ so that Δ' is $n' : \sigma'\,\mathsf{thread}$ (where $\Delta \vdash n : \sigma \to \sigma'\,\mathsf{thread}$) and C' is $C \parallel n'[\mathsf{let}\, x = v \,\mathsf{in}\, t]$. We know that C^{Δ}_{γ} is $\nu n' \,.\, m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{join}\, n\, v]$ and that
  $$\nu n' \,.\, m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{join}\, n\, v] \parallel n[\lambda x \,.\, t]$$
  reduces through join synchronisations to
  $$\nu n' \,.\, m[n'] \parallel l[(\lambda x \,.\, t); \mathsf{true}] \parallel n'[(\lambda x \,.\, t)\, v] \parallel n[\lambda x \,.\, t]$$
  which further reduces to
  $$\nu n' \,.\, m[n'] \parallel l[\mathsf{true}] \parallel n'[\mathsf{let}\, x = v \,\mathsf{in}\, t] \parallel n[\lambda x \,.\, t]$$
  as required.

For the inductive case we must consider transitions generated underneath parallel composition and name restriction. The former case follows easily from the inductive hypothesis so we concentrate on the latter.

- Firstly, suppose that the last rule used to derive the transition was
  $$\frac{(\Delta, n : \sigma \vdash C) \xrightarrow{\gamma} (\Delta', n : \sigma \vdash C')}{(\Delta \vdash \nu n : \sigma \,.\, C) \xrightarrow{\gamma} (\Delta' \vdash \nu n : \sigma \,.\, C')}[n \notin \gamma]$$
  then we know from the inductive hypothesis that $C^{\Delta, n:\sigma}_{\gamma} \parallel C \Rightarrow \nu\Delta' \,.\, (m[\Delta'] \parallel l[\mathsf{true}] \parallel C')$. Note that since we required l and m to be fresh, we have $l \neq n$ and $m \neq n$. It is not too hard to check that the induction and the fact that $n \notin \gamma$ implies that $C^{\Delta}_{\gamma} \parallel C \Rightarrow \nu\Delta' \,.\, (m[\Delta'] \parallel l[\mathsf{true}] \parallel C')$ also. From this we have:
  $$C^{\Delta}_{\gamma} \parallel \nu n \,.\, C \equiv \nu n \,.\, (C^{\Delta}_{\gamma} \parallel C) \Rightarrow \nu\Delta' \,.\, (m[\Delta'] \parallel l[\mathsf{true}] \parallel \nu n \,.\, C')$$
  as required.

- Now suppose that the last rule used was

$$\frac{(\Delta, n' : \sigma \vdash C) \xrightarrow{n.n'} (\Delta, n' : \sigma \vdash C')}{(\Delta \vdash \nu n' : \sigma \,.\, C) \xrightarrow{n.\nu n'} (\Delta, n' : \sigma \vdash C')}[n \neq n']$$

  We know by the induction hypothesis that

$$C^{\Delta, n'}_{n.n'} \parallel C \Rightarrow m[()] \parallel l[\mathsf{true}] \parallel C'$$

  that is,

$$m[()] \parallel l[n' = \mathsf{join}\, n] \parallel C \Rightarrow m[()] \parallel l[\mathsf{true}] \parallel C'$$

  which implies that $C' \equiv n[n'] \parallel C''$ for some C''. In this case,

$$C^{\Delta}_{n.\nu n'} \parallel \nu n' \,.\, C \Rightarrow \nu n' \,.\, (m[n'] \parallel l[n' \notin \Delta] \parallel n[n'] \parallel C'')$$

  and we clearly have that $n' \notin \Delta$ so this reduces as required.

Part (ii) We proceed by case analysis on γ. The reasoning is much the same in each of the cases so we demonstrate only two.

- If $\gamma = n \,.\, \nu n'$ (with $\Delta' = n'$) then, by hypothesis,

$$m[\mathsf{join}\, n] \parallel l[\mathsf{join}\, n \notin \Delta] \parallel C \Rightarrow l[\mathsf{true}] \parallel C''$$

  so we know by analysing the reduction rules, along with the fact that $fn(C) \subseteq \Delta$, that, for some C':

$$C'' \xrightarrow{\beta}{}^{*} \nu n' \,.\, (m[n'] \parallel n[n'] \parallel C').$$

  and so

$$(\Delta \vdash C) \Rightarrow (\Delta \vdash \nu n' \,.\, n[n'] \parallel C') \xrightarrow{n.\nu n'} (\Delta, n' \vdash n[n'] \parallel C')$$

  as required.

- If $\gamma = n \,.\, @v \,.\, n'$ (with $\Delta' = n'$) then, by hypothesis,

$$\nu n' \,.\, m[n'] \parallel l[\mathsf{join}\, n; \mathsf{true}] \parallel n'[\mathsf{join}\, n\, v] \parallel C \Rightarrow l[\mathsf{true}] \parallel C''$$

  then it must be the case, for some Δ'', C''', C'''', that

$$C \Rightarrow \nu \Delta'' \,.\, (n[\lambda x \,.\, t] \parallel C''') \qquad \nu \Delta'' \,.\, (n'[\mathsf{join}\, n\, v] \parallel n[\lambda x \,.\, t] \parallel C''') \Rightarrow C'''' \qquad \nu n' \,.\, (m[n'] \parallel C'''') \equiv C''$$

  So, since $\Delta \vdash v$,

$$\begin{array}{rcl} (\Delta \vdash C) & \Rightarrow & (\Delta \vdash \nu \Delta'' \,.\, (n[\lambda x \,.\, t] \parallel C''')) \\ & \xrightarrow{n.@v.n'} & (\Delta, n' \vdash \nu \Delta'' \,.\, (n[\lambda x \,.\, t] \parallel n'[\mathsf{let}\, x = v\, \mathsf{in}\, t] \parallel C''')) \\ & \xleftarrow{\beta}{}^{*} & (\Delta, n' \vdash \nu \Delta'' \,.\, (n[\lambda x \,.\, t] \parallel n'[\mathsf{join}\, n\, v] \parallel C''')) \\ & \Rightarrow & (\Delta, n' \vdash C'''') \end{array}$$

By confluence (Proposition 3.3), we can find C' such that

$$(\nu\Delta''.\,(n[\lambda x.\,t] \,\|\, n'[\mathsf{let}\, x = v \,\mathsf{in}\, t] \,\|\, C''')) \Rightarrow C' \overset{\beta}{\leftarrow}{}^{*}\, C''''.$$

Therefore

$$(\Delta \vdash C) \xrightarrow{n.@v.n'} (\Delta, n' \vdash C') \qquad C'' \overset{\beta}{\rightarrow}{}^{*}\, \nu n'.\,(m[n'] \,\|\, C')$$

as required. □

3.2 Bisimilarity

A *simulation* is a type-indexed relation on configurations $\mathcal{R}$ such that the following diagram can be completed:

$$\begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftrightarrow} & (\Delta \vdash C_2) \\ {\scriptstyle\alpha}\downarrow & & \\ (\Delta' \vdash C_1') & & \end{array} \quad \text{as} \quad \begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftrightarrow} & (\Delta \vdash C_2) \\ {\scriptstyle\alpha}\downarrow & & \Downarrow{\scriptstyle\hat{\alpha}} \\ (\Delta' \vdash C_1') & \overset{\mathcal{R}}{\longleftrightarrow} & (\Delta' \vdash C_2') \end{array}$$

A *bisimulation* is a simulation whose inverse is also a simulation. Let bisimilarity, $\approx$, denote the largest bisimulation between configurations.

We now state a proof principle which we use heavily, namely weak bisimulation up to β-reduction. We say that a type-indexed relation $\mathcal{R}$ is a simulation up to $(\overset{\beta}{\rightarrow}{}^{*}, \approx)$ if we can complete the diagram:

$$\begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftrightarrow} & (\Delta \vdash C_2) \\ {\scriptstyle\alpha}\downarrow & & \\ (\Delta' \vdash C_1') & & \end{array} \quad \text{as} \quad \begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftarrow\!\!\!\longrightarrow} & (\Delta \vdash C_2) \\ {\scriptstyle\alpha}\downarrow & & \Downarrow{\scriptstyle\hat{\alpha}} \\ (\Delta' \vdash C_1') & \overset{\beta}{\rightarrow}{}^{*}\,\mathcal{R}\,\approx & (\Delta' \vdash C_2') \end{array}$$

As before we say that $\mathcal{R}$ is a bisimulation up to $(\overset{\beta}{\rightarrow}{}^{*}, \approx)$ if both $\mathcal{R}$ and its inverse are simulations up to $(\overset{\beta}{\rightarrow}{}^{*}, \approx)$. The proof principle we appeal to is

Proposition 3.5 *If $\mathcal{R}$ is a bisimulation up to $(\overset{\beta}{\rightarrow}{}^{*}, \approx)$ then $\approx\mathcal{R}\approx$ is a bisimulation.*

Proof: We use the fact that $\mathcal{R}$ is a bisimulation up to $(\overset{\beta}{\rightarrow}{}^{*}, \approx)$ and confluence of β-reduction, Proposition 3.3, to show that we can complete the diagram:

$$\begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftrightarrow} & (\Delta \vdash C_2) \\ \Downarrow{\scriptstyle\hat{\alpha}} & & \\ (\Delta' \vdash C_1') & & \end{array} \quad \text{as} \quad \begin{array}{ccc} (\Delta \vdash C_1) & \overset{\mathcal{R}}{\longleftarrow\!\!\!\longrightarrow} & (\Delta \vdash C_2) \\ \Downarrow{\scriptstyle\hat{\alpha}} & & \Downarrow{\scriptstyle\hat{\alpha}} \\ (\Delta' \vdash C_1') & \overset{\beta}{\rightarrow}{}^{*}\,\mathcal{R}\,\approx & (\Delta' \vdash C_2') \end{array}$$

Given this it is straightforward to demonstrate that $\approx\mathcal{R}\approx$ is a bisimulation by again using confluence to observe that $\overset{\beta}{\rightarrow}{}^{*}$ is contained in $\approx$. □

3.3 Full abstraction

We will now show that bisimilarity for $\mu\nu$CML coincides with barbed equivalence. Soundness follows immediately once we have that bisimilarity is a congruence: this is the subject of the next section.

Proposition 3.6 (Bisimilarity is sound for barbed equivalence) $\Delta \vDash C_1 \approx C_2$ *implies* $\Delta \vDash C_1 \approx^b C_2$

Proof: It is easy to show that $\approx$ is barbed and reduction-closed, and Theorem 4.9 shows that $\approx$ is a congruence. Hence, $\approx$ implies $\approx^b$. □

Having proved soundness, to obtain full abstraction we must prove its converse: completeness. We will actually show a slightly stronger completeness result though. Bisimilarity is complete for a barbed equivalence in which the only contexts used are parallel compositions with some configuration. A binary relation $\mathcal{R}$ on configurations is $\|$-*contextual* if it satisfies:

$$\Delta \vDash C_1 \mathcal{R} C_2 \quad \text{implies} \quad \Delta, \Delta' \vDash C_1 \| C \mathcal{R} C_2 \| C$$

for all $\Delta, \Delta' \vdash C$. Then, $\|$-*barbed equivalence*, $\approx^{pb}$ is defined to be the largest reduction-closed barbed $\|$-contextual relation.

It is immediate that $\approx^b$ is contained in $\approx^{pb}$, so it suffices to show that $\approx$ is complete for $\approx^{pb}$. First though we present some useful technical lemmas.

Lemma 3.7 (Garbage collection) *If* $\Delta, n \vDash n[v] \| C_1 \approx^{pb} n[v] \| C_2$ *and* $n \notin C_1, C_2$ *then* $\Delta \vDash C_1 \approx^{pb} C_2$.

Proof: Straightforward. □

Lemma 3.8 (Extrusion) *If*

$$\Delta, m \vDash \nu\Delta' . (m[\Delta'] \| C_1) \approx^{pb} \nu\Delta' . (m[\Delta'] \| C_2)$$

and $m \notin C_1, C_2$ *then* $\Delta, \Delta' \vDash C_1 \approx^{pb} C_2$.

Proof: We prove this by coinduction. Define $\mathcal{R}$ as:

$$\frac{\Delta, m \vDash \nu\Delta' . (m[\Delta'] \| C_1) \approx^{pb} \nu\Delta' . (m[\Delta'] \| C_2) \quad \Delta, \Delta', m \text{ disjoint}}{\Delta, \Delta' \vDash C_1 \mathcal{R} C_2}$$

We need to demonstrate that $\mathcal{R}$ is barbed, reduction closed and $\|$-contextual. Because $\approx^{pb}$ is the largest such relation, we would then know that $\mathcal{R} \subseteq \approx^{pb}$, thus achieving our result.

- Reduction closure for $\mathcal{R}$ is immediate from the definition.
- To show that $\mathcal{R}$ is barbed closed we suppose $C_1 \Downarrow_n$ for some n. If n is defined in Δ then by assumption we find $C_2 \Downarrow_n$. The more difficult possibility is that n is defined in Δ'. In this case we use parallel contextuality of $\approx^{pb}$ to help. Choose a fresh n' and define C to be

  $$n'[\mathsf{let}\, x = \pi_n(\mathsf{join}\, m)\, \mathsf{in}\, \mathsf{join}\, x]$$

  where π_n refers to the projected component of Δ' at which n occurs. This configuration fetches the private names exported at m and uses the particular name n to synchronise on. We observe that $C_i \Downarrow_n$ if and only if $\nu\Delta' . (m[\Delta'] \| C_i) \| C \Downarrow_{n'}$ (for $i = 1, 2$). We know that C is well-typed so, by hypothesis and $\|$-contextuality, it is easy to see that $C_1 \Downarrow_n$ if and only if $C_2 \Downarrow_n$.

- In order to demonstrate that $\mathcal{R}$ is $\|$-contextual we take $\Delta, \Delta' \vDash C_1 \mathcal{R} C_2$ and some $\Delta, \Delta', \Delta'' \vdash C$ and show that

$$\Delta, \Delta', \Delta'' \vDash C_1 \| C \mathcal{R} C_2 \| C$$

This can be achieved by building a configuration C' based on C but typable in the environment Δ, Δ''. Proceed by noticing that C may be expressed, up to structural equivalence, as

$$\nu\Delta''' . \prod_i n_i[t_i]$$

we define C' to be

$$\nu\Delta''' . \prod_i n_i[\mathsf{let}\,\Delta' = \mathsf{join}\,m\,\mathsf{in}\,t_i]$$

where we write $\mathsf{let}\,\Delta = t_1\,\mathsf{in}\,t_2$ as syntax sugar, defined:

$$\mathsf{let}\,(n_1 : \sigma_1, \ldots, n_k : \sigma_k) = t_1\,\mathsf{in}\,t_2 = \mathsf{let}\,(x_1 : \sigma_1, \ldots, x_k : \sigma_k) = t_1\,\mathsf{in}\,t_2[x_1/n_1, \ldots, x_k/n_k].$$

Note that

$$m[\Delta'] \| C' \xrightarrow{\beta}{}^{*} m[\Delta'] \| C.$$

So, by $\|$-contextuality we know that

$$\Delta, \Delta'', m \vDash \nu\Delta' . (m[\Delta'] \| C_1) \| C' \approx^{pb} \nu\Delta' . (m[\Delta'] \| C_2) \| C'.$$

Since $\xrightarrow{\beta}{}^{*}$ is contained in $\approx^{pb}$ we have

$$\Delta, \Delta'', m \vDash \nu\Delta' . (m[\Delta'] \| C_1 \| C) \approx^{pb} \nu\Delta' . (m[\Delta'] \| C_2 \| C),$$

and so

$$\Delta, \Delta', \Delta'' \vDash C_1 \| C \mathcal{R} C_2 \| C$$

as required. □

Proposition 3.9 (Bisimilarity is complete for barbed equivalence) $\Delta \vDash C_1 \approx^b C_2$ *implies* $\Delta \vDash C_1 \approx C_2$

Proof: We notice immediately that $\approx^b \subseteq \approx^{pb}$ so, by coinduction, it suffices to show that $\approx^{pb}$ forms a bisimulation. So, suppose that $\Delta \vDash C_1 \approx^{pb} C_2$ and that $(\Delta \vdash C_1) \xrightarrow{\alpha} (\Delta, \Delta' \vdash C_1')$. If α is β or τ then we immediately have a match as $\approx^{pb}$ is reduction closed. Otherwise, we know by Proposition 3.4, that for fresh l, m there exists a term $\Delta, l, m \vdash C_\alpha^\Delta$ with the appropriate properties. From this we choose a further fresh name, n and build a new configuration C as:

$$C_\alpha^\Delta \| n[\mathsf{if}\,(\mathsf{join}\,l)\,\mathsf{then}\,(\mathsf{true} \oplus \mathsf{false})\,\mathsf{else}\,\mathsf{true}]$$

where $\oplus$ is syntactic sugar for a suitable encoding of an internal choice operator with reductions

$$n[\mathsf{let}\,x = (v_1 \oplus v_2)\,\mathsf{in}\,t] \Rightarrow n[\mathsf{let}\,x = v_i\,\mathsf{in}\,t] \| \mathsf{G}$$

(for $i = 1, 2$), where G is a 'garbage' configuration bisimilar to $\mathbf{0}$. Now, it should be easy to see that

$$C_1 \| C \Rightarrow C_1''' \quad \text{where} \quad C_1''' \equiv \nu\Delta' . (m[\Delta'] \| l[\mathsf{true}] \| n[\mathsf{false}] \| \mathsf{G} \| C_1').$$

We know that $\approx^{pb}$ is reduction closed and $\|$-contextual so we can find some C_2''' such that

$$C_2 \,\|\, C \Rightarrow C_2''' \quad \text{and} \quad \Delta, l, m, n \vDash C_1''' \approx^{pb} C_2'''.$$

Since $C_1''' \Downarrow_l$ and $C_1''' \not\Downarrow_n$ we know that $C_2''' \Downarrow_l$ and $C_2''' \not\Downarrow_n$. From this and the definition of C we have

$$C_2''' \equiv C_2'' \,\|\, l[\mathsf{true}] \,\|\, n[\mathsf{false}] \,\|\, \mathsf{G} \quad \text{and} \quad C_2 \,\|\, C_\alpha^\Delta \Rightarrow C_2'' \,\|\, l[\mathsf{true}]$$

By Proposition 3.4 we find some C_2' such that

$$(\Delta \vdash C_2) \xrightarrow{\alpha} (\Delta, \Delta' \vdash C_2') \quad \text{and} \quad C_2'' \xrightarrow{\beta}{}^{*} \nu\Delta' \,.\, (C_2' \,\|\, m[\Delta']).$$

We know $C_2''' \xrightarrow{\beta}{}^{*} \nu\Delta' \,.\, (m[\Delta'] \,\|\, l[\mathsf{true}] \,\|\, n[\mathsf{false}] \,\|\, C_2')$ and, because $\xrightarrow{\beta}{}^{*}$ is contained in $\approx^{pb}$, we know that

$$\nu\Delta' \,.\, (m[\Delta'] \,\|\, l[\mathsf{true}] \,\|\, n[\mathsf{false}] \,\|\, \mathsf{G} \,\|\, C_1') \equiv C_1''' \approx^{pb} C_2''' \approx^{pb} \nu\Delta' \,.\, (m[\Delta'] \,\|\, l[\mathsf{true}] \,\|\, n[\mathsf{false}] \,\|\, \mathsf{G} \,\|\, C_2').$$

So, two uses of Lemma 3.7 and one use of Lemma 3.8 gives us

$$\Delta, \Delta' \vDash C_1' \approx^{pb} C_2'$$

as required. □

Theorem 3.10 (Full abstraction) $\Delta \vDash C_1 \approx^{b} C_2$ *if and only if* $\Delta \vDash C_1 \approx C_2$

Proof: Follows from Propositions 3.6 and 3.9. □

4 Congruence properties of bisimilarity

We are left with the task of showing that bisimilarity is a congruence. This is a notoriously difficult problem, and proof techniques which work in the presence of both higher-order features and dynamically generated names are limited [24, 25].

A viable approach to tackling this problem in languages with sufficient power is to represent higher-order computation by first-order means. Indeed, Sangiorgi demonstrates in his thesis [29] that higher-order π-calculus can be encoded, fully abstractly, in the first-order π-calculus by means of reference passing—this transformation is described in two stages, the first of which is known as a trigger encoding and recasts higher-order π-calculus in a sublanguage of itself in which only canonical higher-order values, or triggers, are passed.

We adopt a similar approach here but, owing to the functional nature of the language, our encoding is more complicated than that of the higher-order π-calculus. This is simply because processes in languages such as π-calculus do not compute and return values in the way that functions do. Thus, if one were to encode the evaluation of a function in some context by actually evaluating the function out of context, then the resulting value would eventually need to be replaced in that context. This situation does not arise in the π-calculus. The thrust of the current work is to demonstrate a novel approach to proving a fully abstract trigger encoding which can be used to prove congruence of bisimilarity in higher-order languages.

Rather than compositionally translating our higher-order language into a simpler language, we describe an alternative operational semantics which implements this trigger passing. The intention is that there is a direct proof of congruence of bisimilarity on this alternative operational semantics, and correctness between the two semantics yields congruence on the original. Correctness between the two semantics can be stated quite tightly as:

$$[\![C]\!]_\omega \approx [\![C]\!]_0$$

where $[\![C]\!]_\omega$ is understood to be the interpretation of C using the original semantics and $[\![C]\!]_0$ the triggered semantics.

In fact, to relieve the difficulty of proving correctness we aim to use an induction on the order of the type of C. This leads us to defining a hierarchy of semantics, indexed by type order. In $[\![C]\!]_n$, terms of type lower than n are passed directly, and terms of higher type are trigger-encoded. We can then regard $[\![C]\!]_\omega$ as the 'limit' of the semantics $[\![C]\!]_0, [\![C]\!]_1, \ldots$ Our proof that bisimilarity is a congruence is then broken into three parts:

1. Prove that bisimilarity is a congruence for $[\![\cdot]\!]_0$.
2. Prove that if $[\![C]\!]_0 \approx [\![C']\!]_i$ for all i then $[\![C]\!]_0 \approx [\![C']\!]_\omega$.
3. Prove for all i that $[\![C]\!]_i \approx [\![C]\!]_{i+1}$.

From these three properties, it is easy to prove that bisimulation is a congruence for $[\![\cdot]\!]_\omega$, which is, by definition, exactly our original semantics for $\mu\nu$CML.

Note that this proof relies on a well-founded order on types, and so will not work in the presence of general recursive types. This is not as limiting as might first be thought, since $\sigma\,\mathsf{chan}$ and $\sigma\,\mathsf{thread}$ are considered to be order 0 no matter the order of σ, and so we can deal with any recursive type as long as the recursive type variable is beneath a $\cdot\,\mathsf{chan}$ or $\cdot\,\mathsf{thread}$. This is a similar situation as for most imperative languages, which restrict recursive types to those including pointers. Also note that this restriction is weak enough to include all of the π-calculus sorts, such as the type $\mu X\,.\,X\,\mathsf{chan}$ which describes monomorphic π-calculus channels.

We will now present the triggered semantics and show the three required properties.

4.1 Trigger Semantics for $\mu\nu$CML

In order to describe these semantics concisely it will be helpful to introduce a mild language extension. There are no explicit recursive function definitions in the core language we presented above as such terms can be programmed using the thread synchronization primitives (*cf.* coding the Y-combinator using general references). We introduce a replicated reception primitive, which can indeed be coded using recursive functions, or more directly, with join synchronization. Let us write $*\mathsf{recv}\,n$ to represent this new expression. There is an associated reduction rule for this new expression which behaves as a recv expression but spawns a new thread of evaluation. This is defined as

$$n_1[\mathsf{let}\,x_1 = \mathsf{send}\,(n,v)\,\mathsf{in}\,t_1] \,||\, n_2[\mathsf{let}\,x_2 = *\mathsf{recv}\,n\,\mathsf{in}\,t_2] \xrightarrow{\tau} \nu n_3\,.\left(\begin{array}{c} n_1[\mathsf{let}\,x_1 = ()\,\mathsf{in}\,t_1] || n_2[\mathsf{let}\,x_2 = v\,\mathsf{in}\,t_2] || \\ n_3[\mathsf{let}\,x_2 = *\mathsf{recv}\,n\,\mathsf{in}\,t_2] \end{array}\right)$$

Of course, there is an obvious corresponding transition rule for replicated reception also:

$$(\Delta \vdash n[\mathsf{let}\, x = *\mathsf{recv}\, n' \,\mathsf{in}\, t]) \xrightarrow{\mathsf{recv}(n',v)} (\Delta \vdash \nu n''\,.\, n[\mathsf{let}\, x = v \,\mathsf{in}\, t] \parallel n''[\mathsf{let}\, x = *\mathsf{recv}\, n \,\mathsf{in}\, t])$$

The following pieces of notation will be convenient. Let τ_a denote the term

$$\lambda x\,.\,\mathsf{let}\, r = \mathsf{chan}\,() \,\mathsf{in}\, \mathsf{send}\,(a,(x,r)); \mathsf{recv}\, r$$

The *trigger call* τ_a is used to substitute through terms in place of functions. When the trigger call is applied to an argument, the trigger simply sends the argument off to the actual function (on channel a). It must also wait for the resulting value given by the application on a freshly created private channel. Complementary to this is the *resource* at a, written $a \Leftarrow f$, where we use f to range over λ-abstractions. This is defined to be a replicated receive command:

$$\mathsf{let}\,(x_1,x_2) = *\mathsf{recv}\, a \,\mathsf{in}\, \mathsf{let}\, z = f\, x_1 \,\mathsf{in}\, \mathsf{send}\,(x_2,z)$$

which can continually receive arguments to f, along with a reply channel. It then applies f to the argument and sends the result back along the reply channel.

These are the two basic components of the triggered semantics. We use them to define a notion of type-indexed substitution. Define the order of a type $O(\sigma)$ as:

$$\begin{aligned} O(B) = O(\sigma\,\mathsf{chan}) = O(\sigma\,\mathsf{thread}) &= 0 \\ O(\sigma_1 * \sigma_2) &= \max(O(\sigma_1), O(\sigma_2)) \\ O(\sigma_1 \to \sigma_2) &= \max(O(\sigma_1)+1, O(\sigma_2)) \end{aligned}$$

and the type-order of a closed term $O(t)$ is the order of the term's type. Strictly speaking, this ought to be defined relative to the name environment Δ in which t is typed but the type-order of names is always 0 so we may safely omit this here. Let the level i substitution $[v/x]_i$ be defined by:

$$\begin{aligned} C[b/x]_i &= C[b/x] \\ C[n/x]_i &= C[n/x] \\ C[(v_1,v_2)/x]_i &= (C[(x_1,x_2)/x])[v_1/x_1]_i[v_2/x_2]_i \\ C[f/x]_i &= \begin{cases} C[f/x] \text{ if } O(f) \le i \\ \nu a,n\,.\,(C[\tau_a/x] \parallel n[a \Leftarrow f]) \text{ otherwise} \end{cases} \end{aligned}$$

From the definition of level i substitution, we can now define the level i trigger semantics $[\![\cdot]\!]_i$ by replacing the β-reduction rule for let expressions with

$$n[\mathsf{let}\, x = v \,\mathsf{in}\, t] \xrightarrow{\beta} n[t][v/x]_i$$

and leaving all other rules unchanged.

We now state some useful lemmas concerning the trigger protocol semantics. Firstly, we need to refer to terms which may only contain a particular name n as the argument to a send command. Moreover, this property needs to be maintained as an invariant under transitions. To this end define n is a send-channel in C whenever every free occurrence of n in C is of the form $\mathsf{send}\,(n,v)$ for some v.

The following properties of this relation are easy to verify by structural induction.

Lemma 4.1

(i) *a is a* send *-channel in* τ_a

(ii) *If a is a* send *-channel in f and $a \neq a'$ then a is a* send *-channel in* $a' \Leftarrow f$.

(iii) *If n is a* send *-channel in t and n is a* send *-channel in v then n is a* send *-channel in* $t[v/x]$.

(iv) *If n is a* send *-channel in C and* $(\Delta \vdash C) \xrightarrow{\alpha} (\Delta' \vdash C')$ *(in any level semantics) with* $n \notin \alpha$ *then n is a* send *-channel in* C'.

Lemma 4.2 *Any reductions which are instances of the following are confluent with all other transitions and we label them* β *to reflect this:*

$$\nu r.\,(n[\mathsf{let}\, x = \mathsf{recv}\, r\, \mathsf{in}\, t] \parallel n'[\mathsf{let}\, x' = \mathsf{send}\,(r,v)\, \mathsf{in}\, t']) \xrightarrow{\beta} \nu r.\,(n[\mathsf{let}\, x = v\, \mathsf{in}\, t] \parallel n'[\mathsf{let}\, x' = ()\, \mathsf{in}\, t'])$$

and

$$\nu n\,.\,(C \parallel n_1[\mathsf{let}\, x_1 = \mathsf{send}\,(n,v)\, \mathsf{in}\, t_1] \parallel n_2[\mathsf{let}\, x_2 = *\mathsf{recv}\, n\, \mathsf{in}\, t_2]) \xrightarrow{\beta} \nu n n_3\,.\left(\begin{array}{c} C \parallel n_1[\mathsf{let}\, x_1 = ()\, \mathsf{in}\, t_1] \parallel \\ n_3[\mathsf{let}\, x_2 = *\mathsf{recv}\, n\, \mathsf{in}\, t_2] \parallel n_2[\mathsf{let}\, x_2 = v\, \mathsf{in}\, t_2] \end{array}\right)$$

providing n is a send *-channel in C.*

The first of these observes that channels which have unique points of communication give confluent reduction because no competition between resources occurs. This is used for the return part of the trigger protocol. The latter is slightly more involved and relies upon a side-condition that the sending participant cannot communicate with any party other than the replicated input. We have this property when beginning each trigger protocol communication and the lemmas above show that we can maintain it as an invariant throughout testing. There are a series of technical lemmas we must work through before we can show correctness of the triggered semantics. The first of these serves to demonstrate that we can remove, up to weak bisimulation, unwanted η-expansions introduced by the trigger protocol. Note that because we may not assume congruence of bisimulation at this point we must state these lemmas in context.

Lemma 4.3

(i) $\Delta \vDash [\![\nu\Delta'\,.\,C \parallel n[t]]\!]_i \approx [\![\nu\Delta'\,.\,C \parallel n[\mathsf{let}\, x = t\, \mathsf{in}\, x]]\!]_i$

(ii) $\Delta \vDash [\![\nu\Delta'\,.\,C \parallel n[\mathsf{let}\, x_1 = t_1\, \mathsf{in}\, t_2]]\!]_i \approx [\![\nu\Delta'\,.\,C \parallel n[\mathsf{let}\, x_2 = t_1\, \mathsf{in}\, \mathsf{let}\, x_1 = x_2\, \mathsf{in}\, t_2]]\!]_i$

Proof: This is easy to prove using bisimulations. The only point to watch is the case in which t (or t_1) is itself a let expression. To accommodate this we must build the witness, for (i) say, as

$$\Delta \vDash \nu\Delta'\,.\,C \parallel n[\mathsf{let}\, \vec{x} = \vec{t}\, \mathsf{in}\, t]\ \mathcal{R}\ \nu\Delta'\,.\,C \parallel n[\mathsf{let}\, \vec{x} = \vec{t}\, \mathsf{in}\, \mathsf{let}\, x = t\, \mathsf{in}\, x]$$

where $\mathsf{let}\, \vec{x} = \vec{t}\, \mathsf{in}\, t$ refers to the nested sequence $\mathsf{let}\, x_1 = t_1\, \mathsf{in}\, \mathsf{let}\, x_2 = t_2\, \mathsf{in}\, \ldots \mathsf{let}\, x_n = t_n\, \mathsf{in}\, t$. □

The next lemma is used to establish the correctness of the return end of the trigger protocol.

Lemma 4.4 *When n', r do not occur free in t_1, we have:*

$$\Delta \vDash [\![\nu\Delta' . C \parallel \nu n' r . (n[\mathsf{let}\, x = \mathsf{recv}\, r\, \mathsf{in}\, t_2] \parallel n'[\mathsf{let}\, z = t_1\, \mathsf{in}\, \mathsf{send}\,(r,z)])]\!]_i \approx [\![\nu\Delta' . C \parallel n[\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2]]\!]_i$$

Proof: We use the bisimulation up to $(\xrightarrow{\beta}{}^*, \approx)$ technique here (note that the proof of Proposition 3.5 depended only on confluence of β-reduction with all other transitions, and so holds for the level *i* trigger semantics). The witness must use a stack of evaluation contexts in a similar manner to the previous lemma. Specifically we let $\mathcal{R}$ contain $\approx$ along with pairs of configurations formed from

$$\Delta \vdash \nu\Delta' . C \parallel \nu n' r . (n[\mathsf{let}\, x = \mathsf{recv}\, r\, \mathsf{in}\, t_2] \parallel n'[\mathsf{let}\, \vec{x} = \vec{t}\, \mathsf{in}\, \mathsf{let}\, z = t_1\, \mathsf{in}\, \mathsf{send}\,(r,z)])$$

and

$$\Delta \vdash \nu\Delta' . C \parallel n[\mathsf{let}\, \vec{x} = \vec{t}\, \mathsf{in}\, \mathsf{let}\, x = t_1\, \mathsf{in}\, t_2]$$

and show that $\mathcal{R}$ has the relevant closure properties. This is more or less straightforward checking. The only points of interest occur in the situation when the evaluation stack is empty and t_1 is a value. In this case the communication on r occurs. We note that, because r is private, this communication is an instance of the first special β-reduction of Lemma 4.2 and hence can be used in the up to $\xrightarrow{\beta}{}^*$ technique. Also, the residual of this communication will leave a terminated private thread at n' and a new name declaration for r which is no longer used. We note that these can easily be garbage collected using weak bisimulation. □

The next lemma is the heart of the correctness proof. This essentially states that, for substitutions of functions of type order *i* the trigger protocol correctly implements the substitution. In order to see this we show that the relation $\{C[v/x]_i, C[v/x]_{i+1}\}$ is a bisimulation up to $(\xrightarrow{\beta}{}^*, \approx)$. The difficulty here is seen in the case in which v is a function of order $i+1$, being applied to some argument in C. On the right hand side we have a standard substitution and a standard β-reduction. Whereas on the left hand side we see a triggered substitution, and, by virtue of the argument being of type $< i+1$, a standard β-reduction. It is crucial that no nested trigger substitution is incurred here and we can use the power of the up to technique to finish by appealing to the previous lemma.

Lemma 4.5 $[\![C[v/x]_i]\!]_{i'} \approx [\![C[v/x]_{i+1}]\!]_{i'}$ *for all* $i' \geq i$.

Proof: We begin by induction on the structure of the value to be substituted. For a base case we notice that v must be a constant, with type order 0. This means that the substitution at any i is not triggered and there is nothing to prove. We can use the inductive hypothesis twice to establish the result in the case where v is a pair of values. This follows easily from the definition and simple properties of substitution at level i.

The difficult case occurs when v is an abstraction, $\lambda x . t$, say. We notice that if $O(v) \neq i+1$ then $C[v/x]_i = C[v/x]_{i+1}$ so we may assume that $O(v) = i+1$. Now, we proceed using bisimulation up to $(\xrightarrow{\beta}{}^*, \approx)$ (working in the level i' semantics) to show the result. We let $\mathcal{R}$ contain bisimilarity and let it relate the configurations $\Delta \vdash C[f/x]_i$ and $\Delta \vdash C[f/x]_{i+1}$ where f has type order $i+1$. To check that $\mathcal{R}$

forms a bisimulation, suppose we take a pair of configurations related by $\mathcal{R}$. Either these are bisimilar, hence satisfy the required closure property, or we have

$$\Delta \vDash C[f/x]_i \; \mathcal{R} \; C[f/x]_{i+1}$$

We notice straight away that the right-hand expression contains a standard substitution, whereas the left-hand expression contains a triggered substitution. Suppose then that $(\Delta \vdash C[f/x]_i) \xrightarrow{\alpha} (\Delta' \vdash C')$. We must find a matching transition. By inspecting the transition rules it is apparent that the only transitions which are affected by the change in substitution are those which destroy λ-terms, namely, those in which α is $n\,.\,@v_1\,.\,n'$ and instances of the β-reduction

$$n[\mathsf{let}\, x = (\lambda x_1 t_1)\, v_1 \,\mathsf{in}\, t] \xrightarrow{\beta} n[\mathsf{let}\, x = (\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, t_1)\,\mathsf{in}\, t]$$

we consider these in turn. Firstly, suppose that C is of the form

$$\nu\Delta_0\,.\,(C_1 \parallel n[x])[\lambda x_1\,.\,t_1/x]_i$$

that is to say, that this is a triggered substitution:

$$\nu(\Delta_0, a, n_1)\,.\,(C_1[\tau_a/x] \parallel n[\tau_a] \parallel n_1[a \Leftarrow \lambda x_1\,.\,t_1])$$

so that, after a $n\,.\,@v_1\,.\,n'$ transition, a configuration

$$\Delta, n' \vdash \nu(\Delta_0, a, n_1)\,.\,(C_1[\tau_a/x] \parallel n[\tau_a] \parallel n'[\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, \mathcal{B}(\tau_a)] \parallel n_1[a \Leftarrow \lambda x_1\,.\,t_1]) \equiv C'$$

is reached where $\mathcal{B}(\tau_a)$ refers to the body of the abstraction defining the trigger call. Noting that $O(v_1) < i+1 \leq i'$, we can observe a sequence of β-reductions in which the value is passed (and not triggered) to reach the configuration

$$\Delta, n' \vdash \nu\Delta_0\,.\,(C_1 \parallel n[x] \parallel \nu r, n''\,.\,n'[\mathsf{let}\, x_1 = \mathsf{recv}\, r \,\mathsf{in}\, x_1] \parallel n''[\mathsf{let}\, x_2 = (\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, t_1)\,\mathsf{in}\,\mathsf{send}\,(r, x_2)])[\lambda x_1\,.\,t_1/x]_i$$

It is this configuration, call it C_A, that we will find our match in $\mathcal{R}$ for. Note that these β-reductions include the second special β-reduction of Lemma 4.2 as we know a is a send-channel in $C_1[\tau_a/x]$ (closed under reduction).

To find our match we consider the corresponding C under the level $i+1$ substitution. This is of course a standard substitution so we may observe an $n\,.\,@v_1\,.\,n'$ transition from

$$\Delta \vdash \nu\Delta_0\,.\,(C_1 \parallel n[x])[\lambda x_1\,.\,t_1/x]_{i+1}$$

to

$$\Delta, n' \vdash \nu\Delta_0\,.\,(C_1 \parallel n[x] \parallel n'[\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, t_1])[\lambda x_1\,.\,t_1/x]_{i+1}$$

Given this, call it C_B, we then apply Lemmas 4.3 part (i) and 4.4 to see that C_B is in fact weakly bisimilar to

$$\nu\Delta_0\,.\,(C_1 \parallel n[x] \parallel \nu r, n''\,.\,n'[\mathsf{let}\, x_1 = \mathsf{recv}\, r \,\mathsf{in}\, x_1] \parallel n''[\mathsf{let}\, x_2 = (\mathsf{let}\, x_1 = v_1 \,\mathsf{in}\, t_1)\,\mathsf{in}\,\mathsf{send}\,(r, x_2)])[\lambda x_1\,.\,t_1/x]_{i+1}$$

which is just C_A with a level $i+1$ substitution in place of the level i substitution. Thus we have

$$C' \xrightarrow{\beta}{}^{*} C_A \mathcal{R} \approx C_B$$

with $(\Delta \vdash C[\lambda x_1 \,.\, t_1/x]_{i+1}) \xrightarrow{n.@v.n'} (\Delta, n' \vdash C_B)$ as required.

Similar reasoning can be applied in the case of α being an instance of the β-reduction mentioned above. However, in this case Lemma 4.3 part (ii), is required.

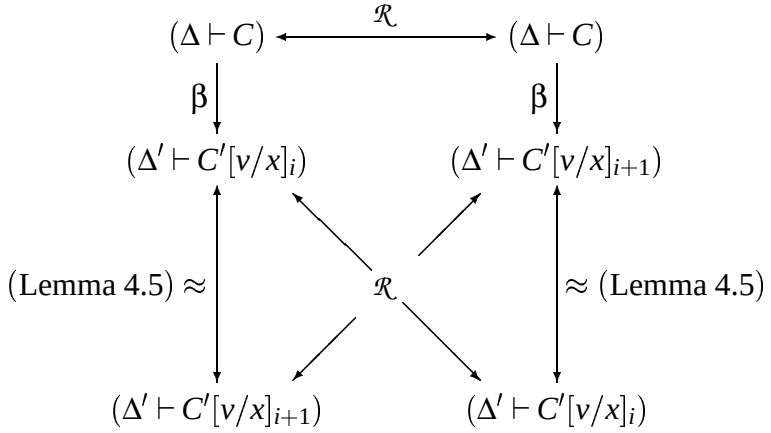
We must also demonstrate that $\mathcal{R}^{-1}$ is a simulation. This is done in exactly the same manner. □

Proposition 4.6

(i) $[\![C]\!]_i \approx [\![C]\!]_{i+1}$ *for all i.*

(ii) If $[\![C]\!]_0 \approx [\![C']\!]_i$ *for all i then* $[\![C]\!]_0 \approx [\![C']\!]_\omega$.

Proof: Part (i) is easy to show using a bisimulation. The witness for the bisimulation is simply the identity relation between terms in the i and $i+1$ level semantics, that is $\Delta \vDash [\![C]\!]_i \; \mathcal{R} \; [\![C]\!]_{i+1}$. To show that this is a bisimulation we must show that any transitions of $\Delta \vdash C$ in the level i semantics can be matched in the $i+1$ semantics (and vice-versa). The only transitions which behave differently between these semantics are of course β-reductions which are instances of the let reduction rule. All other transitions are simply matched with an identical transition to an identical term. We use Lemma 4.5 (with i' instantiated to i and $i+1$) to provide us with an up to weak bisimulation match for these β-reductions, which are best depicted in the following diagram. For reasons of clarity we have omitted the index of the semantics being used, however all terms on the left are to be considered in the level i semantics whilst all terms on the right in the level $i+1$.



For Part (ii) we construct a bisimulation:

$$\mathcal{R} = \{(\Delta \vdash [\![C_1]\!]_0, \Delta \vdash [\![C_2]\!]_\omega) \mid \exists i \,.\, \forall i' > i \,.\, \Delta \vDash [\![C_1]\!]_0 \approx [\![C_2]\!]_{i'}\}$$

We must show that this is actually a bisimulation. Suppose that $\Delta \vDash C_1 \; \mathcal{R} \; C_2$ and $(\Delta \vdash C_1) \xrightarrow{\alpha} (\Delta' \vdash C_1')$. We know that there must exist some i_0 such that for all $j > i_0$ we have some

$$(\Delta \vdash C_2) \xRightarrow{\hat{\alpha}} (\Delta' \vdash C_2'^{j})$$

in the level j semantics with $\Delta' \vDash C_1' \approx C_2'^j$ for terms between the level 0 and level j semantics. In particular we can choose j to be greater than i_0 and the highest type order appearing in the type derivation tree of C_2 and the highest type order appearing in the type derivations of any values appearing in α. We know that, by definition, any substitutions performed in the transitions $(\Delta \vdash C_2) \stackrel{\hat{\alpha}}{\Longrightarrow} (\Delta' \vdash C_2'^j)$ are not triggered. Therefore we also have $(\Delta \vdash C_2) \stackrel{\hat{\alpha}}{=\!\Rightarrow} (\Delta' \vdash C_2'^j)$ in the level ω semantics. Moreover, we know from part (i) that $\Delta' \vDash [\![C_2'^j]\!]_j \approx [\![C_2'^j]\!]_{i'}$ for any $i' > j$. Hence, $\Delta' \vDash C_1' \mathcal{R} C_2'^j$ as required.

The transitions from $\Delta \vdash C_2$ can be matched similarly. □

Corollary 4.7 $[\![C]\!]_0 \approx [\![C]\!]_\omega$ *for all* C.

4.2 Congruence

We have described how, in order to verify congruence of bisimulation equivalence for the standard semantics, it is sufficient to verify congruence of bisimulation equivalence for the completely triggered, level 0, semantics. We show this latter result now.

Proposition 4.8 *For all contexts* $\Delta' \vdash \mathcal{C}[\cdot_\Delta]$,

$$\textit{If } \Delta \vDash [\![C_1]\!]_0 \approx [\![C_2]\!]_0 \textit{ then } \Delta' \vDash [\![\mathcal{C}[C_1]]\!]_0 \approx [\![\mathcal{C}[C_2]]\!]_0$$

Proof: This can now be proved fairly directly using our bisimulation up to technique. The level 0 semantics ensure that the only substitution which occurs is for base values, names and triggers. Bisimilarity on these values is just syntactic identity so any problems with *substitutivity* (in the presence of static scoping) which arise in [10, 15] are avoided. We omit details of this as they can be recovered from the proof of Proposition 5.6. □

Given this we can draw upon the results of Corollary 4.7 and the above Proposition to obtain:

Theorem 4.9 *Bisimilarity is a congruence.*

5 A canonical labelled transition system

So far we have shown that bisimilarity coincides with barbed equivalence. The motivation for providing such a characterisation lies in the need to alleviate the quantification over all contexts present in the definition of barbed equivalence. We achieve this to an extent by reducing contexts to labelled transitions. However, despite being a neater coinductive equivalence, the definition of bisimilarity now quantifies over all transitions. We must question whether this is truly a lighter quantification. One measure we proposed in [15] to answer such a question was to demand that labels be *applicative*. That is to say, whenever a label contains an arbitrary value, the type of that value should be strictly less than the types of the threads being tested. Our labelled transition system defined above is certainly not applicative in this sense. In particular, the $\mathsf{join}(v)\,.\,n$ labels have no such restriction and grant powerful testing abilities. In order to rectify this shortcoming of our bisimilarity we provide a cut-down labelled transition semantics which do have applicative labels for which bisimilarity coincides with the original. This new semantics is closely related to *normal bisimulation* of Sangiorgi [29], in which a beautifully simple characterisation of bisimilarity for higher-order π-calculus is achieved by restricting test values to be either

names or trigger calls alone. We adopt a similar approach here by defining *canonical values* to be those of the form

$$v_c ::= b \mid n \mid x \mid (v_c, v_c) \mid \tau_a$$

Now, the canonical, or normal semantics for configurations is given by the labelled transition rules in Figures 4, 5 with all values in the transition labels restricted to be canonical. Write $[\![C]\!]_i^c$ to signify the canonical semantics with level i substitutions. If we can show that bisimilarity is a congruence on configurations for the canonical semantics it is a relatively simple step to prove the following theorem which justifies the claim that our bisimilarity is a simple characterisation of barbed equivalence.

Theorem 5.1 *The following are equivalent:*

(i) $\Delta \vDash [\![C]\!]_0 \approx [\![C']\!]_0$

(ii) $\Delta \vDash [\![C]\!]_\omega \approx [\![C']\!]_\omega$

(iii) $\Delta \vDash [\![C]\!]_\omega^c \approx [\![C']\!]_\omega^c$

(iv) $\Delta \vDash [\![C]\!]_0^c \approx [\![C']\!]_0^c$

Proof: The proof of (i) implies (ii) appears in the previous section and (ii) implies (iii) is immediate as the labels allowed for testing in the standard bisimulation subsume those allowed for bisimulation in the canonical semantics. Also, (iii) implies (iv) is easily established using the same proof as Corollary 4.7 for the canonical semantics. Finally, (iv) implies (i) makes use of the congruence properties of $\approx$ for the level 0 canonical semantics; we define $\mathcal{R}$ as:

$$\mathcal{R}_\Delta = \left\{ (C, C') \mid \Delta \vDash [\![C]\!]_0^c \approx [\![C']\!]_0^c \right\}$$

and proceed to show that $\mathcal{R}$ forms a bisimulation (up to β) in the level 0 semantics. Take $\Delta \vDash C_1 \; \mathcal{R} \; C_2$ (so that $\Delta \vDash [\![C_1]\!]_0^c \approx [\![C_2]\!]_0^c$) and suppose that $\Delta \vdash C_1 \xrightarrow{\alpha} \Delta' \vdash C_1'$. We must find a matching transition from C_2. If α only contains canonical values then this follows easily from the construction of $\mathcal{R}$. However, we must consider the cases in which α may contain non-canonical values: that is, α is $n\,.\,@v\,.\,n'$, $\mathsf{recv}(n, v)$, or $\mathsf{join}(v)\,.\,n$.

Consider the latter case: C_1' is simply $C_1 \parallel n[v]$. To find a match from C_2 we observe that, by congruence of bisimilarity for the canonical semantics, Proposition 5.6, we have that

$$\Delta \vDash [\![C_1 \parallel n[v]]\!]_0^c \approx [\![C_2 \parallel n[v]]\!]_0^c$$

hence $(\Delta \vdash C_2) \xrightarrow{\mathsf{join}(v).n} (\Delta \vdash C_2 \parallel n[v])$ serves as a matching transition.

In case α is $n\,.\,@v\,.\,n'$ we observe that

- C_1 is (up to structure) of the form $\nu\Delta_1\,.\,(C_0 \parallel n[\lambda x\,.\,t])$
- C_1' is (up to beta-reduction) of the form $\nu(n_0, a)\,.\,(C_A \parallel n_0[a \Leftarrow v])$ where C_A is $\nu\Delta_1\,.\,(C_0 \parallel n[\lambda x\,.\,t] \parallel n'[\mathsf{let}\, x = \tau_a \,\mathsf{in}\, t])$.

Thus we find that there are transitions

$$(\Delta \vdash C_1) \xrightarrow{\mathsf{weak}.a} \xrightarrow{n.@\tau_a.n'} (\Delta, a, n' \vdash C_A)$$

We know that $\Delta \vDash [\![C_1]\!]_0^c \approx [\![C_2]\!]_0^c$ so we have matching transitions

$$(\Delta \vdash C_2) \stackrel{\mathsf{weak}.a}{\Longrightarrow} \stackrel{n.@\tau_a.n'}{\Longrightarrow} (\Delta, a, n' \vdash C_B)$$

for some C_B such that $\Delta, a, n' \vDash [\![C_A]\!]_0^c \approx [\![C_B]\!]_0^c$. We also know by Proposition 5.6 that

$$\Delta, n' \vDash [\![\nu a, n_0 \,.\, (C_A \,\|\, n_0[a \Leftarrow v])]\!]_0^c \approx [\![\nu a, n_0 \,.\, (C_B \,\|\, n_0[a \Leftarrow v])]\!]_0^c$$

and it is straightforward to check that there exists a weak transition

$$(\Delta \vdash C_2) \stackrel{n.@v.n'}{\Longrightarrow} (\Delta, n' \vdash \nu a, n_0 \,.\, C_B \,\|\, n_0[a \Leftarrow v])$$

to match the α transition from C_1.

A similar argument may be used to match $\mathsf{recv}(n, v)$ transitions from C_1 and a symmetric argument enables us to match transitions from C_2. □

This does however oblige us to show congruence for the canonical semantics.

5.1 Bisimilarity is a congruence for the canonical semantics

The proof that bisimilarity for the canonical, level 0, semantics is preserved by contexts is non-trivial, and it will be helpful to present some lemmas to assist in its exposition. These are highly technical in nature and on first reading it may help the reader to skip to Proposition 5.6 in which congruence is proved. In order to assist reading of the proof of this proposition we offer a rough sketch of the difficulties involved:

We are trying to show that, whenever $\Delta \vDash C_1 \approx C_2$ then also, $\Delta \vDash C_1 \,\|\, C \approx C_2 \,\|\, C$. To do this one typically proceeds by coinduction by building a witnessing bisimulation $\mathcal{R}$ relating pairs of $C_1 \,\|\, C$ and $C_2 \,\|\, C$ for bisimilar C_1 and C_2. To match actions from $C_1 \,\|\, C$ (or its symmetric counterpart) one must consider the interactions between C_1 and C and show that they are simulated by interactions between C_2 and C. This gives rise to two difficulties which must be overcome in the proof. Firstly, suppose C_1 is willing to send a higher-order value along channel a. Then we notice

$$(\Delta \vdash C_1) \stackrel{\mathsf{send}(a).n}{\Longrightarrow} (\Delta, n \vdash C_1' \,\|\, n[v])$$

and $\Delta \vDash C_1 \approx C_2$ implies there is a matching transition from C_2 such that

$$\Delta, n \vDash C_1' \,\|\, n[v] \approx C_2' \,\|\, n[w] \tag{1}$$

When considered as an interaction we see that both $C_1 \,\|\, C$ and $C_2 \,\|\, C$ can reduce to reach terms

$$C_1' \,\|\, l[a \Leftarrow v] \,\|\, C'[\tau_a] \qquad \text{and} \qquad C_2' \,\|\, l[a \Leftarrow v] \,\|\, C'[\tau_a]$$

respectively and we need to conclude that these terms are related in $\mathcal{R}$. But this cannot be done immediately as (1) is in the wrong form to allow us to do this. Lemma 5.3(i) bridges this gap. The second

difficulty we encounter arises in this situation in which C_1 is trying to synchronise on a thread n which is to be provided by C as $n[v]$. This scenario is modelled in the labelled transition system (at least in the canonical semantics) by

$$(\Delta \vdash C_1) \xrightarrow{\mathsf{join}(\tau_a).n} (\Delta \vdash C_1' \parallel n[\tau_a]).$$

Using $\Delta \models C_1 \approx C_2$ we see that C_2 has a similar transition such that

$$\Delta \models C_1' \parallel n[\tau_a] \approx C_2' \parallel n[\tau_a] \tag{2}$$

But when considered as interactions in the level 0 semantics we see that $C_1 \parallel C$ and $C_2 \parallel C$ reduce to

$$C_1' \parallel n[v] \parallel C_0' \qquad \text{and} \qquad C_2' \parallel n[v] \parallel C_0'$$

respectively. Again, we need to conclude that these terms are related in $\mathcal{R}$ but cannot do so as (2) is in the wrong form to allow this. In fact, $\mathcal{R}$ must be defined with this property in mind (*cf.* proof of Proposition 5.6).

For the remainder of this section, all configurations are to be understood using the canonical, level 0 semantics.

Lemma 5.2

(i) *If* $\Delta, n:\sigma \models C_1 \approx C_2$ *then* $\Delta \models \nu n:\sigma \,.\, C_1 \approx \nu n:\sigma \,.\, C_2$.

(ii) *If* $\Delta, n \models C_1 \approx C_2$ *then* $\Delta \models C_1 \parallel n[v_c] \approx C_2 \parallel n[v_c]$.

(iii) *If* $\Delta, n:\sigma \models C_1 \parallel n[v_1] \approx C_2 \parallel n[v_2]$ *and* $n \notin C_1, C_2$ *then* $\Delta \models C_1 \approx C_2$.

(iv) *If* $\Delta, n \models C_1 \approx C_2$ *and* $n' \notin \Delta$ *then* $\Delta, n' \models C_1[n'/n] \approx C_2[n'/n]$

Proof: Straightforward coinductions. □

Lemma 5.3 *For* n *of type* $\sigma\,\mathsf{thread}$ *and for fresh* l *of type* $\mathsf{unit\,thread}$ *we have:*

(i) *If* $\Delta, n \models \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \approx \nu\Delta_2 \,.\, C_2 \parallel n[f_2]$ *then*

$$\Delta, n \models \nu\Delta_1, l \,.\, (C_1 \parallel n[f_1] \parallel l[a \Leftarrow f_1]) \approx \nu\Delta_2, l \,.\, (C_2 \parallel n[f_2] \parallel l[a \Leftarrow f_2])$$

for any abstractions f_1, f_2.

(ii) *If* $\Delta, n \models \nu\Delta_1 \,.\, (C_1 \parallel n[t_1]) \approx \nu\Delta_2 \,.\, (C_2 \parallel n[t_2])$ *and* $n \notin fn(C_1), fn(C_2)$ *then*

$$\Delta \models \nu\Delta_1, l \,.\, (C_1 \parallel l[\mathsf{let}\, x = t_1 \,\mathsf{in}\, \mathsf{send}\,(r,x)]) \approx \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{let}\, x = t_2 \,\mathsf{in}\, \mathsf{send}\,(r,x)])$$

for $\Delta \vdash r : \sigma\,\mathsf{chan}$.

(iii) *If* $\Delta, n \models \nu\Delta_1 \,.\, (C_1 \parallel n[v_1]) \approx \nu\Delta_2 \,.\, (C_2 \parallel n[v_2])$ *and* $n \notin fn(C_1), fn(C_2)$ *then*

$$\Delta \models \nu\Delta_1, l \,.\, (C_1 \parallel l[\mathsf{send}\,(r,v_1)]) \approx \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r,v_2)])$$

for $\Delta \vdash r : \sigma\,\mathsf{chan}$.

Proof: We proceed by showing these simultaneously by induction on the functional structure of type σ using the following inductive structure:

- (i) implies (iii),
- (i) and (iii) imply (ii),
- (i) and (ii) (at smaller types than σ) imply (i)

Specifically, we let $\mathsf{bool}, \mathsf{unit}, \sigma'\,\mathsf{thread}, \sigma'\,\mathsf{chan}$ (for any σ') be considered as base types, and order higher types by $\sigma_1, \sigma_2 < \sigma_1 \to \sigma_2$.

We will demonstrate the dependencies listed above. For the sake of a simpler exposition we will assume that all the values v_1, v_2 referred to in the following proof are not of the form (v', v''). This does not constitute any real restriction as an obvious mixture of the proof techniques described below for base and higher types serves to validate the Lemma for such values.

- Suppose (i) holds: we show (iii).

  To begin with we let

  $$\Delta \vDash \nu\Delta_1, l\,.\,(C_1 \parallel l[\mathsf{send}\,(r, v_1)])\ \mathcal{R}\ \nu\Delta_2, l\,.\,(C_2 \parallel l[\mathsf{send}\,(r, v_2)])$$

  hold exactly when $\Delta, n \vDash \nu\Delta_1\,.\,(C_1 \parallel n[v_1]) \approx \nu\Delta_2\,.\,(C_2 \parallel n[v_2])$. We will show that $\mathcal{R} \cup \approx$ forms a bisimulation relation:

  Suppose that $\Delta \vDash C\ \mathcal{R}\ C'$ is witnessed by $\Delta \vDash C \approx C'$. The diagram required to demonstrate bisimulation is trivially closed. Therefore we can assume that $\Delta \vDash C\ \mathcal{R}\ C'$ is of the form:

  $$\Delta \vDash \nu\Delta_1, l\,.\,(C_1 \parallel l[\mathsf{send}\,(r, v_1)])\ \mathcal{R}\ \nu\Delta_2, l\,.\,(C_2 \parallel l[\mathsf{send}\,(r, v_2)]),$$

  witnessed by

  $$\Delta, n \vDash \nu\Delta_1\,.\,(C_1 \parallel n[v_1]) \approx \nu\Delta_2\,.\,(C_2 \parallel n[v_2]).$$

  Furthermore suppose that $\Delta \vdash \nu\Delta_1, l\,.\,C_1 \parallel l[\mathsf{send}\,(r, v_1)] \xrightarrow{\alpha} \Delta' \vdash D_1$ with $n \notin \alpha$. We consider the possible forms for this transition:

  - Firstly, α may have originated in C_1, that is, Δ' is Δ, Δ_0 where the domain of Δ_0 is the bound names of α, and if we write Δ_i as Δ_i', Δ_0 (for $i = 1, 2$), then we have D_1 is, up to structural equivalence, of the form

    $$\nu\Delta_1', l\,.\,(C_1' \parallel l[\mathsf{send}\,(r, v_1)]).$$

    In this case we also know that

    $$\Delta, n \vdash \nu\Delta_1\,.\,(C_1 \parallel n[v_1]) \xrightarrow{\alpha} \Delta', n \vdash \nu\Delta_1'\,.\,(C_1' \parallel n[v_1])$$

    and we know that the closure condition on $\approx$ guarantees a matching transition

    $$\Delta, n \vdash \nu\Delta_2\,.\,(C_2 \parallel n[v_2]) \xRightarrow{\hat{\alpha}} \Delta', n \vdash \nu\Delta_2'\,.\,(C_2' \parallel n[v_2])$$

with

$$\Delta', n \vDash \nu\Delta_1' \,.\, (C_1' \parallel n[v_1]) \quad \approx \quad \nu\Delta_2' \,.\, (C_2' \parallel n[v_2]) \tag{3}$$

Now we know that this transition cannot depend on n as n is not contained in α or C_2. Therefore

$$\Delta \vdash \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r, v_2)]) \stackrel{\hat{\alpha}}{\Longrightarrow} \Delta' \vdash \nu\Delta_2', l \,.\, (C_2' \parallel l[\mathsf{send}\,(r, v_2)])$$

also holds. Let us call the target of these transitions D_2. We use equation (3) and the definition of $\mathcal{R}$ to observe that $\Delta' \vDash D_1 \;\mathcal{R}\; D_2$.

– Secondly, α may be a $\mathsf{join}(v_c) \,.\, n$ transition. We know by Lemma 5.2 that we can simply use a $\mathsf{join}(v_c) \,.\, n$ from $\nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r, v_2)])$ to match this.

– Thirdly, α is $\mathsf{send}(r) \,.\, n'$ and $\Delta' \vdash D_1$ is (up to β-reduction) $\Delta, n' \vdash \nu\Delta_1, l \,.\, (C_1 \parallel l[()] \parallel n'[v_1])$. We simply observe that

$$\Delta \vdash \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r, v_2)]) \xrightarrow{\alpha} \nu\Delta_2, l \,.\, (C_2 \parallel l[()] \parallel n'[v_2])$$

with $\Delta' \vDash D_1 \approx \nu\Delta_2, l \,.\, (C_2 \parallel l[()] \parallel n'[v_2])$ by hypothesis and Lemma 5.2.

– Finally, if α is τ arising from communication then we must have

$$C_1 \equiv \nu\Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\, x = \mathsf{recv}\, r\, \mathsf{in}\, t_1]) \qquad D_1 \stackrel{\beta}{\rightarrow}{}^{*} \nu\Delta_1, \Delta_1', l \,.\, (C_1' \parallel n_0[\mathsf{let}\, x = v_1\, \mathsf{in}\, t_1] \parallel l[()]).$$

In this case we also know that if v_1 has base type then it is actually a canonical value and hence

$$\Delta \vdash \nu\Delta_1 \,.\, (C_1 \parallel n[v_1]) \xrightarrow{\mathsf{recv}(r, v_1)} \Delta \vdash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\, x_1 = v_1\, \mathsf{in}\, t_1] \parallel n[v_1])$$

is a valid transition. In this case v_2 is also canonical as it must be identical to v_1. We appeal to the hypothesis to see that

$$\Delta \vdash \nu\Delta_2 \,.\, (C_2 \parallel n[v_2]) \stackrel{\mathsf{recv}(r, v_1)}{\Longrightarrow} \Delta \vdash \nu\Delta_2 \,.\, (C_2' \parallel n[v_2])$$

for some C_2' such that $\Delta \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\, x = v_1\, \mathsf{in}\, t_1] \parallel n[v_1]) \approx \nu\Delta_2 \,.\, (C_2' \parallel n[v_2])$. Therefore we know that

$$\Delta \vdash \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r, v_2)]) \Rightarrow \Delta \vdash \nu\Delta_2, l \,.\, (C_2' \parallel l[()])$$

and we now just need to use Lemma 5.2 to see that

$$\Delta \vDash D_1 \approx \nu\Delta_2, l \,.\, C_2' \parallel l[()]$$

We must also consider the case in which v_1 has functional type. Here, v_1 may not be canonical so we cannot immediately use a $\mathsf{recv}(r, v_1)$ transition to help us find our match. However, we may note that we have

$$\Delta' \vDash D_1 \stackrel{\beta}{\rightarrow}{}^{*} \nu\Delta_1, \Delta_1', l, l', a \,.\, (C_1' \parallel n_0[t_1[\tau_a/x]] \parallel l[()] \parallel l'[a \Leftarrow v_1]).$$

Thus, we can use a weakening transition and

$$(\Delta, n, a \vdash \nu\Delta_1 \,.\, (C_1 \parallel n[v_1])) \xrightarrow{\mathsf{recv}(r,\tau_a)} (\Delta, n, a \vdash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\, x = \tau_a \,\mathsf{in}\, t_1] \parallel n[v_1]))$$

to obtain matching transitions

$$(\Delta, n, a \vdash \nu\Delta_2 \,.\, (C_2 \parallel n[v_2])) \xRightarrow{\mathsf{recv}(r,\tau_a)} (\Delta, n, a \vdash \nu\Delta_2 \,.\, (C_2' \parallel n[v_2]))$$

such that

$$\Delta, n, a \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\, x = \tau_a \,\mathsf{in}\, t_1] \parallel n[v_1]) \quad \approx \quad \nu\Delta_2 \,.\, (C_2' \parallel n[v_2]) \tag{4}$$

We use these transitions, and the fact that $n \notin fn(C_2), \alpha$, to observe that there must exist some D_2 such that

$$\Delta \vdash \nu n_2, l \,.\, (C_2 \parallel l[\mathsf{send}\,(r, v_2)]) \Rightarrow \Delta \vdash D_2 \qquad \Delta \vDash D_2 \equiv \nu n_2, l, l', a \,.\, (C_2' \parallel l[()] \parallel l'[a \Leftarrow v_2]).$$

We now simply apply part (i) and Lemma 5.2 to (4) to obtain

$$\Delta \vDash D_1 \approx \nu n_2, l, l', a \,.\, (C_2' \parallel l[()] \parallel l'[a \Leftarrow v_2]) \approx D_2$$

as required.

- Suppose (i) and (iii) hold: we show (ii).

  Again we build a relation $\mathcal{R}$ and show that $\mathcal{R} \cup \approx$ forms a bisimulation.

  Suppose (wlog) we have

$$\Delta \vDash \nu\Delta_1, l \,.\, (C_1 \parallel l[\mathsf{let}\, x = t_1 \,\mathsf{in}\, \mathsf{send}\,(r, x)]) \;\mathcal{R}\; \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{let}\, x = t_2 \,\mathsf{in}\, \mathsf{send}\,(r, x)])$$

  witnessed by

$$\Delta, n \vDash \nu\Delta_1 \,.\, (C_1 \parallel n[t_1]) \approx \nu\Delta_2 \,.\, (C_2 \parallel n[t_2])$$

  and suppose that

$$\Delta \vdash \nu\Delta_1, l \,.\, (C_1 \parallel l[\mathsf{let}\, x = t_1 \,\mathsf{in}\, \mathsf{send}\,(r, x)]) \xrightarrow{\alpha} \Delta' \vdash D_1$$

  with $n \notin \alpha$. Now if α originates in C_1 or t_1, or even as an interaction between the two, this is easily dealt with using the hypothesis. Similarly, if α is a $\mathsf{join}(v) \,.\, n'$ transition then we can easily find a matching transition. The case of interest arises when t_1 is actually a value, v_1, say and α is a β-reduction. If v_1 is of base type then v_1 is necessarily a canonical value so D_1 will be of the form $\nu\Delta_1, l \,.\, (C_1 \parallel l[\mathsf{send}\,(r, v_1)])$. It is relatively easy to see using the value transitions at n that the witness guarantees that there exists some transitions

$$\Delta, n \vdash \nu\Delta_2 \,.\, (C_2 \parallel n[t_2]) \Rightarrow \Delta, n \vdash \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[v_2])$$

  such that $\Delta, n \vDash \nu\Delta_1 \,.\, (C_1 \parallel n[v_1]) \approx \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[v_2])$ with v_1 identical to v_2 (note that this supposes that v_1 is not contained in Δ_1, in which case the new name transitions can be used to weaken the name environment Δ to include v_1). Given this we can also see that

$$\Delta \vdash \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{let}\, x = t_2 \,\mathsf{in}\, \mathsf{send}\,(r, x)]) \Rightarrow \Delta \vdash \nu\Delta_2, l, \Delta_2' \,.\, (C_2' \parallel l[\mathsf{send}\,(r, v_2)])$$

By hypothesis, we can now use (iii) to see that

$$\Delta \vDash D_1 \approx \nu\Delta_2, l, \Delta_2' \,.\, (C_2' \parallel l[\mathsf{send}\,(r, v_2)]).$$

Otherwise, v_1 is of functional type $\sigma_1 \to \sigma_2$. In this case we know that, D_1 must be of the form

$$\nu\Delta_1, l, l', a' \,.\, (C_1 \parallel l[\mathsf{send}\,(r, \tau_{a'})] \parallel l'[a' \Leftarrow v_1]).$$

As above we still know, because of $n \Downarrow$ transitions, that there must exist transitions

$$\Delta \vdash \nu\Delta_2 \,.\, (C_2 \parallel n[t_2]) \Rightarrow \Delta \vdash \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[v_2])$$

with $\Delta, n \vDash \nu\Delta_1 \,.\, (C_1 \parallel n[v_1]) \approx \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[v_2])$. We can apply Lemma 5.2 to this, after weakening to obtain

$$\Delta, n, n', a' \vDash \nu\Delta_1 \,.\, (C_1 \parallel n[v_1] \parallel n'[\tau_{a'}]) \approx \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[v_2] \parallel n'[\tau_{a'}])$$

and we can then apply part (iii) to obtain

$$\Delta, n, a' \vDash \nu\Delta_1, l \,.\, (C_1 \parallel n[v_1] \parallel l[\mathsf{send}\,(r, \tau_{a'})]) \quad \approx \quad \nu\Delta_2, \Delta_2', l \,.\, (C_2' \parallel n[v_2] \parallel l[\mathsf{send}\,(r, \tau_{a'})]). \quad (5)$$

We conclude by observing that

$$\Delta \vdash \nu\Delta_2, l \,.\, (C_2 \parallel l[\mathsf{let}\, x = t_2 \,\mathsf{in}\, \mathsf{send}\,(r, x)]) \Rightarrow \Delta \vdash \nu\Delta_2, \Delta_2', l, l', a' \,.\, (C_2' \parallel l'[a' \Leftarrow v_2] \parallel l[\mathsf{send}\,(r, \tau_{a'})])$$

(call this latter term D_2) We use part (i) with (5) (with suitable renamings) and Lemma 5.2, to get

$$\Delta \vDash D_1 \approx D_2$$

as required.

- Suppose (i) holds at type σ_1 and (ii) holds at type σ_2 where σ is $\sigma_1 \to \sigma_2$: we show (i).
  As before we build a relation $\mathcal{R}$ by defining

$$\Delta, n \vDash \nu\Delta_1, l \,.\, (C_1 \parallel n[f_1] \parallel l[a \Leftarrow f_1]) \;\mathcal{R}\; \nu\Delta_2, l \,.\, (C_2 \parallel n[f_2] \parallel l[a \Leftarrow f_2])$$

  if

$$\Delta, n \vDash \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \approx \nu\Delta_2 \,.\, C_2 \parallel n[f_2]$$

  for some $a \in \Delta$. We show that $\mathcal{R}$ forms a bisimulation up to $(\xrightarrow{\beta}{}^*, \approx)$. We will assume that f_1 is of the form $\lambda x.t_1$. Suppose also that

$$\Delta, n \vdash \nu\Delta_1, l \,.\, (C_1 \parallel n[f_1] \parallel l[a \Leftarrow f_1]) \xrightarrow{\alpha} \Delta' \vdash D_1$$

  for some Δ' and D_1. We consider the cases for α:

  – Firstly, if α originates in $C_1 \parallel n[f_1]$, or is a $\mathsf{join}(v) \,.\, n$ transition then we may find the matching transition as above.

– Of more interest is the case in which the environment calls the resource $l[a \Leftarrow f_1]$ with a $\mathsf{recv}(a,(v_c,r))$ transition. That is D_1 is, up to β-reduction, of the form:

$$\nu\Delta_1, l, l' \,.\, (C_1 \parallel n[f_1] \parallel l[\mathsf{let}\, z = (\mathsf{let}\, x = v_c\, \mathsf{in}\, t_1)\, \mathsf{in}\, \mathsf{send}\,(r,z)] \parallel l'[a \Leftarrow f_1]).$$

Now we know that (for $\alpha = \mathsf{recv}(a,(v_c,r))$)

$$\begin{array}{l}\Delta, n \vdash \nu\Delta_2, l \,.\, (C_2 \parallel n[f_2] \parallel l[a \Leftarrow f_2]) \\ \quad \stackrel{\alpha}{\Longrightarrow} \Delta, n \vdash \nu\Delta_2, l, l' \,.\, (C_2 \parallel n[f_2] \parallel l[\mathsf{let}\, z = f_2\, v_c\, \mathsf{in}\, \mathsf{send}\,(r,z)] \parallel l'[a \Leftarrow f_2])\end{array}$$

is also a valid transition, call the target of it D_2, say. We also know that,

$$\Delta, n \vdash \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \xrightarrow{n.@v_c.n'} \Delta, n, n' \vdash \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \parallel n'[\mathsf{let}\, x = v_c\, \mathsf{in}\, t_1].$$

Thus, by hypothesis, we know that there exists a matching weak transition,

$$\Delta, n \vdash \nu\Delta_2 \,.\, C_2 \parallel n[f_2] \stackrel{n.@v_c.n'}{\Longrightarrow} \Delta, n, n' \vdash \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[f_2] \parallel n'[t_2])$$

such that

$$\Delta, n, n' \vDash \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \parallel n'[\mathsf{let}\, x = v_c\, \mathsf{in}\, t_1] \quad \approx \quad \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n[f_2] \parallel n'[t_2]). \tag{6}$$

Moreover, it can easily seen by analysing the matching transitions that

$$\Delta, n \vdash D_2 \Rightarrow \Delta, n \vdash \nu\Delta_2, \Delta_2', l, l' \,.\, (C_2' \parallel n[f_2] \parallel l[\mathsf{let}\, z = t_2\, \mathsf{in}\, \mathsf{send}\,(r,z)] \parallel l'[a \Leftarrow f_2])$$

also. Call the target of these transitions D_2'. Thus we have

$$(\Delta, n \vdash \nu\Delta_2, l \,.\, C_2 \parallel l[a \Leftarrow f_2]) \stackrel{\alpha}{\Longrightarrow} (\Delta, n \vdash D_2) \Rightarrow (\Delta, n \vdash D_2')$$

and, by using part (ii) at n' at type σ_2 with (6), and then by definition of $\mathcal{R}$, we have $\Delta \vDash D_1 \stackrel{\beta}{\rightarrow}{}^* \mathcal{R} \approx D_2'$.

– Finally, we consider the case in which α is τ, a communication between C_1 and $l[a \Leftarrow f_1]$. In this case, C_1 must be (up to $\equiv$) of the form

$$\nu\Delta_1' \,.\, (C_1' \parallel n_0[\mathsf{let}\,() = \mathsf{send}\,(a,(v_1,r))\, \mathsf{in}\, t_1']).$$

We know that, by hypothesis,

$$\Delta, n \vDash \nu\Delta_1 \,.\, C_1 \parallel n[f_1] \approx \nu\Delta_2 \,.\, C_2 \parallel n[f_2],$$

and we also know that

$$(\Delta, n \vdash \nu\Delta_1 \,.\, (C_1 \parallel n[f_1])) \xrightarrow{\mathsf{send}(a).n'} \stackrel{\beta}{\rightarrow}{}^* (\Delta, n, n' \vdash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n'[(v_1,r)] \parallel n[f_1])).$$

Given this we can find matching transitions

$$(\Delta, n \vdash \nu\Delta_2 \,.\, (C_2 \parallel n[f_2])) \stackrel{\mathsf{send}(a).n'}{\Longrightarrow} (\Delta, n, n' \vdash \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n'[w] \parallel n[f_2]))$$

with

$$\Delta, n, n' \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n'[(r, v_1)] \parallel n[f_1]) \approx \nu\Delta_2, \Delta_2' \,.\, (C_2' \parallel n'[w] \parallel n[f_2]).$$

Now, by using $n' \,.\, \mathsf{fst}.$ and $n' \,.\, \mathsf{snd}.$ projection transitions and Lemma 5.2, and because $n' \notin C_2'$, we know that w must be of the form (v_2, r) for some v_2, moreover we can find C_2'' and n'' such that

$$\Delta, n, n'' \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n''[v_1] \parallel n[f_1]) \quad \approx \quad \nu\Delta_2, \Delta_2' \,.\, (C_2'' \parallel n''[v_2] \parallel n[f_2]) \qquad (7)$$

with $C_2' \parallel n[f_2] \Rightarrow C_2'' \parallel n[f_2]$.

Now we must consider two subcases according to the type of v_1:

Case(a): If v_1 is of base type then D_1, up to β-reduction, will be of the form (where f_1 is $\lambda x.t_1$)

$$\nu\Delta_1, \Delta_1', l, l' \,.\, (C_1' \parallel n_0[t_1'] \parallel n[f_1] \parallel l[\mathsf{let}\, x' = (\mathsf{let}\, x = v_1\, \mathsf{in}\, t_1)\, \mathsf{in}\, \mathsf{send}\, (r, x')] \parallel l'[a \Leftarrow f_1])$$

In this case, we know that v_1 is canonical and, by the previous equivalence, we know that v_2 is also canonical, and moreover is identical to v_1. We can use Lemma 5.2 to see that

$$\Delta, n \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n[f_1]) \approx \nu\Delta_2, \Delta_2' \,.\, (C_2'' \parallel n[f_2])$$

(note that this ignores the case in which v_1, v_2 contain private names, but we may assume, because of the $n \,.\, \nu m$ transitions, that such names have already been extruded). We have stated that v_1 is canonical so there exists a transition

$$\begin{aligned} &\Delta, n \vdash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n[f_1]) \\ &\quad \xrightarrow{n.@v_1.n'} \Delta, n, n' \vdash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n[f_1] \parallel n'[\mathsf{let}\, x = v_1\, \mathsf{in}\, t_1]) \end{aligned}$$

where n' is fresh. This means that there must exist matching transitions

$$\Delta, n \vdash \nu\Delta_2, \Delta_2' \,.\, (C_2'' \parallel n[f_2]) \xRightarrow{n.@v_1.n'} \Delta, n, n' \vdash \nu\Delta_2, \Delta_2', \Delta_2'' \,.\, (C_2''' \parallel n[f_2] \parallel n'[t_2])$$

such that

$$\begin{gathered} \Delta, n, n' \vDash \nu\Delta_1, \Delta_1' \,.\, (C_1' \parallel n_0[t_1'] \parallel n[f_1] \parallel n'[\mathsf{let}\, x = v_1\, \mathsf{in}\, t_1]) \\ \approx \\ \nu\Delta_2, \Delta_2', \Delta_2'' \,.\, (C_2''' \parallel n[f_2] \parallel n'[t_2]) \end{gathered} \qquad (8)$$

and $\nu\Delta_2, \Delta_2' \,.\, (C_2'' \parallel n[f_2] \parallel n'[f_2\, v_2]) \Rightarrow \nu\Delta_2, \Delta_2', \Delta_2'' \,.\, (C_2''' \parallel n[f_2] \parallel n'[t_2])$. By collecting the above transitions together though we see that

$$\begin{aligned} &\Delta, n \vdash \nu\Delta_2, l \,.\, (C_2 \parallel n[f_2] \parallel l[a \Leftarrow f_2]) \\ &\quad \Rightarrow \Delta, n \vdash \nu\Delta_2, \Delta_2', l, l' \,.\, (C_2'' \parallel n[f_2] \parallel l[\mathsf{let}\, x = f_2\, v_2\, \mathsf{in}\, \mathsf{send}\, (r, x)] \parallel l'[a \Leftarrow f_2]) \end{aligned}$$

and moreover, this configuration further reduces to

$$\Delta, n \vdash \nu\Delta_2, \Delta_2', \Delta_2'', l, l' \,.\, (C_2''' \parallel n[f_2] \parallel l[\mathsf{let}\, x = t_2\, \mathsf{in}\, \mathsf{send}\, (r, x)] \parallel l'[a \Leftarrow f_2])$$

which we will call D_2. We can now use part (ii) at n'' at type σ_2 on (8), and the definition of $\mathcal{R}$ to see that $\Delta, n \vDash D_1 \xrightarrow{\beta}{}^* \mathcal{R} \approx D_2$ as required.

Case(b): If v_1 is an abstraction, necessarily of functional type. We know that D_1 will again be, up to β-reduction, of the form

$$\nu\Delta_1, \Delta_1', a', l, l', l''.$$
$$(C_1' \parallel n_0[t_1'] \parallel n[f_1] \parallel l[\mathsf{let}\, x' = (\mathsf{let}\, x = \tau_a\, \mathsf{in}\, t_1)\, \mathsf{in}\, \mathsf{send}\,(r, x')] \parallel l'[a \Leftarrow f_1] \parallel l''[a' \Leftarrow v_1])$$

We may not be able to use an $n\,.\,@v_1\,.\,n'$ transition as above because v_1 may be non-canonical, or even a trigger call on a private name. However, we can choose a fresh trigger name, a', say, and, after weakening, observe the following transition:

$$\Delta, n, n'', a' \vdash \nu\Delta_1, \Delta_1'\,.\,(C_1' \parallel n_0[t_1'] \parallel n''[v_1] \parallel n[f_1])$$
$$\xrightarrow{n.@\tau_{a'}.n'} \Delta, n, n', n'', a' \vdash \nu\Delta_1, \Delta_1'\,.\,(C_1' \parallel n_0[t_1'] \parallel n''[v_1] \parallel n[f_1] \parallel n'[\mathsf{let}\, x = \tau_{a'}\, \mathsf{in}\, t_1])$$

where n' is fresh and f_1 is $\lambda x.t_1$. This means that, by (7), there must exist matching transitions

$$\Delta, n, n'', a' \vdash \nu\Delta_2, \Delta_2'\,.\,(C_2'' \parallel n''[v_2] \parallel n[f_2])$$
$$\Longrightarrow^{n.@\tau_{a'}.n'}$$
$$\Delta, n, n', n'', a' \vdash \nu\Delta_2, \Delta_2', \Delta_2''\,.\,(C_2''' \parallel n''[v_2] \parallel n[f_2] \parallel n'[t_2])$$

such that

$$\Delta, n, n', n'', a' \vDash \nu\Delta_1, \Delta_1'\,.\,(C_1' \parallel n_0[t_1'] \parallel n''[v_1] \parallel n[f_1] \parallel n'[\mathsf{let}\, x = \tau_{a'}\, \mathsf{in}\, t_1]) \tag{9}$$
$$\approx$$
$$\nu\Delta_2, \Delta_2', \Delta_2''\,.\,(C_2''' \parallel n''[v_2] \parallel n[f_2] \parallel n'[t_2])$$

and $\nu\Delta_2, \Delta_2'\,.\,(C_2'' \parallel n[f_2] \parallel n'[f_2\,\tau_{a'}]) \Rightarrow \nu\Delta_2, \Delta_2', \Delta_2''\,.\,(C_2''' \parallel n[f_2] \parallel n'[t_2])$. We can apply part (i) at n'' at type σ_1, Lemma 5.2 and part (ii) at n' at type σ_2 to equation (9) to obtain

$$\Delta, n \vdash \nu\Delta_1, \Delta_1', l, l'', a'\,.\,(C_1' \parallel n_0[t_1'] \parallel n[f_1] \parallel l[\mathsf{let}\, x' = \mathsf{let}\, x = \tau_{a'}\, \mathsf{in}\, t_1\, \mathsf{in}\, \mathsf{send}\,(r, x')] \parallel l''[a' \Leftarrow v_1])$$
$$\approx$$
$$\nu\Delta_2, \Delta_2', \Delta_2'', a', l, l''\,.\,(C_2''' \parallel l''[a' \Leftarrow v_2] \parallel n[f_2] \parallel l[\mathsf{let}\, x = t_2\, \mathsf{in}\, \mathsf{send}\,(r, x)]) \tag{10}$$

We also note that, by collecting the matching transitions together,

$$\Delta, n \vdash \nu\Delta_2, l\,.\,(C_2 \parallel n[f_2] \parallel l[a \Leftarrow f_2])$$
$$\Rightarrow$$
$$\Delta, n \vdash \nu\Delta_2, \Delta_2', l, l', l'', a'\,.\,(C_2'' \parallel n[f_2] \parallel l[\mathsf{let}\, x = f_2\, \tau_{a'}\, \mathsf{in}\, \mathsf{send}\,(r, x)] \parallel l'[a \Leftarrow f_2] \parallel l''[a' \Leftarrow v_2])$$

and moreover, this configuration further reduces to

$$\Delta \vdash \nu\Delta_2, \Delta_2', \Delta_2'', l, l', l'', a'\,.\,(C_2''' \parallel n[f_2] \parallel l[\mathsf{let}\, x = t_2\, \mathsf{in}\, \mathsf{send}\,(r, x)] \parallel l'[a \Leftarrow f_2] \parallel l''[a' \Leftarrow v_2])$$

which we will call D_2. This means that, by (10), we have

$$\Delta \vdash \nu\Delta_2, l\,.\,(C_2 \parallel l[a \Leftarrow f_2]) \Rightarrow \Delta \vdash D_2$$

such that that $\Delta \vDash D_1 \xrightarrow{\beta}{}^{*} \mathcal{R} \approx D_2$ as required. □

Lemma 5.4 $\Delta \vDash \nu a, l\,.\,(C[v/x]_0 \,\|\, l[a \Leftarrow v]) \approx \nu a, l\,.\,(C[\tau_a/x] \,\|\, l[a \Leftarrow v])$ *for all* C, v.

Proof: Let the type-indexed relation $\mathcal{R}$ be defined:

$$\Delta \vDash \nu a, l\,.\,(C[a/a'] \,\|\, l[a \Leftarrow v]) \;\mathcal{R}\; \nu a, a', l, l'\,.\,(C \,\|\, l[a \Leftarrow v] \,\|\, l'[a' \Leftarrow v])$$

when a and a' are send-channels in C, and $l, l' \notin C$. We show that $\mathcal{R}$ is a bisimulation, and the result follows. □

Lemma 5.5 *If*

$$(\Delta, a \vdash C \,\|\, n[\tau_a]) \xrightarrow{\alpha} (\Delta', a \vdash C' \,\|\, n[\tau_a])$$

with α *not of the form* $n\,.\,@v_0\,.\,n_0$, *and* a *is a* send*-channel in* C *and* $\Delta, a \vdash C_0$ *is of the form*

$$\nu\Delta_0\,.\,(n[v] \,\|\, \nu l\,.\,l[a \Leftarrow v] \,\|\, C_0')$$

with $n, a \notin \Delta_0$, *then*

$$(\Delta \vdash \nu a\,.\,(C \,\|\, C_0)) \xrightarrow{\alpha} \approx (\Delta' \vdash \nu a\,.\,(C' \,\|\, C_0)).$$

Proof: The only transition of $C \,\|\, n[\tau_a]$ which may be prevented by $\nu a\,.\,(C \,\|\, C_0)$ is that in which C performs a join communication on n to receive τ_a. That is, C is of the form

$$\nu\Delta_1\,.\,(C_1 \,\|\, n_1[\mathsf{let}\, x = \mathsf{join}\, n \,\mathsf{in}\, t_1])$$

and C' is of the form

$$\nu\Delta_1\,.\,(C_1 \,\|\, n_1[\mathsf{let}\, x = \tau_a \,\mathsf{in}\, t_1]).$$

Clearly though,

$$\begin{array}{rcl}
\nu a\,.\,(C \,\|\, C_0) & \rightarrow & \nu a, \Delta_1\,.\,(C_1 \,\|\, n_1[\mathsf{let}\, x = v \,\mathsf{in}\, t_1] \,\|\, C_0) \\
 & \xrightarrow{\beta} & \nu a, \Delta_1\,.\,(C_1 \,\|\, n_1[t_1] \,\|\, C_0)[v/x]_0 \\
\text{(Lemma 5.4)} & \approx & \nu a, \Delta_1\,.\,(C_1 \,\|\, n_1[t_1[\tau_a/x]] \,\|\, C_0) \\
 & \equiv & \nu a\,.\,(C' \,\|\, C_0)
\end{array}$$

as required. □

Having shown these rather technical lemmas we may now proceed with the main Proposition: congruence of bisimilarity for the level 0, canonical semantics.

Proposition 5.6 *If* $\Delta, \Delta_0 \vDash C_1 \approx C_2$ *and* $\Delta, \Delta_0 \vdash C$ *then* $\Delta \vDash \nu\Delta_0\,.\,(C_1 \,\|\, C) \approx \nu\Delta_0\,.\,(C_2 \,\|\, C)$.

Proof: Define:

$$\Delta \vDash \nu\Delta_0\,.\,(C_1 \,\|\, C) \;\mathcal{R}\; \nu\Delta_0\,.\,(C_2 \,\|\, C)$$

iff $\Delta, \Delta_0 \vdash C$ and there exists some $\vec{n}$, $\vec{a}$, such that

$$\Delta, \Delta_0 \vDash C_1 \,\|\, \prod_{i=1}^{k} n_i[\tau_{a_i}] \approx C_2 \,\|\, \prod_{i=1}^{k} n_i[\tau_{a_i}]$$

and such that a_i is a send-channel in C_1, C_2, $n_i \notin \Delta_0$, $a_i \in \Delta_0$ (for $1 \leq i \leq k$), and

$$C \equiv \nu\Delta', \vec{l}\,.\,(\prod_{i=1}^{k} n_i[v_i] \,\|\, \prod_{i=1}^{k} l_i[a_i \Leftarrow v_i] \,\|\, C').$$

We will demonstrate $\mathcal{R}$ to be bisimulation up to $(\stackrel{\beta}{\rightarrow}{}^*, \approx)$ and our result follows in the case $k = 0$. Suppose then that

$$\Delta \vDash \nu\Delta_0\,.\,(C_1 \,\|\, C)\ \mathcal{R}\ \nu\Delta_0\,.\,(C_2 \,\|\, C)$$

and also suppose that $(\Delta \vdash \nu\Delta_0\,.\,(C_1 \,\|\, C)) \xrightarrow{\alpha} (\Delta' \vdash D_1)$ for some $\Delta' \vdash D_1$. We must check the bisimulation closure property. If α originates in C then it is clear that there is a matching transition from $(\Delta \vdash \nu\Delta_0\,.\,(C_2 \,\|\, C))$. Similarly, if α originates in C_1, we consider the definition of $\mathcal{R}$ to see that

$$\Delta, \Delta_0 \vDash C_1 \,\|\, \prod_{i=1}^{k} n_i[\tau_{a_i}] \approx C_2 \,\|\, \prod_{i=1}^{k} n_i[\tau_{a_i}].$$

For the purposes of exposition, unless otherwise stated we will suppose that $k = 1$ as this suffices to show the relevant proof without burdening the reader with excessive detail. For other values of k the proof follows in a very similar manner. Therefore we know that there must exist some matching transitions from

$$(\Delta, \Delta_0 \vdash C_2 \,\|\, n[\tau_a])$$

and that by iterating Lemma 5.5 we see that these matching transitions are valid from

$$(\Delta \vdash \nu\Delta_0\,.\,(C_2 \,\|\, C))$$

also. The difficulties arise in the cases in which α is the result of an interaction between C_1 and C, in particular: channel communication between these configurations and join synchronizations.

We consider these cases in turn. Note that we will omit the cases for communication with a replicated receive expression as they are very similar to the cases for the single receive. Also, we will assume for the sake of clear exposition that all values communicated will be either base type values, names, or abstractions. This is of course no real restriction as the same proof technique with judicious use of projection transitions is valid for tupled values also.

- Suppose

$$\begin{array}{rcl} C_1 & \equiv & \nu\Delta_1\,.\,(C_1' \,\|\, n_1[\mathsf{let}\,() = \mathsf{send}\,(c, v_1)\,\mathsf{in}\,t_1]) \\ C & \equiv & \nu\Delta_0'\,.\,(C_0' \,\|\, n_0[\mathsf{let}\,x = \mathsf{recv}\,c\,\mathsf{in}\,t_0]) \\ D_1 & \stackrel{\beta}{\rightarrow}{}^* & \nu\Delta_0, \Delta_1, \Delta_0'\,.\,(C_1' \,\|\, n_1[t_1] \,\|\, C_0' \,\|\, n_0[\mathsf{let}\,x = v_1\,\mathsf{in}\,t_0]) \end{array}$$

  We know that

$$(\Delta, \Delta_0 \vdash C_1) \xrightarrow{\mathsf{send}(c).n'} \stackrel{\beta}{\rightarrow}{}^* (\Delta, \Delta_0, n' \vdash \nu\Delta_1\,.\,(C_1' \,\|\, n_1[t_1] \,\|\, n'[v_1]))$$

  and we know by hypothesis that $\Delta, \Delta_0 \vDash C_1 \approx C_2$ or $\Delta, \Delta_0 \vDash C_1 \,\|\, n[\tau_a] \approx C_2 \,\|\, n[\tau_a]$. In, for instance, the latter case we find matching transitions

$$(\Delta, \Delta_0 \vDash C_2 \,\|\, n[\tau_a]) \Longrightarrow (\Delta, \Delta_0 \vDash \nu\Delta_2\,.\,(C_2' \,\|\, n[\tau_a] \,\|\, n'[v_2]))$$

say, with

$$\Delta, \Delta_0, n' \vDash \nu\Delta_1 \,.\, (C_1' \,\|\, n_1[t_1] \,\|\, n[\tau_a] \,\|\, n'[v_1]) \approx \nu\Delta_2 \,.\, (C_2' \,\|\, n[\tau_a] \,\|\, n'[v_2]) \tag{11}$$

We note that, if v_1 is of base type then the value transitions at n' guarantee that v_2 is identical to v_1 and by Lemma 5.2 we have

$$\Delta, \Delta_0 \vDash \nu\Delta_1 \,.\, (C_1' \,\|\, n_1[t_1] \,\|\, n[\tau_a]) \approx \nu\Delta_2 \,.\, (C_2' \,\|\, n[\tau_a]). \tag{12}$$

Note that, for the sake of simplicity, we use the new name transitions to allow us to assume that any name in v_1 has already been extruded. Now, Lemma 5.5 tells us that

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \,\|\, C)) \Rightarrow\approx (\Delta \vdash \nu\Delta_0 \,.\, (\nu\Delta_2 \,.\, (C_2') \,\|\, \nu\Delta_0' \,.\, (C_0' \,\|\, n_0[t_0[v_2/x]])))$$

(call the target term D_2), and by noticing that

$$D_1 \xrightarrow{\beta}{}^{*} \nu\Delta_0 \,.\, (\nu\Delta_1 \,.\, (C_1' \,\|\, n_0[t_0]) \,\|\, \nu\Delta_0' \,.\, (C_0' \,\|\, n_0[t_0[v_1/x]]))$$

along with the fact that $v_1 \equiv v_2$, (12) tells us

$$\Delta \vDash D_1 \xrightarrow{\beta}{}^{*} \mathcal{R} \approx D_2$$

as required.

Otherwise, v_1 must be an abstraction and

$$D_1 \xrightarrow{\beta}{}^{*} \nu\Delta_0, a' \,.\, (\nu\Delta_1, l \,.\, (C_1' \,\|\, n_1[t_1] \,\|\, l[a' \Leftarrow v_1]) \,\|\, \nu\Delta_0' \,.\, (C_0' \,\|\, n_0[t_0[\tau_{a'}/x]])).$$

We use Lemmas 5.3 and 5.2 to see that

$$\Delta, \Delta_0, n' \vDash \nu\Delta_1 \,.\, (C_1', l \,\|\, n_1[t_1] \,\|\, n[\tau_a] \,\|\, l[a' \Leftarrow v_1]) \approx \nu\Delta_2, l \,.\, (C_2' \,\|\, n[\tau_a] \,\|\, l[a' \Leftarrow v_2]) \tag{13}$$

Again, Lemma 5.5 tells us that

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \,\|\, C)) \Rightarrow\approx (\Delta \vdash \nu\Delta_0, a' \,.\, (\nu\Delta_2, l \,.\, (C_2' \,\|\, l[a' \Leftarrow v_2]) \,\|\, \nu\Delta_0' \,.\, (C_0' \,\|\, n_0[t_0[\tau_{a'}/x]])))$$

(call the target term D_2). Thus by using (13) we see that $\Delta \vDash D_1 \xrightarrow{\beta}{}^{*} \mathcal{R} \approx D_2$ as required.

- Suppose

$$\begin{array}{rcl} C_1 & \equiv & \nu\Delta_1 \,.\, (C_1' \,\|\, n_1[\mathsf{let}\, x = \mathsf{recv}\, c \,\mathsf{in}\, t_1]) \\ C & \equiv & \nu\Delta_0' \,.\, (C_0' \,\|\, n_0[\mathsf{let}\, () = \mathsf{send}\, (c, v) \,\mathsf{in}\, t_0]) \\ D_1 & \xrightarrow{\beta}{}^{*} & \nu\Delta_0, \Delta_1, \Delta_0' \,.\, (C_1' \,\|\, n_1[\mathsf{let}\, x = v \,\mathsf{in}\, t_1] \,\|\, C_0' \,\|\, n_0[t_0]) \end{array}$$

Again, we must consider whether v_1 is of base or higher type. We demonstrate only the latter as the arguments for both are very similar. We observe immediately a further β-reduction from D_1 such that

$$D_1 \xrightarrow{\beta}{}^{*} \nu\Delta_0, a' \,.\, (\nu\Delta_1 \,.\, (C_1' \,\|\, n_1[t_1[\tau_{a'}/x]]) \,\|\, \nu\Delta_0', l \,.\, (C_0' \,\|\, n_0[t_0] \,\|\, l[a' \Leftarrow v]))$$

We know that

$$(\Delta, \Delta_0, a' \vdash C_1) \xrightarrow{\mathsf{recv}(c,\tau_{a'})} \approx (\Delta, \Delta_0, a' \vdash \nu\Delta_1' \,.\, (C_1' \parallel n_1[t_1[\tau_{a'}/x]]))$$

so, by the hypothesis that

$$\Delta, \Delta_0 \vDash C_1 \parallel n[\tau_a] \approx C_2 \parallel n[\tau_a],$$

say, we also know that there exists some

$$(\Delta, \Delta_0, a' \vdash C_2 \parallel n[\tau_a]) \xRightarrow{\mathsf{recv}(c,\tau_{a'})} (\Delta, \Delta_0, a' \vdash \nu\Delta_2' \,.\, (C_2' \parallel n[\tau_a])$$

such that

$$\Delta, \Delta_0, a' \vDash \nu\Delta_1' \,.\, (C_1' \parallel n_1[t_1[\tau_{a'}/x]] \parallel n[\tau_a]) \approx \nu\Delta_2' \,.\, (C_2' \parallel n[\tau_a]) \tag{14}$$

We use Lemma 5.5 to observe that

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \parallel C)) \Rightarrow\approx (\Delta \vdash \nu\Delta_0, a' \,.\, (\nu\Delta_2' \,.\, C_2' \parallel \nu\Delta_0', l \,.\, (C_0' \parallel n_0[t_0] \parallel l[a' \Leftarrow v])))$$

call the target term D_2. We use (14) to conclude that $\Delta \vdash D_1 \xrightarrow{\beta}{}^* \mathcal{R}\ D_2$.

- Suppose

$$\begin{array}{rcl} C_1 & \equiv & \nu\Delta_1 \,.\, (C_1' \parallel n_1[v_1]) \\ C & \equiv & \nu\Delta_0' \,.\, (C_0' \parallel n_0[\mathsf{let}\, x = \mathsf{join}\, n_1 t_0 \,\mathsf{in}\,]) \\ D_1 & \equiv & \nu\Delta_0, \Delta_1, \Delta_0' \,.\, (C_1' \parallel n_1[v_1] \parallel C_0' \parallel n_0[\mathsf{let}\, x = v_1 \,\mathsf{in}\, t_0]) \end{array}$$

We know that $(\Delta, \Delta_0 \vdash C_1) \xrightarrow{n_1 \Downarrow} (\Delta, \Delta_0, \vdash C_1)$ and that, by hypothesis,

$$\Delta, \Delta_0 \vDash C_1 \parallel n[\tau_a] \approx C_2 \parallel n[\tau_a],$$

say. This means that there must exist some

$$(\Delta, \Delta_0 \vdash C_2 \parallel n[\tau_a]) \xRightarrow{n_1 \Downarrow} (\Delta, \Delta_0 \vdash \nu\Delta_2 \,.\, (C_2' \parallel n_1[v_2] \parallel n[\tau_a]))$$

with

$$\Delta, \Delta_0 \vDash C_1 \parallel n[\tau_a] \approx (\Delta, \Delta_0 \vdash \nu\Delta_2 \,.\, (C_2' \parallel n_1[v_2] \parallel n[\tau_a])).$$

This tells us that if v_1 is of base type then it is clear that v_2 must be identical to it. We can then use Lemma 5.5 to obtain the matching transitions required fairly easily. However, if v_1 is an abstraction then we we can apply Lemma 5.3 to obtain

$$\Delta, \Delta_0, a' \vDash \nu\Delta_1 \,.\, (C_1', l \parallel n_1[v_1] \parallel l[a' \Leftarrow v_1] \parallel n[\tau_a]) \approx \nu\Delta_2, l \,.\, (C_2' \parallel n_1[v_2] \parallel l[a' \Leftarrow v_2] \parallel n[\tau_a]) \tag{15}$$

Lemma 5.5 allows us to observe

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \parallel C)) \Rightarrow\approx (\Delta \vdash \nu\Delta_0, a' \,.\, (\nu\Delta_2, l \,.\, (C_2' \parallel n_1[v_2] \parallel l[a' \Leftarrow v_2]) \parallel \nu\Delta_0' \,.\, (C_0' \parallel n_0[t_0[\tau_{a'}/x]])))$$

(call the target term D_2). Therefore, as

$$D_1 \xrightarrow{\beta}{}^* \nu\Delta_0, a' \,.\, (\nu\Delta_1, l \,.\, (C_1' \parallel n_1[v_1] \parallel l[a' \Leftarrow v_1]) \parallel \nu\Delta_0 \,.\, (C_0' \parallel n_0[t_0[\tau_{a'}/x]]))$$

we can use (15) to see that $\Delta \vDash D_1 \xrightarrow{\beta}{}^* \mathcal{R} \approx D_2$.

- Suppose

$$\begin{array}{rcl} C_1 & \equiv & \nu\Delta_1 \,.\, (C_1' \parallel n_1[\mathsf{let}\, x = \mathsf{join}\, n_0\, \mathsf{in}\, t_1]) \\ C & \equiv & \nu\Delta_0' \,.\, (C_0' \parallel n_0[v]) \\ D_1 & \equiv & \nu\Delta_0, \Delta_1, \Delta_0' \,.\, (C_1' \parallel n_1[\mathsf{let}\, x = v\, \mathsf{in}\, t_1] \parallel C_0' \parallel n_0[v]) \end{array}$$

Suppose firstly that v is of base type. We know that it must be the case that our hypothesis is $\Delta, \Delta_0 \vdash C_1 \parallel n[\tau_a] \approx C_2 \parallel n[\tau_a]$ with no thread at n_0 defined in C_1 or C_2. Note that, if v is not typable in environment Δ, Δ_0 then we must weaken this environment by some name $(v : \sigma)$ from Δ_0' and use

$$(\Delta, \Delta_0, (v:\sigma) \vdash C_1 \parallel n[\tau_a]) \xrightarrow{\mathsf{join}(v).n_0} \xrightarrow{\beta}{}^{*} (\Delta, \Delta_0, (v:\sigma) \vdash \nu\Delta_1 \,.\, (C_1' \parallel n_1[\mathsf{let}\, x = v\, \mathsf{in}\, t_1] \parallel n[\tau_a] \parallel n_0[v]))$$

and our hypothesis to tell us that there exist transitions

$$(\Delta, \Delta_0, (v:\sigma) \vdash C_2 \parallel n[\tau_a]) \xRightarrow{\mathsf{join}(v).n_0} (\Delta, \Delta_0, (v:\sigma) \vdash \nu\Delta_2' \,.\, (C_2' \parallel n[\tau_a] \parallel n_0[v]))$$

such that

$$\Delta, \Delta_0, (v:\sigma) \vDash \nu\Delta_1 \,.\, (C_1' \parallel n_1[\mathsf{let}\, x = v\, \mathsf{in}\, t_1] \parallel n_0[v] \parallel n[\tau_a]) \approx \nu\Delta_2' \,.\, (C_2' \parallel n_0[v] \parallel n[\tau_a]) \qquad (16)$$

It should be clear by Lemma 5.5 that

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \parallel C)) \Rightarrow\approx (\Delta \vdash \nu(\Delta_0, (v:\sigma)) \,.\, (\nu\Delta_2 \,.\, (C_2' \parallel n_0[v]) \parallel \nu(\Delta_0' \setminus (v:\sigma)) \,.\, C_0'))$$

and we call the target of this D_2. Thus, by (16), we see that $\Delta \vDash D_1 \xrightarrow{\beta}{}^{*} \mathcal{R} \approx D_2$.
Otherwise, v must be an abstraction. We know by hypothesis that

$$\Delta, \Delta_0 \vDash C_1 \parallel n[\tau_a] \approx C_2 \parallel n[\tau_a].$$

There are two cases to consider here according to whether n and n_0 coincide.
Suppose that n is not n_0. We notice that

$$(\Delta, \Delta_0, a' \vdash C_1 \parallel n[\tau_a]) \xrightarrow{\mathsf{join}(\tau_{a'}).n_0} \approx (\Delta, \Delta_0, a' \vdash \nu\Delta_1 \,.\, (C_1' \parallel n_1[t_1[\tau_{a'}/x]] \parallel n[\tau_a] \parallel n_0[\tau_{a'}]))$$

must be matched by some

$$(\Delta, \Delta_0, a' \vdash C_2 \parallel n[\tau_a]) \xRightarrow{\mathsf{join}(\tau_{a'}).n_0} (\Delta, \Delta_0, a' \vdash \nu\Delta_2 \,.\, (C_2' \parallel n[\tau_a] \parallel n_0[\tau_{a'}]))$$

such that

$$\Delta, \Delta_0, a' \vDash \nu\Delta_1 \,.\, (C_1' \parallel n_1[t_1[\tau_{a'}/x]] \parallel n[\tau_a] \parallel n_0[\tau_{a'}]) \approx \nu\Delta_2 \,.\, (C_2' \parallel n[\tau_a] \parallel n_0[\tau_{a'}]) \qquad (17)$$

We now use Lemma 5.5 to observe that

$$(\Delta \vdash \nu\Delta_0 \,.\, (C_2 \parallel C)) \Rightarrow\approx (\Delta \vdash \nu\Delta_0, a' \,.\, (\nu\Delta_2 \,.\, C_2' \parallel \nu\Delta_0', l \,.\, (C_0' \parallel n_0[v] \parallel l[a' \Leftarrow v])))$$

(call the target of this D_2). We should point out that the subterm $l[a' \Leftarrow v]$ in the target here is either created by some derivate of C_2 performing a $\mathsf{join}\, n_0$ command and a subsequent substitution, or, in the absence of this, we can insert it artificially to obtain a term which is weakly bisimilar to the actual derivate of $\nu\Delta_0 \,.\, (C_2 \,\|\, C)$). In either case, because

$$D_1 \xrightarrow{\beta}{}^{*} \nu\Delta_0, a' \,.\, (\nu\Delta_1 \,.\, (C_1' \,\|\, n_1[t_1[\tau_{a'}/x]]) \,\|\, \nu\Delta_0', l \,.\, (C_0' \,\|\, n_0[v] \,\|\, l[a' \Leftarrow v]))$$

we see by (17) that $\Delta \vDash D_1 \xrightarrow{\beta}{}^{*} \mathcal{R} \approx D_2$ as required.

To finish we note that if n is equal to n_0 then we may proceed as above but rather than using a $\xrightarrow{\mathsf{join}(\tau_{a'}).n_0}$ transition to obtain a match in C_2 we simply use the internal join communication along n in $C_1 \,\|\, n[\tau_a]$. □

6 Example

We will present an example to demonstrate the usefulness of our canonical semantics in reasoning about program equivalence. The example is formed by considering a simple persistent server which repeatedly accepts programs of type $\mathsf{unit} \to \mathsf{bool}$ along a request channel named req. It then runs the program by applying it to the unit value and returns the result along an acknowledgement channel named ack.

As the server itself is stateless and is repeatedly and persistently available then deploying the same server twice should be (barbed) equivalent to deploying it a single time. We aim to prove this using the canonical semantics.

Define

$$\begin{array}{lcl} \mathsf{instance} & \stackrel{def}{=} & \mathsf{let}\, x = \mathsf{recv}\, \mathsf{req}\, \mathsf{in}\, \mathsf{let}\, z = x()\, \mathsf{in}\, \mathsf{send}\, (\mathsf{ack}, z) \\ \mathsf{spawn\_instance} & \stackrel{def}{=} & \mathsf{spawn}\, (\lambda() \,.\, \mathsf{instance}); f() \\ \mathsf{start\_server} & \stackrel{def}{=} & \mathsf{spawn}\, (\lambda() \,.\, \mathsf{let}\, x = (\mu f \,.\, \lambda() \,.\, \mathsf{spawn\_instance})\, \mathsf{in}\, x()) \end{array}$$

We will outline a proof that

$$\Delta \vDash n[\mathsf{start\_server}] \approx^{b} n[\mathsf{start\_server}; \mathsf{start\_server}]$$

where Δ is $n : \mathsf{unit\,thread}, \mathsf{req} : (\mathsf{unit} \to \mathsf{bool})\,\mathsf{chan}, \mathsf{ack} : \mathsf{bool\,chan}$.

A direct proof of this statement using barbed bisimulations is extremely difficult as it must quantify over all possible programs which can be passed in to the server and we invite the reader to attempt this. However, by making use of various Lemmas and our characterisation of $\approx^{b}$ in the canonical semantics, this is achievable. The full details are still rather complicated and would be best suited to machine checking rather than writing them out explicitly here. We do indicate how the proof proceeds though. Firstly, we notice that $\mathsf{start\_server}$ does not contain instances of the join operator. This allows us to simplify things by using the following lemma

Lemma 6.1 *If C_1 and C_2 do not contain* join *and $\Delta \vDash C_1 \approx C_2$ holds in the labelled transition system without* $\mathsf{join}(v) \,.\, n$ *labels then $\Delta \vDash C_1 \approx C_2$ holds generally.*

Proof: Straightforward coinduction using the canonical semantics. □

In order to define a witness bisimulation we identify (up to β-equivalence and garbage collection (Lemma 3.7)) the states which the server can get in to after performing labelled actions. Let

$$\begin{array}{lcl} U(c_0) & \stackrel{def}{=} & \mathsf{let}\, g = \mathsf{recv}\, c_o \,\mathsf{in}\, (\mathsf{send}\,(c_0, g) \parallel g\,()) \\ U'(c_0) & \stackrel{def}{=} & \mathsf{send}\,(c_0, \lambda()\,.\,\mathsf{spawn\_instance}[\lambda()\,.\,U(c_0)/f]) \end{array}$$

By considering the definition of start_server and that of recursive functions, it is easy to check that

$$n[\mathsf{start\_server}] \stackrel{\beta}{\rightarrow}{}^{*}\ \nu(c_0, n_0, n_0', m_0)\,.\,(n_0[U(c_0)] \parallel n_0'[U'(c_0)] \parallel m_0[\mathsf{instance}])$$

up to the removal of garbage terms such as $n[()]$. Call the target of these reductions C_0. We can also check that $n[\mathsf{start\_server}; \mathsf{start\_server}] \stackrel{\beta}{\rightarrow}{}^{*} C_0 \parallel C_0$ and we can then reduce our obligation to showing

$$\Delta \vDash C_0 \approx C_0 \parallel C_0.$$

We proceed by building a witness bisimulation and showing that it forms a bisimulation up to $(\stackrel{\beta}{\rightarrow}{}^{*}, \approx)$. In order to do this we define parameterised configurations

$$\begin{array}{lcl} D_0 & \stackrel{def}{=} & \nu m\,.\,m[\mathsf{instance}] \\ D_1(a) & \stackrel{def}{=} & \nu(m,r)\,.\,m[\mathsf{let}\, z = \mathsf{send}\,(a, ((), r)); \mathsf{recv}\, r \,\mathsf{in}\, \mathsf{send}\,(\mathsf{ack}, z)] \\ D_2(o) & \stackrel{def}{=} & \nu(m,r)\,.\,(m[\mathsf{let}\, z = \mathsf{recv}\, r \,\mathsf{in}\, \mathsf{send}\,(\mathsf{ack}, z)] \parallel o[((), r)]) \\ D_3(\vec{o}) & \stackrel{def}{=} & \nu(m,r)\,.\,(m[\mathsf{let}\, z = \mathsf{recv}\, r \,\mathsf{in}\, \mathsf{send}\,(\mathsf{ack}, z)] \parallel o[((), r)] \parallel \vec{o}'[()]^{*} \parallel \vec{o}''[r]^{*}) \\ D_4(\vec{o}, r) & \stackrel{def}{=} & \nu m\,.\,(m[\mathsf{let}\, z = \mathsf{recv}\, r \,\mathsf{in}\, \mathsf{send}\,(\mathsf{ack}, z)] \parallel o[((), r)] \parallel \vec{o}'[()]^{*} \parallel \vec{o}''[r]^{+}) \\ D_5(\vec{o}, r, b) & \stackrel{def}{=} & \nu m\,.\,(m[\mathsf{send}\,(\mathsf{ack}, z)] \parallel o[((), r)] \parallel \vec{o}'[()]^{*} \parallel \vec{o}''[r]^{+}) \\ D_6(\vec{o}, r, b,) & \stackrel{def}{=} & o[((), r)] \parallel o'''[b] \parallel \vec{o}'[()]^{*} \parallel \vec{o}''[r]^{+} \end{array}$$

where D_3 is defined using the notation $\vec{o}[v]^{*}$ meaning zero or more occurrences of this thread with a different name used from $\vec{o}$ for each and D_4, D_5, D_6 use a similar suggestive notation using $+$ to indicate one or more occurrences of the thread. The meta-variable b ranges over boolean values. Now, we say that a configuration C is a *server configuration* if

$$C \equiv \nu(c_0, n_0, n_0')\,.\,(n_0[U(c_0)] \parallel n_0'[U'(c_0)] \parallel \prod_{i \in I} C_i)$$

where each C_i is one of the forms $D_i(\vec{p})$ listed above with a, r and $\vec{o}$ distinct for each. Let $\mathcal{R}$ be defined such that

$$\Delta \vDash C_L\ \mathcal{R}\ C_R' \parallel C_R''$$

whenever $\Delta \vdash C_L, C_R', C_R''$ are well-typed server configurations such that

$$D_i(\vec{p}) \in C_L \text{ if and only if } (D_i(\vec{p}) \in C_R' \text{ or } D_i(\vec{p}) \in C_R'')$$

It is clear that C_0 is related to $C_0 \parallel C_0$ by $\mathcal{R}$.

To show that $\mathcal{R}$ forms a bisimulation up to $(\xrightarrow{\beta}{}^*, \approx)$ we consider what actions are possible from server configurations. There are β reductions arising from communication between $U(c_0)$ and $U'(c_0)$ but these simply result in spawning another $\nu m\,.\,m[\mathsf{instance}]$ resulting in a server configuration. Thus $\mathcal{R}$ is unaffected by these. Weakening actions are also always available from server configurations and are easily matched by $\mathcal{R}$ related terms. Other actions arise from the $D_i(\vec{p})$ components in the following ways: Let α_i and β_i range over the actions as detailed below.

$$\begin{array}{ll}
\alpha_0 ::= \mathsf{recv}(\mathsf{req}, \tau_a) & \beta_0 ::= o \Downarrow \\
\alpha_1 ::= \mathsf{send}(a)\,.\,o & \beta_1 ::= \beta_0 \mid o\,.\,\mathsf{fst}\,.\,o' \mid o\,.\,\mathsf{snd}\,.\,o'' \\
\alpha_2 ::= o\,.\,\mathsf{fst}\,.\,o' \mid o\,.\,\mathsf{snd}\,.\,o'' & \beta_2 ::= \beta_1 \mid o''\,.\,r \\
\alpha_3 ::= o''\,.\,\nu r & \beta_3 ::= \beta_2 \mid o'''\,.\,b \\
\alpha_4 ::= \mathsf{recv}(r, b) & \\
\alpha_5 ::= \mathsf{send}(\mathsf{ack})\,.\,b &
\end{array}$$

It is easy to check that $D_i(\vec{p}) \xrightarrow{\alpha_i} \xrightarrow{\beta}{}^* D_{i+1}(\vec{p}')$ for $1 \leq i \leq 5$. Note that the definition of τ_a is expanded when moving to D_1. We also have

$$D_2 \xrightarrow{\beta_0} D_2 \qquad D_3 \xrightarrow{\beta_1} D_3 \qquad D_4 \xrightarrow{\beta_2} D_4 \qquad D_5 \xrightarrow{\beta_2} D_5 \qquad D_6 \xrightarrow{\beta_3} D_6$$

These identify all possible named actions from the D_i terms except for β, $\mathsf{join}(v)\,.\,n$ and weakening actions. From this, we now consider $\Delta \vDash C_L \;\mathcal{R}\; C'_R \,\|\, C''_R$ and try match every non-β, $\mathsf{join}(v)\,.\,n$ and weakening action. If $(\Delta \vDash C_L) \xrightarrow{\alpha} (\Delta' \vDash C_L)$ then it must be that α derived from a $D_i(\vec{p})$ component. In which case, it is one of the α_i or β_j actions listed above. By definition, there must be a corresponding $D_i(\vec{p})$ in C'_R or C''_R which can be used to match the action. The resulting configurations are still $\mathcal{R}$ related as the α_i and β_i actions listed above preserve the property that $D_i(\vec{p}) \in C_L$ if and only if $D_i(\vec{p}) \in C'_R$ or C''_R. A symmetric argument to match actions from $C'_R \,\|\, C''_R$ can be used to finish.

7 Concluding remarks

We have developed an operational account of program equivalence for a fragment of Concurrent ML which features higher-order functions, concurrency primitives and statically-scoped local names. The bisimulation equivalence, and in particular that for the canonical semantics provide a lightweight characterisation of barbed equivalence in this setting. This is the first such treatment for a language containing all of these features.

The proof techniques employed here owe much to Sangiorgi [29] and we consider the hierarchical approach to trigger correctness a useful generalization of Sangiorgi's method to the functional setting. Indeed such techniques could be employed in any functional language sufficiently expressive to encode the trigger passing mechanism. We have also identified a useful 'bisimulation up to' technique based on confluent reduction.

There is a striking relationship between location based mobile agent languages in the sense of [4, 27, 32] and the thread identifiers. It could be fruitful to adapt the techniques used here to such a setting. In particular, trigger encodings could address the issues of migrating processes and scope in much the same way they help us achieve congruence here. However, such languages often include features such as currentthread or thread death which depend on the name of the current thread identifier. An attempt to cope with currentthread is described in [17].

References

[1] K.L. Bernstein and E.W. Stark. Operational semantics of a focussing debugger. In *Proc. MFPS 95*. Springer-Verlag, 1995. Vol 1. Electronic Notes in Comp. Sci.

[2] G. Boudol. Asynchrony and the π-calculus. Technical Report 1702, INRIA, Sophia-Antipolis, 1991.

[3] G. Boudol. The π-calculus in direct style. *Higher-order and Symbolic Computation*, 11, 1998.

[4] L. Cardelli and A. Gordon. Mobile ambients. In *Proc. FoSSaCS '98*, LNCS. Springer-Verlag, 1998.

[5] R. Milner D. Berry and D. Turner. A Semantics for ML Concurrency Primitives. In *Proceedings of the 19th ACM Symposium on Principles of Programmings Languages*, 1992.

[6] W. Ferreira, M. Hennessy, and A.S.A Jeffrey. A theory of weak bisimulation for core CML. *J. Functional Programming*, 8(5):447–491, 1998.

[7] C. Fournet and G. Gonthier. The reflexive CHAM and the join-calculus. In *Proc. ACM-POPL*, 1996.

[8] C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. In *Proc. ICALP*, volume 1443 of *Lecture notes in computer science*. Springer-Verlag, 1998.

[9] C. Fournet, G. Gonthier, J-J. Levy, L. Maranget, and D. Remy. A calculus of mobile agents. In *Proc. CONCUR*, volume 1119 of *Lecture notes in computer science*. Springer-Verlag, 1996.

[10] A. Gordon. Bisimilarity as a theory of functional programming. In *Proc. MFPS 95*, number 1 in Electronic Notes in Comp. Sci. Springer-Verlag, 1995.

[11] A. Gordon and P.D. Hankin. A concurrent object calculus. In *Proc. HLCL*, 1998.

[12] K. Honda and M. Tokoro. An object calculus for asynchronous communication. In *Proc. ECOOP 91, Geneve*, 1991.

[13] K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.

[14] Douglas Howe. Equality in lazy computation systems. In *Proc. IEEE Logic in Computer Science*, pages 198–203. IEEE Computer Society Press, 1989.

[15] A.S.A. Jeffrey and J. Rathke. Towards a theory of bisimilarity for local names. In *Proc. IEEE Logic in Computer Science*, pages 56–66. IEEE Computer Society Press, 1999.

[16] A.S.A. Jeffrey and J. Rathke. Towards a theory of bisimilarity for local names. In *Proc. IEEE Logic in Computer Science*, pages 311–321. IEEE Computer Society Press, 2000.

[17] A.S.A. Jeffrey and J. Rathke. A fully abstract may testing semantics for concurrent objects. In *Proc. IEEE Logic in Computer Science*, pages 101–112. IEEE Computer Society Press, 2002.

[18] J. J. Leifer and R. Milner. Deriving bisimulation congruences for reactive systems. In C. Palamidessi, editor, *Proc. CONCUR 2000*, volume 1877 of *Lecture notes in Computer Science*. Springer-Verlag, 2000.

[19] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture notes in Computer Science*. Springer-Verlag, 1980.

[20] R. Milner. *Communication and mobile systems: the π-calculus*. Cambridge University Press, 1999.

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

[22] R. Milner and D. Sangiorgi. Barbed bisimulation. In *Proc. ICALP*, volume 623 of *Lecture notes in Computer Science*. Springer-Verlag, 1992.

[23] E. Moggi. Notions of computation and monads. *Information and Computing*, 93:55–92, 1991.

[24] A. M. Pitts and I. D. B. Stark. Observable properties of higher order functions that dynamically create local names, or: What's new? In *Proc. MFCS 93*, pages 122–141. Springer-Verlag, 1993. LNCS 711.

[25] A.M. Pitts and I.D.B. Stark. Operational reasoning for functions with local state. In A.D. Gordon and A.M. Pitts, editors, *Higher Order Operational Techniques in Semantics*, pages 227–273. Cambridge University Press, 1998. Publications of the Newton Institute.

[26] J. Reppy. *Higher-Order Concurrency*. PhD thesis, Cornell University, 1992. Technical Report TR 92-1285.

[27] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Proc. POPL*. ACM Press, 1998.

[28] M.Tofte R.Milner and R.Harper. *The Defi nition of Standard ML*. MIT Press, 1990.

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

[30] D. Sangiorgi and R. Milner. On the problem of 'weak bisimulation up to'. In *Proc. CONCUR*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46. Springer-Verlag, 1992.

[31] P. Sewell. From rewrite rules to bisimulation congruences. In *Proc. CONCUR*, volume 1466 of *Lecture Notes in Computer Science*, pages 269–284. Springer-Verlag, 1992.

[32] P. Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Proc. ICALP*, volume 1443 of *Lecture Notes in Computer Science*, pages 695–706. Springer-Verlag, 1998.

[33] B. Thomsen. *Calculi for Higher-Order Communicating Systems*. PhD thesis, University of London, 1990.