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

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-

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

$$\frac{\Gamma;\Delta \vdash \boldsymbol{v}_1:\sigma\,\mathsf{thread} \qquad \Gamma;\Delta \vdash \boldsymbol{v}_2:\sigma\,\mathsf{thread}}{\Gamma;\Delta \vdash \boldsymbol{v}_1=\boldsymbol{v}_2:\mathsf{bool}} \qquad \frac{\Gamma;\Delta \vdash \boldsymbol{v}_1:\sigma\,\mathsf{chan} \qquad \Gamma;\Delta \vdash \boldsymbol{v}_2:\sigma\,\mathsf{chan}}{\Gamma;\Delta \vdash \boldsymbol{v}_1=\boldsymbol{v}_2:\mathsf{bool}}$$

$$\frac{\Gamma;\Delta \vdash \boldsymbol{v}_1:\boldsymbol{B} \qquad \Gamma;\Delta \vdash \boldsymbol{v}_2:\boldsymbol{B}}{\Gamma;\Delta \vdash \boldsymbol{v}_1=\boldsymbol{v}_2:\mathsf{bool}} \qquad \frac{\Gamma;\Delta \vdash \boldsymbol{v}:\mathsf{bool} \qquad \Gamma;\Delta \vdash \boldsymbol{t}_1:\sigma \qquad \Gamma;\Delta \vdash \boldsymbol{t}_2:\sigma}{\Gamma;\Delta \vdash \mathsf{if}\,\boldsymbol{v}\,\mathsf{then}\,\boldsymbol{t}_1\,\mathsf{else}\,\boldsymbol{t}_2:\sigma}$$

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

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

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

$$\frac{;\Delta,\boldsymbol{n}:\sigma\,\mathsf{thread} \vdash \boldsymbol{t}:\sigma}{\Delta,\boldsymbol{n}:\sigma\,\mathsf{thread} \vdash \boldsymbol{n}\,\boldsymbol{t}} \qquad \frac{}{\Delta \vdash \boldsymbol{0}} \qquad \frac{\Delta \vdash \boldsymbol{C}_1 \qquad \Delta \vdash \boldsymbol{C}_2}{\Delta \vdash \boldsymbol{C}_1 \otimes \boldsymbol{C}_2} \qquad \frac{\Delta,\boldsymbol{n}:\sigma \vdash \boldsymbol{C}}{\Delta \vdash \nu\boldsymbol{n}:\sigma.\boldsymbol{C}}$$

Figure 2: Configuration type inference rules

including at least the unit type unit and the boolean type bool. The grammar of types is given:

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

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

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

Threads take the form $\mathsf{let}\,\boldsymbol{x}_1=\boldsymbol{e}_1\,\mathsf{in}\cdots\mathsf{let}\,\boldsymbol{x}_n=\boldsymbol{e}_n\,\mathsf{in}\,\boldsymbol{v}$ and consist of a stack of expressions $\boldsymbol{e}_1,\ldots,\boldsymbol{e}_n$ to be evaluated, followed by a return value $\boldsymbol{v}$. The grammar of $\boldsymbol{\mu}\nu$CML threads is given:

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

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.
- $\mathsf{send}(\boldsymbol{c},\boldsymbol{v})$ sends value $\boldsymbol{v}$ along channel $\boldsymbol{c}$ to a matching expression $\mathsf{recv}\,\boldsymbol{c}$, which returns $\boldsymbol{v}$.

$$
\begin{array}{rcll}
\mathbf{0} \parallel \boldsymbol{C} & \equiv & \boldsymbol{C} & \\
(\boldsymbol{C}_1 \parallel \boldsymbol{C}_2 \parallel \boldsymbol{C}_3 & \equiv & \boldsymbol{C}_1 \parallel (\boldsymbol{C}_2 \parallel \boldsymbol{C}_3 & \\
\boldsymbol{C}_1 \parallel \boldsymbol{C}_2 & \equiv & \boldsymbol{C}_2 \parallel \boldsymbol{C}_1 & \\
\boldsymbol{C}_1 \parallel \nu \boldsymbol{n} . \boldsymbol{C}_2 & \equiv & \nu \boldsymbol{n} . (\boldsymbol{C}_1 \parallel \boldsymbol{C}_2 & (\boldsymbol{n} \notin \boldsymbol{C}_1 \\
\nu \boldsymbol{n} . \nu \boldsymbol{n}' . \boldsymbol{C} & \equiv & \nu \boldsymbol{n}' . \nu \boldsymbol{n} . \boldsymbol{C} &
\end{array}
$$

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

- $\mathsf{spawn}\,\boldsymbol{v}$ creates a new named thread, which executes $\boldsymbol{v}(\ )$, and returns the thread identifier.
- $\mathsf{join}\,\boldsymbol{i}$ blocks waiting for the thread with identifier $\boldsymbol{i}$ to terminate with value $\boldsymbol{v}$, which is then returned (this is similar to Reppy's [23] ***joinVal*** function).

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

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

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

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

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

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

where $\boldsymbol{x}$ does not occur free in $\boldsymbol{e}'$.

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

$$\Gamma; \Delta \parallel \boldsymbol{t} : \sigma$$

where Γ is the type context for free variables and Δ the type context for free names.

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 $\parallel$ to express concurrency and the configuration $\mathbf{0}$ represents the empty configuration and forms a unit for $\parallel$. We use the scoping operator $\nu \boldsymbol{n} : \sigma . \cdot$ to delimit the portion of the configuration in which the identifier $\boldsymbol{n}$ is deemed to exist. The grammar for configurations is as follows:

$$\boldsymbol{C} ::= \mathbf{0} \mid \boldsymbol{C} \parallel \boldsymbol{C} \mid \nu \boldsymbol{n} : \sigma . \boldsymbol{C} \mid \boldsymbol{n}\,\boldsymbol{t}$$

Let the thread names of a configuration be defined:

$$
\begin{array}{rclrcl}
\boldsymbol{tn}(\mathbf{0} & = & \emptyset & \boldsymbol{tn}(\boldsymbol{C}_1 \parallel \boldsymbol{C}_2 & = & \boldsymbol{tn}(\boldsymbol{C}_1 \parallel \boldsymbol{tn}(\boldsymbol{C}_2 \\
\boldsymbol{tn}(\boldsymbol{n}\,\boldsymbol{t} & = & \{\boldsymbol{n}\} & \boldsymbol{tn}(\nu \boldsymbol{n} . \boldsymbol{C} & = & \boldsymbol{tn}(\boldsymbol{C} \setminus \{\boldsymbol{n}\}
\end{array}
$$

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

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 [30] 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 [26] showed this 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 [28] and Leifer and Milner [16].

Our labelled transition system is defined as a relation between well-typed configurations. The rules are presented in Figure 5 but we elide type information for thread identifiers. 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 $(\Delta \Box \boldsymbol{C} \xrightarrow{\beta} (\Delta \Box \boldsymbol{C}'$ holds whenever $\boldsymbol{C} \xrightarrow{\beta} \boldsymbol{C}'$ holds (and similarly for τ).

Let α range over γ, τ and β transitions. We define $(\Delta \Box \boldsymbol{C} \overset{\alpha}{=\Rightarrow} \Box (\Delta' \Box \boldsymbol{C}'$ as $(\Delta \Box \boldsymbol{C} \Box \overset{\alpha}{\rightarrow} \Box (\Delta' \Box \boldsymbol{C}'$ and $(\Delta \Box \boldsymbol{C} \overset{\hat{\alpha}}{=\Rightarrow} \Box (\Delta' \Box \boldsymbol{C}'$ as $(\Delta \Box \boldsymbol{C} \Box (\Delta' \Box \boldsymbol{C}'$ when α is β or τ and $(\Delta \Box \boldsymbol{C} \overset{\alpha}{=\Rightarrow} \Box (\Delta' \Box \boldsymbol{C}'$ otherwise.

The labels used take various forms, many are prepended with an identifier, for example, $\xrightarrow{\boldsymbol{n}.\boldsymbol{b}}$. This signifies which named thread we are currently investigating. Some are followed by another identifier, for example, $\xrightarrow{\boldsymbol{n}.\,\mathsf{st}.\boldsymbol{n}'}$ indicates that we can observe that thread $\boldsymbol{n}$ 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 $\boldsymbol{n}'$. Because the only way in which an observer may interact with thread $\boldsymbol{n}$ 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,

$$\boldsymbol{n}\ \mathsf{let}\,\boldsymbol{x} = \mathsf{join}\,\boldsymbol{n}'\,\mathsf{in}\,\mathsf{true} \;\cong\; \boldsymbol{n}\ \mathsf{let}\,\boldsymbol{x} = \mathsf{join}\,\boldsymbol{n}'\,\mathsf{in}\,\mathsf{false}$$

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 $\boldsymbol{n}\,\Box$ 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

(Δ □ ***n v***

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 follow diagram can be completed:***

$$\begin{array}{ccc} (\Delta \,\square\, \boldsymbol{C} & \xrightarrow{\beta} & (\Delta \,\square\, \boldsymbol{C}' \\ \alpha \big| & & \\ (\Delta' \,\square\, \boldsymbol{C}'' & & \end{array} \qquad \textit{as} \qquad \begin{array}{ccc} (\Delta \,\square\, \boldsymbol{C} & \xrightarrow{\beta} & (\Delta \,\square\, \boldsymbol{C}' \\ \alpha \big| & & \alpha \big| \\ (\Delta' \,\square\, \boldsymbol{C}'' & \xrightarrow{\beta} & (\Delta' \,\square\, \boldsymbol{C}''' \end{array}$$

or $\boldsymbol{C}' \equiv \boldsymbol{C}''$ ***if*** α ***is*** β**.**

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

$$\nu\Delta_0 \,.\, (\boldsymbol{C}_1 \,\square\, \boldsymbol{C}_2 \xrightarrow{\beta} \nu\Delta_0 \,.\, (\boldsymbol{C}_1' \,\square\, \boldsymbol{C}_2$$

where $\boldsymbol{C}_1 \xrightarrow{\beta} \boldsymbol{C}_1'$ is an instance of a β-reduction axiom.

Now, suppose (wlog) that $(\Delta \,\square\, \nu\Delta_0 \,.\, (\boldsymbol{C}_1 \,\square\, \boldsymbol{C}_2 \xrightarrow{\alpha} (\Delta' \,\square\, \boldsymbol{C}''$ also and $\boldsymbol{C}'$ is $\nu\Delta_0 \,.\, (\boldsymbol{C}_1' \,\square\, \boldsymbol{C}_2$. Given this, it is then easy to see by inspecting the reduction and transition axioms that, for $\alpha \boxminus \beta$, save for the case in which the join synchronisation β-reduction occurs, it must be that $(\Delta \,\square\, \nu\Delta_0 \,.\, \boldsymbol{C}_2 \xrightarrow{\alpha} (\Delta' \,\square\, \nu\Delta_0' \,.\, \boldsymbol{C}_2'$ for appropriate Δ_0'. So $\boldsymbol{C}''$ must be of the form $\nu\Delta_0' \,.\, (\boldsymbol{C}_1 \,\square\, \boldsymbol{C}_2'$ and if we let $\boldsymbol{C}'''$ be $\nu\Delta_0' \,.\, (\boldsymbol{C}_1' \,\square\, \boldsymbol{C}_2'$ we are done.

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

$$\nu\Delta_0 \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\, \boldsymbol{C}_2 \xrightarrow{\beta} \nu\Delta_0 \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \boldsymbol{v} \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\, \boldsymbol{C}_2$$

then we notice that α may be derived not only from $\boldsymbol{C}_2$ but also from $\boldsymbol{n}_2\ \boldsymbol{v}$. In this situation we also see that all observations, α, deriving from this value have the general form

$$\Delta \,\square\, \nu\Delta_0 \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\, \boldsymbol{C}_2 \xrightarrow{\alpha} \Delta' \,\square\, \nu\Delta_0' \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\, \boldsymbol{C}_2'$$

thus $\boldsymbol{n}_2\ \boldsymbol{v}$ 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 $\boldsymbol{C}_1$ (then $\boldsymbol{C}' \equiv \boldsymbol{C}''$), or it be a different reduction originating entirely in $\boldsymbol{C}_2$, in which case the two clearly commute. Alternatively, it could be an overlapping instance of the join axiom

$$\begin{array}{l} \nu\Delta_0 \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\; \boldsymbol{n}_3 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t}' \;\square\, \boldsymbol{C}_2 \\ \quad \xrightarrow{\beta} \nu\Delta_0 \,.\, (\boldsymbol{n}_1 \;\mathsf{let}\, \boldsymbol{x} = \mathsf{join}\, \boldsymbol{n}_2 \,\mathsf{in}\, \boldsymbol{t} \;\square\; \boldsymbol{n}_2 \;\boldsymbol{v} \;\square\; \boldsymbol{n}_3 \;\mathsf{let}\, \boldsymbol{x} = \boldsymbol{v} \,\mathsf{in}\, \boldsymbol{t}' \;\square\, \boldsymbol{C}_2 \end{array}$$

Again, we notice that the $\boldsymbol{n}_2\ \boldsymbol{v}$ is residual in the target term and this allows the two β-reductions to commute. □

- Suppose $\boldsymbol{C}$ is $\boldsymbol{n}\ \lambda\boldsymbol{x}.\boldsymbol{t}$ and γ is $\boldsymbol{n}.@\boldsymbol{v}.\boldsymbol{n}'$ so that Δ' is $\boldsymbol{n}' : \sigma'\ \mathsf{thread}$ (where $\Delta \uplus \boldsymbol{n} : \sigma \to \sigma'\ \mathsf{thread}$) and $\boldsymbol{C}'$ is $\boldsymbol{C} \uplus \boldsymbol{n}'\ \mathsf{let}\,\boldsymbol{x} = \boldsymbol{v}\,\mathsf{in}\,\boldsymbol{t}$. We know that $\boldsymbol{C}_\gamma^\Delta$ is $\nu\boldsymbol{n}'.\boldsymbol{m}\ \boldsymbol{n}' \uplus \boldsymbol{l}\ \mathsf{join}\,\boldsymbol{n};\mathsf{true} \uplus \boldsymbol{n}'\ \mathsf{join}\,\boldsymbol{n}\boldsymbol{v}$ and that

$$\nu\boldsymbol{n}'.\boldsymbol{m}\ \boldsymbol{n}' \uplus \boldsymbol{l}\ \mathsf{join}\,\boldsymbol{n};\mathsf{true} \uplus \boldsymbol{n}'\ \mathsf{join}\,\boldsymbol{n}\boldsymbol{v} \uplus \boldsymbol{n}\ \lambda\boldsymbol{x}.\boldsymbol{t}$$

reduces through join synchronisations to

$$\nu\boldsymbol{n}'$$

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 $\gamma = \boldsymbol{n}\,.\,\nu\boldsymbol{n}'$ (with $\Delta' = \boldsymbol{n}'$) then, by hypothesis,

  $$\boldsymbol{m}\ \mathsf{join}\boldsymbol{n}\ \boldsymbol{l}\ \mathsf{join}\boldsymbol{n}\ \Delta\ \boldsymbol{C}\ \boldsymbol{l}\ \mathsf{true}\ \boldsymbol{C}''$$

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

  $$\boldsymbol{C}''\ \xrightarrow{\beta}\ \nu\boldsymbol{n}'\,.\,(\boldsymbol{m}\ \boldsymbol{n}'\ \boldsymbol{n}\ \boldsymbol{n}'\ \boldsymbol{C}'\ .$$

  and so

  $$(\Delta\ \boldsymbol{C}\ \ (\Delta\ \nu\boldsymbol{n}'\,.\,\boldsymbol{n}\ \boldsymbol{n}'\ \boldsymbol{C}'\ \xrightarrow{\boldsymbol{n}.\nu\boldsymbol{n}'}\ (\Delta,\boldsymbol{n}'\ \boldsymbol{n}\ \boldsymbol{n}'\ \boldsymbol{C}'$$

  as required.

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

  $$\nu\boldsymbol{n}'\,.\,\boldsymbol{m}\ \boldsymbol{n}'\ \boldsymbol{l}\ \mathsf{join}\boldsymbol{n};\mathsf{true}\ \boldsymbol{n}'\ \mathsf{join}\boldsymbol{n}\boldsymbol{v}\ \boldsymbol{C}\ \boldsymbol{l}\ \mathsf{true}\ \boldsymbol{C}''$$

  then it must be the case, for some $\Delta'', \boldsymbol{C}''', \boldsymbol{C}''''$, that

  $$\boldsymbol{C}\ \nu\Delta''\,.\,(\boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{C}'''\qquad \nu\Delta''\,.\,(\boldsymbol{n}'\ \mathsf{join}\boldsymbol{n}\boldsymbol{v}\ \boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{C}'''\ \ \boldsymbol{C}''''\qquad \nu\boldsymbol{n}'\,.\,(\boldsymbol{m}\ \boldsymbol{n}'\ \boldsymbol{C}''''\ \equiv \boldsymbol{C}''$$

  So, since $\Delta\ \boldsymbol{v}$,

  $$\begin{array}{lll}(\Delta\ \boldsymbol{C} & & (\Delta\ \nu\Delta''\,.\,(\boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{C}''' \\ & \xrightarrow{\boldsymbol{n}.@\boldsymbol{v}.\boldsymbol{n}'} & (\Delta,\boldsymbol{n}'\ \nu\Delta''\,.\,(\boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{n}'\ \mathsf{let}\boldsymbol{x}=\boldsymbol{v}\mathsf{in}\boldsymbol{t}\ \boldsymbol{C}''' \\ & \xleftarrow{\beta} & (\Delta,\boldsymbol{n}'\ \nu\Delta''\,.\,(\boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{n}'\ \mathsf{join}\boldsymbol{n}\boldsymbol{v}\ \boldsymbol{C}''' \\ & & (\Delta,\boldsymbol{n}'\ \boldsymbol{C}''''\end{array}$$

  By confluence (Proposition 3.3), we can find $\boldsymbol{C}'$ such that

  $$(\nu\Delta''\,.\,(\boldsymbol{n}\ \lambda\boldsymbol{x}\,.\,\boldsymbol{t}\ \boldsymbol{n}'\ \mathsf{let}\boldsymbol{x}=\boldsymbol{v}\mathsf{in}\boldsymbol{t}\ \boldsymbol{C}'''\ \ \boldsymbol{C}'\ \xleftarrow{\beta}\ \boldsymbol{C}''''.$$

  Therefore

  $$(\Delta\ \boldsymbol{C}\ \xrightarrow{\boldsymbol{n}.@\boldsymbol{v}.\boldsymbol{n}'}\ (\Delta,\boldsymbol{n}'\ \boldsymbol{C}'\qquad \boldsymbol{C}''\ \xrightarrow{\beta}\ \nu\boldsymbol{n}'\,.\,(\boldsymbol{m}\ \boldsymbol{n}'\ \boldsymbol{C}'$$

  as required. □

A ***bisimulation***

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

Proof: Straightforward. □

Lemma 3.8 (Extrusion) *If*

$$\Delta, m \vDash \nu\Delta'.(m\ \Delta'\ \square C_1 \approx^{pb} \nu\Delta'.(m\ \Delta'\ \square C_2$$

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

Proof: We prove this by coinduction. Let $\mathcal{R}$ be defined such that $\Delta, \Delta' \vDash C_1\ \mathcal{R}\ C_2$ if and only if

$$\Delta, m \vDash \nu\Delta'.(m\ \Delta'\ \square C_1 \approx^{pb} \nu\Delta'.(m\ \Delta'\ \square C_2$$

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

- Reduction closure for $\mathcal{R}$ is immediate from the definition.
- To show that $\mathcal{R}$ is barbed closed we suppose $C_1 \square_n$ for some n. If n is defined in Δ then by assumption we find $C_2 \square_n$. 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

$$n' \text{ let } x = \pi_n(\text{join } m \text{ in join } x$$

where π_n refers to projected component of Δ' at which n occurs. This configuration fetches the private names exported at m and uses the particular.15(the≈pb fmiT)Tj1411.950ality of

Note that

$$\boldsymbol{m}\,\Delta\;\boldsymbol{C} \xrightarrow{\beta} \boldsymbol{m}\,\Delta\;\boldsymbol{C}.$$

S6.4028 (k)(4)-23 0.91638 Tf 24.93398 0 Td (f) Tj/R8 591.9552Tf77[(34084/R3611.9552T87.057420Td(97

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 [21, 22].

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 [26] 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:

$$\boldsymbol{C}_{\omega} \approx \boldsymbol{C}_{0}$$

where $\boldsymbol{C}_{\omega}$ is understood to be the interpretation of $\boldsymbol{C}$ using the original semantics and $\boldsymbol{C}_{0}$ 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 $\boldsymbol{C}$. This leads us to defining a hierarchy of semantics, indexed by type order. In $\boldsymbol{C}_{\boldsymbol{n}}$, terms of type lower than $\boldsymbol{n}$ are passed directly, and terms of higher type are trigger-encoded. We can then regard $\boldsymbol{C}_{\omega}$ as the 'limit' of the semantics $\boldsymbol{C}_{0}, \boldsymbol{C}_{1}, \ldots$ 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 $\boldsymbol{C}_{0} \approx \boldsymbol{C}^{\square}_{\boldsymbol{i}}$ for all $\boldsymbol{i}$ then $\boldsymbol{C}_{0} \approx \boldsymbol{C}^{\square}_{\omega}$.
3. Prove for all $\boldsymbol{i}$ that $\boldsymbol{C}_{\boldsymbol{i}} \approx \boldsymbol{C}_{\boldsymbol{i}-1}$.

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 $\boldsymbol{\mu}\nu$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 is 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:

$$(\Delta \mid n\ \mathsf{let}\,x = !\mathsf{recv}\,n'\,\mathsf{in}\,t \quad \xrightarrow{\tau\ \ \nu(n'',v)} \quad (\Delta \mid \nu n''\,.\,n\ \mathsf{let}\,x = v\,\mathsf{in}\,t\ \mid n''\ \mathsf{let}\,x = !\mathsf{recv}\,n\,\mathsf{in}\,t$$

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 $a \mid f$, 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. Recall that the order of a type $O(\sigma$ is defined by induction such that $O(\sigma_1 < O(\sigma_1 \to \sigma_2 \geq O(\sigma_2$ and $O(\sigma\,\mathsf{thread} = O(\sigma\,\mathsf{chan} = 0$ 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

here. Let the level i substitution v/x_i be defined by:

$$\begin{array}{lcl} C\ b/x_i & = & C\ b/x \\ C\ n/x_i & = & C\ n/x \\ C\ (v_1, v_2\ /x_i & = & (C\ (x_1, x_2\ /x\quad v_1/x_{1\ i}\ v_2/x_{2\ i} \\ C\ f/x_i & = & \begin{cases} C\ f/x \text{ if } O(f \leq i \\ \nu a, n. \quad / \quad . \quad / \end{cases} \end{array}$$

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) $\Delta \vDash \nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathbf{t}] \approx \nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathsf{let}\,\mathbf{x}=\mathbf{t}\,\mathsf{in}\,\mathbf{x}]$

(ii) $\Delta \vDash \nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathsf{let}\,\mathbf{x}_1=\mathbf{t}_1\,\mathsf{in}\,\mathbf{t}_2] \approx \nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathsf{let}\,\mathbf{x}_2=\mathbf{t}_1\,\mathsf{in}\,\mathsf{let}\,\mathbf{x}_1=\mathbf{x}_2\,\mathsf{in}\,\mathbf{t}_2]$

Proof: This is easy to prove using bisimulations. The only point to watch is the case in which $\mathbf{t}$ (or $\mathbf{t}_1$) is itself a let expression. To accommodate this we must build the witness, for (i) say, as

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

where $\mathsf{let}\,\mathbf{x}=\mathbf{t}\,\mathsf{in}\,\mathbf{t}$ refers to the nested sequence $\mathsf{let}\,\mathbf{x}_1=\mathbf{t}_1\,\mathsf{in}\,\mathsf{let}\,\mathbf{x}_2=\mathbf{t}_2\,\mathsf{in}\,\ldots\,\mathsf{let}\,\mathbf{x}_n=\mathbf{t}_n\,\mathsf{in}\,\mathbf{t}$. □

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

Lemma 4.4

$$\Delta \vDash \nu\Delta'.\mathbf{C}[\nu\mathbf{n}'\mathbf{r}.(\mathbf{n}\ \mathsf{let}\,\mathbf{x}=\mathsf{recv}\,\mathbf{r}\,\mathsf{in}\,\mathbf{t}_2 \mid \mathbf{n}'\ \mathsf{let}\,\mathbf{z}=\mathbf{t}_1\,\mathsf{in}\,\mathsf{send}\,(\mathbf{r},\mathbf{z}] \approx \nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathsf{let}\,\mathbf{x}=\mathbf{t}_1\,\mathsf{in}\,\mathbf{t}_2]$$

Proof: We use the bisimulation up to $(\xrightarrow{\beta}[\,], \approx$ technique here. 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

$$\Delta[\nu\Delta'.\mathbf{C}[\nu\mathbf{n}'\mathbf{r}.(\mathbf{n}\ \mathsf{let}\,\mathbf{x}=\mathsf{recv}\,\mathbf{r}\,\mathsf{in}\,\mathbf{t}_2 \mid \mathbf{n}'\ \mathsf{let}\,\mathbf{x}=\mathbf{t}\,\mathsf{in}\,\mathsf{let}\,\mathbf{z}=\mathbf{t}_1\,\mathsf{in}\,\mathsf{send}\,(\mathbf{r},\mathbf{z}]$$

and

$$\Delta[\nu\Delta'.\mathbf{C}[\mathbf{n}\ \mathsf{let}\,\mathbf{x}=\mathbf{t}\,\mathsf{in}\,\mathsf{let}\,\mathbf{x}=\mathbf{t}_1\,\mathsf{in}\,\mathbf{t}_2]$$

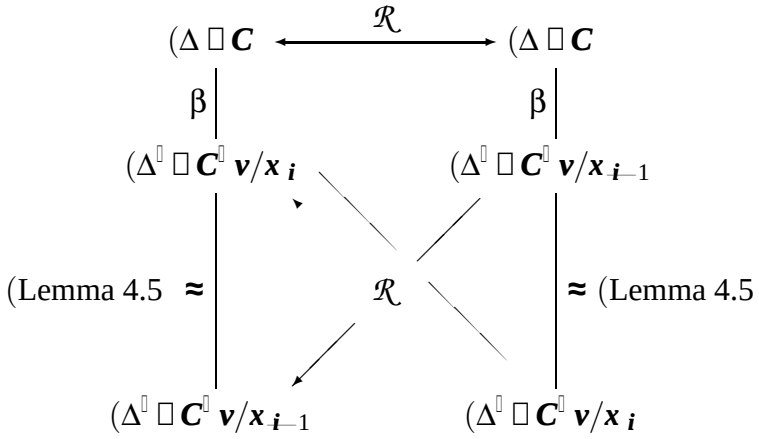
and show that $\mathcal{R}$ has the relevant closure properties. This is more or less straightforward checking. The only points of interest occur in the situation when the evaluation stack is empty and $\mathbf{t}_1$ is a value. In this case the communication on $\mathbf{r}$ occurs. We note that, because $\mathbf{r}$ 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 $\mathbf{n}'$ and a new name declaration for $\mathbf{r}$ 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 $C\ v/x_{i\ i} \approx C$

is reached where $\mathcal{B}(\tau_{\boldsymbol{a}}$ refers to the body of the abstraction defining the trigger call. Noting that $\boldsymbol{O}(\boldsymbol{v} \ < \boldsymbol{i}$

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:

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

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

$$(\Delta \vdash C_2 \stackrel{\hat{\alpha}}{=\!=} (\Delta' \vdash C_2^{'j}$$

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, 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 substitution performed in the transitions $(\Delta \vdash C_2 \stackrel{\hat{\alpha}}{=\!=} (\Delta' \vdash C_2^{'j}$ are not triggered. Therefore we also have $(\Delta \vdash C_2 \stackrel{\hat{\alpha}}{=} (\Delta' \vdash C_2^{'j}$ 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 jR1611.9552Tf+t

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

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 technical lemmas to assist in its exposition. 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*** $\Delta \models C_1 \{n\ v_c\} \approx C_2 \{n\ v_c\}$ **.**

(iii) If $\Delta, n : \sigma \models C_1 \{n\ v_1\} \approx C_2 \{n\ v_2\}$ ***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*** σ thread ***and for fresh l of type*** unit thread ***we have:***

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

$$\Delta, n \models \nu\Delta_1, l . (C_1 \{n\ f_1\} \{l\ a\}) \approx \nu\Delta_2, l . (C_2 \{n\ f_2\} \{l\ a\})$$

for any abstractions f_1, f_2**.**

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

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

for $\Delta \{r : \sigma\,\text{chan}\}$**.**

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

$$\Delta \models \nu\Delta_1, l . (C_1 \{l\ \text{send}$$

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

  To begin with we let

  $$\Delta \vDash \nu\Delta_1, \boldsymbol{l} \,.\, (\boldsymbol{C}_1 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_1 \quad \mathcal{R}\ \nu\Delta_2, \boldsymbol{l} \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_2$$

  hold exactly when $\Delta, \boldsymbol{n} \vDash \nu\Delta_1 \,.\, (\boldsymbol{C}_1 \mid \boldsymbol{n}\ \boldsymbol{v}_1 \quad \approx \nu\Delta_2 \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{n}\ \boldsymbol{v}_2$. We will show that $\mathcal{R} \mid \approx$ forms a bisimulation relation:

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

  $$\Delta \vDash \nu\Delta_1, \boldsymbol{l} \,.\, (\boldsymbol{C}_1 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_1 \quad \mathcal{R}\ \nu\Delta_2, \boldsymbol{l} \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_2 \quad ,$$

  witnessed by

  $$\Delta, \boldsymbol{n} \vDash \nu\Delta_1 \,.\, (\boldsymbol{C}_1 \mid \boldsymbol{n}\ \boldsymbol{v}_1 \quad \approx \nu\Delta_2 \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{n}\ \boldsymbol{v}_2 \quad .$$

  Furthermore suppose that $\Delta \mid \nu\Delta_1, \boldsymbol{l} \,.\, \boldsymbol{C}_1 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_1 \xrightarrow{\alpha} \Delta' \mid \boldsymbol{D}_1$ with $\boldsymbol{n} \# \alpha$. We consider the possible forms for this transition:

  - Firstly, α may have originated in $\boldsymbol{C}_1$, that is, Δ' is Δ, Δ_0 where the domain of Δ_0 is the bound names of α, and if we write $\Delta_{\boldsymbol{i}}$ as $\Delta'_{\boldsymbol{i}}, \Delta_0$ (for $\boldsymbol{i} = 1, 2$), then we have $\boldsymbol{D}_1$ is, up to structural equivalence, of the form

    $$\nu\Delta'_1, \boldsymbol{l} \,.\, (\boldsymbol{C}'_1 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_1 \quad .$$

    In this case we also know that

    $$\Delta, \boldsymbol{n} \mid \nu\Delta_1 \,.\, (\boldsymbol{C}_1 \mid \boldsymbol{n}\ \boldsymbol{v}_1 \xrightarrow{\alpha} \Delta', \boldsymbol{n} \mid \nu\Delta'_1 \,.\, (\boldsymbol{C}'_1 \mid \boldsymbol{n}\ \boldsymbol{v}_1$$

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

    $$\Delta, \boldsymbol{n} \mid \nu\Delta_2 \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{n}\ \boldsymbol{v}_2 \xRightarrow{\hat{\alpha}} \Delta', \boldsymbol{n} \mid \nu\Delta'_2 \,.\, (\boldsymbol{C}'_2 \mid \boldsymbol{n}\ \boldsymbol{v}_2$$

    with

    $$\Delta', \boldsymbol{n} \vDash \nu\Delta'_1 \,.\, (\boldsymbol{C}'_1 \mid \boldsymbol{n}\ \boldsymbol{v}_1 \quad \approx \quad \nu\Delta'_2 \,.\, (\boldsymbol{C}'_2 \mid \boldsymbol{n}\ \boldsymbol{v}_2 \tag{1}$$

    Now we know that this transition cannot depend on $\boldsymbol{n}$ as $\boldsymbol{n}$ is not contained in α or $\boldsymbol{C}_2$. Therefore

    $$\Delta \mid \nu\Delta_2, \boldsymbol{l} \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_2 \quad \xRightarrow{\hat{\alpha}} \Delta' \mid \nu\Delta'_2, \boldsymbol{l} \,.\, (\boldsymbol{C}'_2 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_2$$

    also holds. Let us call the target of these transitions $\boldsymbol{D}_2$. We use equation (1) and the definition of $\mathcal{R}$ to observe that $\Delta' \vDash \boldsymbol{D}_1\ \mathcal{R}\ \boldsymbol{D}_2$.

  - Secondly, α may be a $\mathsf{join}(\boldsymbol{v}_{\boldsymbol{c}}$ $.\boldsymbol{n}$ transition. We know by Lemma 5.2 that we can simply use a $\mathsf{join}(\boldsymbol{v}_{\boldsymbol{c}}$ $.\boldsymbol{n}$ from $\nu\Delta_2, \boldsymbol{l} \,.\, (\boldsymbol{C}_2 \mid \boldsymbol{l}\ \mathsf{send}\,(\boldsymbol{r}, \boldsymbol{v}_2$ to match this.

– Thirdly, α is $\mathsf{send}(\boldsymbol{r}\ .\boldsymbol{n}$ and $\Delta \ \boldsymbol{D}_1$ is (up to β-reduction) $\Delta, \boldsymbol{n}\ \nu\Delta_1, \boldsymbol{l}\ .(\boldsymbol{C}_1\ $

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

$$\Delta \vdash \nu n_2, l.(C_2 \mid l\ \mathsf{send}(r, v_2 \quad \Rightarrow \Delta \vdash D_2 \qquad \Delta \models D_2 \equiv \nu n_2, l, l', a.(C_2' \mid l\ (\ \mid l'\ a \Rightarrow v_2 \quad .$$

We now simply apply part (i) and Lemma 5.2 to (2) to obtain

$$\Delta \models D_1 \approx \nu n_2, l, l', a.(C_2' \mid l\ (\ \mid l'\ a \Rightarrow v_2 \quad \approx D_2$$

as required.

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

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

  Suppose (wlog) we have

$$\Delta \models \nu\Delta_1, l.(C_1 \mid l\ \mathsf{let}\, x = t_1\, \mathsf{in}\, \mathsf{send}(r, x \quad \mathcal{R}\ \nu\Delta_2, l.(C_2 \mid l\ \mathsf{let}\, x = t_2\, \mathsf{in}\, \mathsf{send}(r, x$$

  witnessed by

$$\Delta, n \models \nu\Delta_1.(C_1 \mid n\ t_1 \quad \approx \nu\Delta_2.(C_2 \mid n\ t_2$$

  and suppose that

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

  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 $\nu\Delta_1, l.(C_1 \mid l\ \mathsf{send}(r, v_1 \quad$. It is relatively easy to see using the value transitions at n that the witness guarantees that there exists some transitions

$$\Delta, n \vdash \nu\Delta_2.(C_2 \mid n\ t_2 \quad \Rightarrow \Delta, n \vdash \nu\Delta_2, \Delta_2'.(C_2' \mid n\ v_2$$

with $\Delta, \boldsymbol{n} \vDash \nu\Delta_1 . (\boldsymbol{C}_1 \; \boldsymbol{n}\; \boldsymbol{v}_1 \approx \nu\Delta_2, \Delta_2' . (\boldsymbol{C}_2' \; \boldsymbol{n}\; \boldsymbol{v}_2$. We can apply Lemma 5.2 to this, after weakening to obtain

$$\Delta, \boldsymbol{n}, \boldsymbol{n}', \boldsymbol{a}' \vDash \nu\Delta_1 . (\boldsymbol{C}_1 \; \boldsymbol{n}\; \boldsymbol{v}_1 \; \boldsymbol{n}' \; \tau_{\boldsymbol{a}'} \approx \nu\Delta_2, \Delta_2' . (\boldsymbol{C}_2' \; \boldsymbol{n}\; \boldsymbol{v}_2 \; \boldsymbol{n}' \; \tau_{\boldsymbol{a}'}$$

and we can then apply part (iii) to obtain

$$\Delta, \boldsymbol{n}, \boldsymbol{a}' \vDash \nu\Delta_1, \boldsymbol{l} . (\boldsymbol{C}_1 \; \boldsymbol{n}\; \boldsymbol{v}_1 \; \boldsymbol{l}\; \mathsf{send}\,(\boldsymbol{r}, \tau_{\boldsymbol{a}'} \approx \nu\Delta_2, \Delta_2', \boldsymbol{l} . (\boldsymbol{C}_2' \; \boldsymbol{n}\; \boldsymbol{v}_2 \; \boldsymbol{l}\; \mathsf{send}\,(\boldsymbol{r}, \tau_{\boldsymbol{a}'} \quad . \quad (3)$$

We conclude by observing that

$\Delta \; \nu\Delta$

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

$$\Delta, n \triangleright \nu\Delta_2 . C_2 \triangleright n\ f_2 \xRightarrow{n.@v_c.n'} \Delta, n, n' \triangleright \nu\Delta_2, \Delta_2' . (C_2' \triangleright n\ f_2 \triangleright n'\ t_2$$

such that

$$\Delta, n, n' \vDash \nu\Delta_1 . C_1 \triangleright n\ f_1 \triangleright n'\ \mathsf{let}\, x = v_c\, \mathsf{in}\, t_1 \quad \approx \quad \nu\Delta_2, \Delta_2' . (C_2' \triangleright n\ f_2 \triangleright n'\ t_2 \quad . \qquad (4)$$

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

$$\Delta, n \triangleright D_2 \ \Rightarrow \Delta, n \triangleright \nu\Delta_2, \Delta_2', l, l' . (C_2' \triangleright n\ f_2 \triangleright l\ \mathsf{let}\, z = t_2\, \mathsf{in}\, \mathsf{send}(r, z \quad \triangleright l'\ a \triangleright f_2$$

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

$$(\Delta, n \triangleright \nu\Delta_2, l . C_2 \triangleright l\ a \triangleright f_2 \xRightarrow{\alpha} (\Delta, n \triangleright D_2 \quad \Rightarrow (\Delta, n \triangleright D_2'$$

and, by using part (ii) at n' at type σ_2 with (4), and then by definition of $\mathcal{R}$, we have $\Delta \vDash D_1 \xrightarrow{\beta} \mathcal{R} \approx D_2'$.

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

$$\nu\Delta_1' . (C_1' \triangleright n_0\ \mathsf{let}\,(\ = \mathsf{send}(a, (v_1, r \quad \mathsf{in}\, t_1' \quad .$$

We know that, by hypothesis,

$$\Delta, n \vDash \nu\Delta_1 . C_1 \triangleright n\ f_1 \ \approx \nu\Delta_2 . C_2 \triangleright n\ f_2 \ ,$$

and we also know that

$$(\Delta, n \triangleright \nu\Delta_1 . (C_1 \triangleright n\ f_1 \quad \xrightarrow{\mathsf{snd}(a.n')} \xrightarrow{\beta} \triangleright (\Delta, n, n' \triangleright \nu\Delta_1, \Delta_1' . (C_1' \triangleright n_0\ t_1' \triangleright n'\ (v_1, r \quad \triangleright n\ f_1 \quad .$$

Given this we can find matching transitions

$$(\Delta, n \triangleright \nu\Delta_2 . (C_2 \triangleright n\ f_2 \quad \xRightarrow{\mathsf{snd}(a.n')} \Rightarrow (\Delta, n, n' \triangleright \nu\Delta_2, \Delta_2' . (C_2' \triangleright n'\ w \triangleright n\ f_2$$

with

$$\Delta, n, n' \vDash \nu\Delta_1, \Delta_1' . (C_1' \triangleright n_0\ t_1' \triangleright n'\ (r, v_1 \quad \triangleright n\ f_1 \quad \approx \nu\Delta_2, \Delta_2' . (C_2' \triangleright n'\ w \triangleright n\ f_2 \quad .$$

Now, by using $n'.\mathsf{fst}.$ and $n'.\mathsf{snd}.$ projection transitions and Lemma 5.2, and because $n' \triangleright C_2'$, 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

$$\Delta, n, n'' \vDash \nu\Delta_1, \Delta_1' . (C_1' \triangleright n_0\ t_1' \triangleright n''\ v_1 \triangleright n\ f_1 \quad \approx \quad \nu\Delta_2, \Delta_2' . (C_2'' \triangleright n''\ v_2 \triangleright n\ f_2 \qquad (5)$$

with $C_2' \triangleright n\ f_2 \ \Rightarrow C_2'' \triangleright n\ f_2$.

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

Case(a): If $\boldsymbol{v}_1$ is of base type then $\boldsymbol{D}_1$, up to β-reduction, will be of the form (where $\boldsymbol{f}_1$ is $\lambda \boldsymbol{x}.\boldsymbol{t}_1$)

$$\nu\Delta_1, \Delta_1', \boldsymbol{l}, \boldsymbol{l}' . (\boldsymbol{C}_1' \,|\, \boldsymbol{n}_0\ \boldsymbol{t}_1' \,|\, \boldsymbol{n}\ \boldsymbol{f}_1 \,|\, \boldsymbol{l}\ \mathsf{let}\, \boldsymbol{x}' = (\mathsf{let}\, \boldsymbol{x} = \boldsymbol{v}_1\, \mathsf{in}\, \boldsymbol{t}_1\ \ \mathsf{in}\, \mathsf{send}\, (\boldsymbol{r}, \boldsymbol{x}' \,|\, \boldsymbol{l}'\ \boldsymbol{a} \,|\, \boldsymbol{f}_1$$

In this case, we know that $\boldsymbol{v}_1$ is canonical and, by the previous equivalence, we know that $\boldsymbol{v}_2$ is also canonical, and moreover is identical to $\boldsymbol{v}_1$. We can use Lemma 5.2 to see that

$$\Delta, \boldsymbol{n} \vDash \nu\Delta_1, \Delta_1' . (\boldsymbol{C}_1' \,|\, \boldsymbol{n}_0\ \boldsymbol{t}_1' \,|\, \boldsymbol{n}\ \boldsymbol{f}_1 \quad \approx \nu\Delta_2, \Delta_2' . (\boldsymbol{C}_2' \,|\, \boldsymbol{n}\ \boldsymbol{f}_2$$

(note that this ignores the case in which $\boldsymbol{v}_1, \boldsymbol{v}_2$ contain private names, but we may assume, because of the $\boldsymbol{n} . \nu \boldsymbol{m}$ transitions, that such names have already been extruded). We have stated that $\boldsymbol{v}_1$ is canonical so there exists a transition

$$\Delta, \boldsymbol{n} \,|\, \nu\Delta_1, \Delta_1' . (\boldsymbol{C}_1' \,|\, \boldsymbol{n}_0\ \boldsymbol{t}_1' \,|\, \boldsymbol{n}\ \boldsymbol{f}_1$$
$$\underline{\qquad \boldsymbol{n}.@\boldsymbol{v}_1 \qquad}$$

where n' is fresh and f_1 is $\lambda x.t_1$. This means that, by (5), there must exist matching transitions

$$\Delta, n, n'', a' \triangleright \nu\Delta_2, \Delta_2' . (C_2'' \triangleright n'' \; v_2 \; \triangleright n \; f_2 \xrightarrow{n.@\tau_{a'}.n'} \triangleright$$

$$\Delta, n, n', n'', a' \triangleright \nu\Delta_2, \Delta_2', \Delta_2'' . (C_2''' \triangleright n'' \; v_2 \; \triangleright n \; f_2 \; \triangleright n' \; t_2$$

such that

$$\Delta, n, n', n'', a' \models \nu\Delta_1, \Delta_1' . (C_1' \triangleright n_0 \; t_1' \; \triangleright n'' \; v_1 \; \triangleright n \; f_1 \; \triangleright n' \; \text{let}\, x = \tau_{a'} \, \text{in} \, t_1 \qquad (7)$$

$$\approx$$

$$\nu\Delta_2, \Delta_2', \Delta_2'' . (C_2''' \triangleright n'' \; v_2 \; \triangleright n \; f_2 \; \triangleright n' \; t_2$$

and $\nu\Delta_2, \Delta_1$

Lemma 5.5 *If*

$$(\Delta, a \mathrel{\square} C \mathrel{\square} n\ \tau_a \xrightarrow{\alpha} (\Delta', a \mathrel{\square} C' \mathrel{\square} n\ \tau_a$$

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

$$\nu\Delta_0\,.\,(n\ v \mathrel{\square} \nu l\,.\,l\ a \mathrel{\square} v \mathrel{\square} C_0'$$

with $n, a \mathrel{\square} \Delta_0$*, then*

$$(\Delta \mathrel{\square} \nu a\,.\,(C \mathrel{\square} C_0 \xrightarrow{\alpha} \approx (\Delta' \mathrel{\square} \nu a\,.\,(C' \mathrel{\square} C_0 \quad .$$

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

$$\nu\Delta_1\,.\,(C_1 \mathrel{\square} n_1\ \mathsf{let}\, x = \mathsf{join}\, n\, \mathsf{in}\, t_1$$

and C' is of the form

$$\nu\Delta_1\,.\,(C_1 \mathrel{\square} n_1\ \mathsf{let}\, x = \tau_a\, \mathsf{in}\, t_1 \quad .$$

Clearly though,

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

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* $\Delta, \Delta_0 \mathrel{\square} C$ *then* $\Delta \vDash \nu\Delta_0\,.\,(C_1 \mathrel{\square} C \approx \nu\Delta_0\,.\,(C_2 \mathrel{\square} C$ **.**

Proof: Define:

$$\Delta \vDash \nu\Delta_0\,.\,(C_1 \mathrel{\square} C \quad \mathcal{R}\ \nu\Delta_0\,.\,(C_2 \mathrel{\square} C$$

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

$$\Delta, \Delta_0 \vDash C_1 \mathrel{\square} \prod_{i=0}^{k} n_i\ \tau_{a_i} \approx C_2 \mathrel{\square} \prod_{i=0}^{k} n_i\ \tau_{a_i}$$

and such that a_i is a send-channel in C_1, C_2, $n_i \mathrel{\square} \Delta_0$, $a_i \mathrel{\square} \Delta_0$ (for $0 \le i \le k$), and

$$C \equiv \nu\Delta', l\,.\,(\prod_{i=1}^{k} n_i\ v_i \mathrel{\square} \prod_{i=0}^{k} l_i\ a_i \mathrel{\square} v_i \mathrel{\square} C' \quad .$$

We will demonstrate $\mathcal{R}$ to be bisimulation up to $(\xrightarrow{\beta} \mathrel{\square}, \approx$ and our result follows in the case $k = 0$. Suppose then that

$$\Delta \vDash \nu\Delta_0\,.\,(C_1 \mathrel{\square} C \quad \mathcal{R}\ \nu\Delta_0\,.\,(C_2 \mathrel{\square} C$$

and also suppose that $(\Delta \square \nu\Delta_0 \,.\, (\boldsymbol{C}_1 \square \boldsymbol{C} \xrightarrow{\alpha} (\Delta' \square \boldsymbol{D}_1$

Note that, for the sake of simplicity, we use the new name transitions to allow us to assume that any name in $\boldsymbol{v}_1$ has already been extruded. Now, Lemma 5.5 tells us that

$$(\Delta \vdash \nu\Delta_0 . (\boldsymbol{C}_2 \mid \boldsymbol{C} \quad \vdash \approx (\Delta \vdash \nu\Delta_0 . (\nu\Delta_2 . (\boldsymbol{C}_2' \mid \nu\Delta_0' . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0\ \boldsymbol{v}_2/\boldsymbol{x}$$

(call the target term $\boldsymbol{D}_2$), and by noticing that

$$\boldsymbol{D}_1 \xrightarrow{\beta} \nu\Delta_0 . (\nu\Delta_1 . (\boldsymbol{C}_1' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0 \quad \mid \nu\Delta_0' . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0\ \boldsymbol{v}_1/\boldsymbol{x}$$

along with the fact that $\boldsymbol{v}_1 \equiv \boldsymbol{v}_2$, (10) tells us

$$\Delta \vDash \boldsymbol{D}_1 \xrightarrow{\beta} \mathcal{R} \approx \boldsymbol{D}_2$$

as required.

Otherwise, $\boldsymbol{v}_1$ must be an abstraction and

$$\boldsymbol{D}_1 \xrightarrow{\beta} \nu\Delta_0, \boldsymbol{a}' . (\nu\Delta_1, \boldsymbol{l} . (\boldsymbol{C}_1' \mid \boldsymbol{n}_1\ \boldsymbol{t}_1 \mid \boldsymbol{l}\ \boldsymbol{a}' \vdash \boldsymbol{v}_1 \quad \mid \nu\Delta_0' . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0\ \tau_{\boldsymbol{a}'}/\boldsymbol{x} \quad .$$

We use Lemmas 5.3 and 5.2 to see that

$$\Delta, \Delta_0, \boldsymbol{n}' \vDash \nu\Delta_1 . (\boldsymbol{C}_1', \boldsymbol{l} \mid \boldsymbol{n}_1\ \boldsymbol{t}_1 \mid \boldsymbol{n}\ \tau_{\boldsymbol{a}} \mid \boldsymbol{l}\ \boldsymbol{a}' \vdash \boldsymbol{v}_1 \quad \approx \nu\Delta_2, \boldsymbol{l} . (\boldsymbol{C}_2' \mid \boldsymbol{n}\ \tau_{\boldsymbol{a}} \mid \boldsymbol{l}\ \boldsymbol{a}' \vdash \boldsymbol{v}_2 \qquad (11)$$

Again, Lemma 5.5 tells us that

$$(\Delta \vdash \nu\Delta_0 . (\boldsymbol{C}_2 \mid \boldsymbol{C} \quad \vdash \approx (\Delta \vdash \nu\Delta_0, \boldsymbol{a}' . (\nu\Delta_2, \boldsymbol{l} . (\boldsymbol{C}_2' \mid \boldsymbol{l}\ \boldsymbol{a}' \vdash \boldsymbol{v}_2 \quad \mid \nu\Delta_0' . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0\ \tau_{\boldsymbol{a}'}/\boldsymbol{x}$$

(call the target term $\boldsymbol{D}_2$). Thus by using (11) we see that $\Delta \vDash \boldsymbol{D}_1 \xrightarrow{\beta} \mathcal{R} \approx \boldsymbol{D}_2$ as required.

- Suppose

$$\begin{aligned} \boldsymbol{C}_1 \quad &\equiv \quad \nu\Delta_1 . (\boldsymbol{C}_1' \mid \boldsymbol{n}_1\ \mathsf{let}\, \boldsymbol{x} = \mathsf{recv}\, \boldsymbol{c}\, \mathsf{in}\, \boldsymbol{t}_1 \\ \boldsymbol{C} \quad &\equiv \quad \nu\Delta_0' . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \mathsf{let}\, (\ = \mathsf{send}\, (\boldsymbol{c}, \boldsymbol{v}\ \mathsf{in}\, \boldsymbol{t}_0 \\ \boldsymbol{D}_1 \quad &\xrightarrow{\beta} \quad \nu\Delta_0, \Delta_1, \Delta_0' . (\boldsymbol{C}_1' \mid \boldsymbol{n}_1\ \mathsf{let}\, \boldsymbol{x} = \boldsymbol{v}\, \mathsf{in}\, \boldsymbol{t}_1 \mid \boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0 \end{aligned}$$

Again, we must consider whether $\boldsymbol{v}_1$ 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 $\boldsymbol{D}_1$ such that

$$\boldsymbol{D}_1 \xrightarrow{\beta} \nu\Delta_0, \boldsymbol{a}' . (\nu\Delta_1 . (\boldsymbol{C}_1' \mid \boldsymbol{n}_1\ \boldsymbol{t}_1\ \tau_{\boldsymbol{a}'}/\boldsymbol{x} \quad \mid \nu\Delta_0', \boldsymbol{l} . (\boldsymbol{C}_0' \mid \boldsymbol{n}_0\ \boldsymbol{t}_0 \mid \boldsymbol{l}\ \boldsymbol{a}' \vdash \boldsymbol{v}$$

We know that

$$(\Delta, \Delta_0, \boldsymbol{a}' \vdash \boldsymbol{C}_1 \xrightarrow{\mathsf{recv}\ \boldsymbol{c}, \tau_{\boldsymbol{a}'}} \approx (\Delta, \Delta_0, \boldsymbol{a}' \vdash \nu\Delta_1' . (\boldsymbol{C}_1' \mid \boldsymbol{n}_1\ \boldsymbol{t}_1\ \tau_{\boldsymbol{a}'}/\boldsymbol{x}$$

so, by the hypothesis that

$$\Delta, \Delta_0 \vDash \boldsymbol{C}_1 \mid \boldsymbol{n}\ \tau_{\boldsymbol{a}} \approx \boldsymbol{C}_2 \mid \boldsymbol{n}\ \tau_{\boldsymbol{a}}\ ,$$

say, we also know that there exists some

$$(\Delta, \Delta_0, \vec{a} \cdot \boldsymbol{C}_2 \cdot \boldsymbol{n}\ \tau_{\boldsymbol{a}} \xrightarrow{\nu(\boldsymbol{c}, \tau_{\vec{a}})} (\Delta, \Delta_0, \vec{a} \cdot \nu\Delta_2^{\vec{}} . (\boldsymbol{C}_2^{\vec{}} \cdot \boldsymbol{n}\ \tau_{\boldsymbol{a}}$$

such that

$$\Delta, \Delta_0, \vec{a} \vDash \nu\Delta_1^{\vec{}} . (\boldsymbol{C}_1^{\vec{}} \cdot \boldsymbol{n}_1\ \boldsymbol{t}_1\ \tau_{\vec{a}}/\boldsymbol{x}\ \cdot \boldsymbol{n}\ \tau_{\boldsymbol{a}} \approx \nu\Delta_2^{\vec{}} . (\boldsymbol{C}_2^{\vec{}} \cdot \boldsymbol{n}\ \tau_{\boldsymbol{a}} \quad (12)$$

We use Lemma 5.5 to observe that

$$(\Delta \cdot \nu\Delta_0 . (\boldsymbol{C}_2 \cdot \boldsymbol{C}\ \cdot \approx (\Delta \cdot \nu\Delta_0, \vec{a} . (\nu\Delta_2^{\vec{}} . \boldsymbol{C}_2^{\vec{}} \cdot \nu\Delta_0^{\vec{}}, \boldsymbol{l} . (\boldsymbol{C}_0^{\vec{}} \cdot \boldsymbol{n}_0\ \boldsymbol{t}_0\ \cdot \boldsymbol{l}\ \vec{a} \cdot \boldsymbol{v}$$

call the target term $\boldsymbol{D}_2$. We use (12) to conclude that $\Delta, \Delta_0, \boldsymbol{D}_1^2 \boldsymbol{C}^{\beta} \cdot \mathcal{R}\ \boldsymbol{D}_2$.

- Suppose

$$\begin{aligned} \boldsymbol{C}_1 &\equiv \nu\Delta_1 . (\boldsymbol{C}_1^{\vec{}} \cdot \boldsymbol{n}_1\ \boldsymbol{v}_1 \\ \boldsymbol{C} &\equiv \nu\Delta_0^{\vec{}} . (\boldsymbol{C}_0^{\vec{}} \cdot \boldsymbol{n}_0\ \text{let}\ \boldsymbol{x} = \boldsymbol{y}\ \text{in} \end{aligned}$$

- Suppose

$$\begin{array}{rcl} \boldsymbol{C}_1 & \equiv & \nu\Delta_1\,.\,(\boldsymbol{C}_1' \,\square\, \boldsymbol{n}_1\ \mathsf{let}\,\boldsymbol{x} = \mathsf{join}\,\boldsymbol{n}_0\,\mathsf{in}\,\boldsymbol{t}_1 \\ \boldsymbol{C} & \equiv & \nu\Delta_0'\,.\,(\boldsymbol{C}_0' \,\square\, \boldsymbol{n}_0\ \boldsymbol{v} \\ \boldsymbol{D}_1 & \equiv & \nu\Delta_0, \Delta_1, \Delta_0'\,.\,(\boldsymbol{C}_1' \,\square\, \boldsymbol{n}_1\ \mathsf{let}\,\boldsymbol{x} = \boldsymbol{v}\,\mathsf{in}\,\boldsymbol{t}_1\ \square\, \boldsymbol{C}_0' \,\square\, \boldsymbol{n}_0\ \boldsymbol{v} \end{array}$$

  Suppose firstly that