First-Order Reasoning for Higher-Order Concurrency

Vasileios Koutavas  Matthew Hennessy
First-Order Reasoning for Higher-Order Concurrency*

Vasileios Koutavas Matthew Hennessy

{Vasileios.Koutavas, Matthew.Hennessy}@cs.tcd.ie

Trinity College Dublin
February 4, 2010

Abstract

We present a practical first-order theory of a higher-order \( \pi \)-calculus which is both sound and complete with respect to a standard semantic equivalence. The theory is a product of combining and simplifying two of the most prominent theories for HO\( \pi \) of Sangiorgi et al. and Jeffrey and Rathke [13, 6], and a novel approach to scope extrusion. In this way we obtain an elementary labelled transition system where the standard theory of first-order weak bisimulation and its corresponding propositional Hennessy-Milner logic can be applied.

The usefulness of our theory is demonstrated by straightforward proofs of equivalences between compact but intricate higher-order processes using witness first-order bisimulations, and proofs of inequivalence using the propositional Hennessy-Milner logic.

Beyond HO\( \pi \), our technique is applicable to other languages, including languages with distribution. In support of this claim we outline how our first-order theory can be easily adapted to model located higher-order processes, in which names can be locally scoped.

To our knowledge this is the first practical and fully-abstract theory for such a language.

Contents

1 Introduction 3

2 The Language 5
2.1 Syntax . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Reduction semantics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.3 A behavioural equivalence . . . . . . . . . . . . . . . . . . . . . . . . . . . 7

3 The Labelled Transition System 8

4 Bisimulations 14
4.1 Strong Bisimulations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
4.2 Weak Bisimulations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
4.3 Logical Characterisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16

5 Soundness of Weak Bisimilarity 18
5.1 Reductions versus \( \tau \)-steps . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
5.2 Reduction-closure and preservation of barbs . . . . . . . . . . . . . . . . . . 20
5.3 Parallel Contexts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21

*The financial support of SFI is gratefully acknowledged
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Completeness of Weak Bisimilarity</td>
<td>22</td>
</tr>
<tr>
<td>6.1</td>
<td>Concretion of Configurations</td>
<td>22</td>
</tr>
<tr>
<td>6.2</td>
<td>Completeness</td>
<td>24</td>
</tr>
<tr>
<td>7</td>
<td>Full Barbed Congruence</td>
<td>27</td>
</tr>
<tr>
<td>8</td>
<td>Examples</td>
<td>29</td>
</tr>
<tr>
<td>8.1</td>
<td>Implementation of Replication</td>
<td>31</td>
</tr>
<tr>
<td>8.2</td>
<td>A Trigger-Installing Ping Service</td>
<td>32</td>
</tr>
<tr>
<td>8.3</td>
<td>A Trigger-Promoting Ping Service</td>
<td>34</td>
</tr>
<tr>
<td>8.4</td>
<td>Composition of Triggers with Replication</td>
<td>35</td>
</tr>
<tr>
<td>8.5</td>
<td>The Processes in Figure 1</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>First-Order Processes</td>
<td>39</td>
</tr>
<tr>
<td>10</td>
<td>Extension to Distributed Localities</td>
<td>40</td>
</tr>
<tr>
<td>10.1</td>
<td>Adaptation of the LTS</td>
<td>41</td>
</tr>
<tr>
<td>10.2</td>
<td>Full-abstraction</td>
<td>43</td>
</tr>
<tr>
<td>10.2.1</td>
<td>Soundness</td>
<td>44</td>
</tr>
<tr>
<td>10.3</td>
<td>Completeness</td>
<td>45</td>
</tr>
<tr>
<td>10.4</td>
<td>A Delay Ping Service</td>
<td>47</td>
</tr>
<tr>
<td>11</td>
<td>Conclusions</td>
<td>47</td>
</tr>
<tr>
<td>A</td>
<td>Proof of Theorem 5.7 ($(≈)^{ext} \subseteq (≈)$)</td>
<td>50</td>
</tr>
</tbody>
</table>
Developing effective reasoning techniques for programming languages with higher-order constructs is a challenging problem, made even more challenging by the presence of concurrency, mobility, and distribution. The difficulties involved are exemplified by the search for reasonable proof techniques for establishing behavioural equivalences between processes written in higher-order versions of the $\pi$-calculus [10, 11, 12, 14, 6, 13, 15] in which, besides first-order values, processes can be communicated.

To illustrate the challenges of reasoning in higher-order concurrent languages let us consider the pairs of higher-order processes shown in Figure 1, where $\oplus$ is an internal choice operator and $\text{app}$ causes the execution of a suspended process. The differences within each pair of processes is highlighted. All processes initially receive two suspended processes $X$ and $Y$ on channel $c$ and dynamically create a local channel $t$. Then each one combines $X$ and $Y$ in a slightly different way into two possible replies on channel $c$, chosen non-deterministically. After a reply is sent, all that is left of the processes is the same replication (denoted by the $\ast$-operator) guarded by the private channel $c$.

The aim of the current paper is to design an elementary bisimulation theory for higher-order concurrency which, besides being fully-abstract with respect to a semantic equivalence, supports a reasonable proof methodology and scales to languages with distribution. Our methodology combines and simplifies ideas carefully selected from the literature [6, 13], together with a novel treatment of names and extrusion. This results to a purely first-order theory of bisimulation in which actions take a particularly simple form, and in which the examples in Figure 1 can be handled in a straightforward manner. Furthermore, our theory scales to other higher-order languages, including the ones with distribution.

A number of different reasoning techniques have been developed for HO$\pi$ including the translation of higher-order communication to triggers in the first-order $\pi$-calculus [10, 11] and its improvement in [6], and environmental bisimulations [13]. All have been shown to coincide with contextual bisimulation [12] (often referred to as being fully-abstract) but it is unclear how these techniques scale to languages with distribution. In such languages the behaviour of a process depends on its dynamic environment, hence a fully abstract translation of higher-order communication to triggers is not possible [19].

Sangiorgi et al. [13], motivated by work in functional languages [18, 17, 8], use a standard LTS and annotate bisimulations with an environment (relation) containing the knowledge currently known to the interrogator; this allows the interrogating actions to be meaningfully based on the interrogator’s current knowledge. Their method simplifies the metatheory (e.g. showing that bisimilarity is a congruence), but leads to a definition for bisimulations with many and arguably complex conditions. As an example, for higher-order inputs of related
processes one has to consider all possible input values constructed by identical contexts with related values in their holes. This strong proof obligation is sometimes mitigated by the use of up-to context techniques.

Jeffrey and Rathke [6] use a more restrictive approach of formal triggers. A higher-order output of a process is transformed to a special trigger service holding the actual value and only a pointer for invoking the service is passed to the interrogator. Similarly, a higher-order input is fed with a trigger with which the process can intuitively run the actual value—but actually only an observable action is recorded in the LTS.

We believe that both methods have useful intuitions and that their combination has greater value than the sum of its parts. Our theory incorporates and simplifies their insights.

We use knowledge environments in the LTS, rather than on bisimulations, that record the values exposed to the context, and test related processes with symbolic higher-order inputs. We also take this one step further by including an explicit representation of the information known only to the process. Thus configurations take the form $v \bar{\tau} (\Delta, P)$ which consists of the (higher-order) process $P$ under interrogation, a representation $\Delta$ of the knowledge currently known to the interrogator about this process, and the information $\bar{\tau}$ known to the process but currently unknown to the interrogator.

This extension allows us to simplify considerably the actions on which our bisimulations are based; in particular it eliminates the need for explicitly extruding new information and communication actions are labelled simply $c!v$ and $c?v$ thereby relieving us of the need to manage the complications inherent in the use of extrusion. A significant consequence is that bisimulation in our theory is characterised by a propositional Hennessy-Milner Logic (HML) [4], which would not be possible with other LTS’s.

The main contributions of the paper can be summarised as follows:

(i) We define a first-order, fully-abstract, theory of standard weak bisimulation equivalence for a higher-order $\pi$-calculus, called pp-$\pi$, that unifies two distinct techniques. The theory is compositional in the sense that the equivalence is preserved by arbitrary process contexts.

(ii) The associated coinductive reasoning technique for pp-$\pi$ processes is effective: because the theory is first-order it is straightforward to demonstrate equivalences between processes by exhibiting witness bisimulations. In support of this we provide a series of compelling example process equivalences.

(iii) We give the first propositional HML characterisation of weak bisimulation for a higher-order $\pi$-calculus; this result easily transfers to the first-order $\pi$-calculus. We use this to give simple proofs of inequivalence between higher-order processes, which is difficult to achieve with existing theories.

(iv) We prove that contextual equivalence in a higher-order setting is a conservative extension of the first-order $\pi$-calculus, thus confirming that results and reasoning methods from first-order $\pi$-calculus transfer to a higher-order setting.

(v) We illustrate the robustness of our technique by extending it to a language with distributed localities that dynamically confine the communication over a set of channels. For such a language there is no fully-abstract or practical translation to first-order $\pi$-calculus [19], and it is unclear how environmental bisimulations could provide a useful theory. To our knowledge this is the first practical, fully-abstract theory for such a language.

A direct consequence of our theory is that it brings the analysis of higher-order concurrent and distributed systems within the scope of existing first-order proof technologies.
The remainder of the paper is organised as follows: the next section defines the language pp-π, giving the syntax, a reduction semantics and a simple type system for ensuring that communicated values are appropriately typed. Section 3 details our first-order LTS for pp-π, and Section 4 defines strong and weak bisimulations and a characterisation of the latter in terms of a propositional Hennessy-Milner Logic. Sections 5 and 6 contain the proofs of soundness and completeness of our theory with respect to contextual equivalence that preserves only parallel contexts, and Section 7 proves that our theory is fully abstract with respect to the full contextual equivalence. Section 8 is devoted to proving several interesting equivalences by using weak bisimulations, and an inequivalence by providing a discriminating HML formula. Section 9 proves the conservativity theorem. Section 10 extends the language with localities that restrict communication over certain channels, adapts our theory to this language, and proves an equivalence in this extended language. The paper closes in Section 11 with conclusions and a discussion of related work.

2 The Language

2.1 Syntax

We study the language pp-π (process-passing-π), a higher-order version of the π-calculus, which allows processes to be communicated and is roughly equivalent to the language studied in [12].

We assume a set of channel names Name, ranged over by a, b, . . . and a separate set of variables Variable, ranged over by x, y, . . ., and use u, v, . . . to denote identifiers, from (Variable ∪ Name).

The syntax of the language is given in Figure 2. The basic constructs in pp-π are the input and output of typed values along channels, u?⟨V : t⟩. P and u!(V : t). P. In the former a
value of type $t$ is received on channel $u$ and bound to the variable $x$ in $P$, while in the latter the value $V$ of type $t$ is output on channel $u$ and the process continues with the execution of the code $P$. In addition we have the standard constructs of the $\pi$-calculus: replication $* (P)$, parallel execution $(P | Q)$, the generation of new names $\nu P$, and the testing of these names $\text{if } u = v \text{ then } P \text{ else } Q$.

In the $\pi$-calculus the only values which can be transmitted along channels are names, but in pp-$\pi$ thunked or suspended processes, of the form $\lambda P$, are also allowed; when such a value is received by a process it can be executed, via the new construct $\text{app } V$.

**Typing:** We have a very-lightweight notion of type whose purpose is simply to ensure, dynamically, that at any point in time when a value is received at a certain type it is subsequently only used at that type. Values can be one of two types, $\text{Nm}$ for names and $\text{Pr}$ for (suspended) processes. Type inference is with respect to type environments, consisting of finite sets of variable-type associations $x : t$. Then the typing judgements take the form

- $\Gamma \vdash V : t$, indicating that relative to $\Gamma$ the value $V$ has types $t$
- $\Gamma \vdash P : \text{OK}$, indicating that the process term $P$ is well-typed relative to $\Gamma$.

The rules for inferring the judgements are also given in Figure 2.

**Example:** Consider the following process, which describes a service at $s$:

$$ * (s^{?}(x:\text{Nm}). s^{?}(y:\text{Pr}). v f. v r. x! f). x! r. ) $$

$$ * (f^{?}(z:\text{Nm}). z! (y:\text{Pr}). \emptyset) | $$

$$ * (r^{?}. \text{app } y) $$

It first receives as input a reply channel name, bound to $x$, and then a (suspended) process bound to $y$. It generates two new names $f$ and $r$ which it returns on the reply channel, and then sets up two new servers at those names. The first, at $f$, receives a name and forwards the suspended process there; the second, at $r$, runs the suspended process on request.

Notice that we do not assume any static typing for channel names. At different points in time they may be used to communicate values of different type. So, for example, a client using this service at $s$ will be expected to follow an implicit protocol, whereby first a name is sent on $s$ and then a process.

### 2.2 Reduction semantics

The reduction semantics is expressed as a relation

$$ P \rightarrow Q $$

Figure 3: Reduction semantics for pp-$\pi$
where \( P \) and \( Q \) are assumed to be well-typed processes, that is process terms satisfying \( \emptyset \vdash P : \text{OK} \) and \( \emptyset \vdash Q : \text{OK} \). The rules for inferring these judgements are given in Figure 3 and are relatively standard. The main rule is for communication,

\[
a!(V:t).P \mid a?(x:t).Q \rightarrow P \mid Q[V/x]
\]

Note that this communication along \( a \) can only happen if the partners agree on the type of the value being transmitted. The other significant rule is for the initiation of a suspended process,

\[
\text{app } \lambda P \rightarrow P
\]

The remaining rules are standard, borrowed from the \( \pi \)-calculus; in particular reductions are relative to a structural equivalence \( P \equiv Q \) which we now define.

**Definition 2.1** (Structural equivalences). Limited structural equivalence (\( \equiv \)) is defined to be the least equivalence relation on processes satisfying the axioms

\[
P = \emptyset \mid P \quad P \mid Q = Q \mid P \quad (P_1 \mid P_2) \mid P_3 = P_1 \mid (P_2 \mid P_3)
\]

and closed under the two operators \( - \mid - \) and \( \nu a. - \).

Structural equivalence (\( \equiv \)) is obtained by adding the further axioms

\[
\begin{align*}
\nu a. \nu b. P & = \nu b. \nu a. P \quad \ast\!(P) = P \mid \ast\!(P) \\
\nu a. \emptyset & = \emptyset \quad \nu a. (P \mid Q) = (\nu a. P) \mid Q \quad (a \not\in \text{fn}(Q))
\end{align*}
\]

As we have already stated, structural equivalence (\( \equiv \)) is used in the reduction semantics, but the more restrictive limited equivalence (\( \equiv \)) will be useful in proofs of equivalence.

**Lemma 2.2** (Substitution). If \( \Gamma, x : t \vdash P : \text{OK} \) and \( \emptyset \vdash \nu V : t \) then \( \Gamma \vdash P[V/x] : \text{OK} \).

**Proof.** By rule induction. \( \square \)

**Lemma 2.3.** If \( \Gamma, x : t \vdash P : \text{OK} \) and \( x \not\in \text{fn}(P) \) then \( \Gamma \vdash \nu V : t \).

**Proof.** By rule induction. \( \square \)

**Lemma 2.4.** If \( P \equiv Q \) and \( \Gamma \vdash P : \text{OK} \) then \( \Gamma \vdash Q : \text{OK} \).

**Proof.** By case analysis on Definition 2.1, using Lemma 2.3. \( \square \)

**Proposition 2.5** (Preservation). If \( \emptyset \vdash P : \text{OK} \) and \( P \rightarrow Q \) then \( \emptyset \vdash Q : \text{OK} \).

**Proof.** By rule induction on \( P \rightarrow Q \), using Lemmas 2.2 and 2.4. \( \square \)

### 2.3 A behavioural equivalence

We focus on reasoning about reduction-closed barred congruence \([6, 7, 14, 11, 10]\) of closed, well-typed processes, but in this section we content ourselves with a simplified version of it. We write \( P \mathrel{R} P' \) when \( R \) is a binary relation on closed, well-typed processes and \((P, P') \in R\).

We consider the basic observable of a process to be the ability to output on a given channel, called a barb.

**Definition 2.6** (Barbs). We write \( P \downarrow b \) if and only if there exist \( \overline{c}, V, t, P_1, P_2, \) with \( b \not\in \{\overline{c}\} \), such that \( P \equiv \nu \overline{c}. (b!(V:t).P_1 \mid P_2) \).

We write \( P \downarrow b \) if and only if there exists \( Q \) such that \( P \rightarrow^* Q \) and \( Q \downarrow b \).
Definition 2.7 (Parallel Semantic Equivalence ($\equiv_{\text{pcxt}}$)). ($\equiv_{\text{pcxt}}$) is the largest relation on closed processes that preserves barbs, is reduction closed, and is preserved by parallel contexts; i.e. $P \equiv_{\text{pcxt}} P'$ if and only if

(i) Barb preserving: for all $b$, $P \downarrow b$ iff $P' \downarrow b$.

(ii) Reduction closed: for all $P_1$ with $P \rightarrow P_1$ there exists $P'_1$ such that $P' \rightarrow^* P'_1$ and $P_1 \equiv_{\text{pcxt}} P'_1$, and vice-versa, and

(iii) Preserves parallel constructs: for all well-typed processes $Q$, $P \mid Q \equiv_{\text{pcxt}} P' \mid Q$.

It is straightforward to show that $\equiv_{\text{pcxt}}$ is an equivalence relation. On the other hand to give a direct proof that two processes are related is very difficult, especially in the higher-order $\pi$-calculus. In the following sections we define a labelled transition system (LTS) and show that ($\equiv_{\text{pcxt}}$) coincides with weak bisimulation in the LTS. We also demonstrate the usefulness of bisimulation as a proof technique of equivalence via several examples. Finally we show that the equivalence remains unchanged if we extend the third requirement (3) to demand that the relation be preserved by all contexts.

3 The Labelled Transition System

The idea behind an LTS-based semantics for a process language is to describe the interactions which an observer can have with processes; indeed semantic equivalences such as bisimulation equivalence can be expressed in terms of games, and strategies for such games, over these interactions [16].

We first give an informal account of the kinds of interactions we envision for pp-$\pi$ and then consider their formalisation. For the standard (first-order) $\pi$-calculus observers interact with processes via inputs and outputs on channels. But these interactions are constrained by the knowledge which the observer has of the process being interrogated. For example if an observer has no knowledge of channel $b$ then it can not distinguish between the two processes $a! \cdot 0 \mid b! \cdot 0$ and $a! \cdot 0$ as the only possible known source of interaction is the channel $a$.

In general the observer’s knowledge is accumulated by receiving values from the process under interrogation. In pp-$\pi$ the observer also accumulates knowledge about higher-order values, and may use these to further interrogate the process. This further interrogation can either take the form of transmitting these values along communication channels or executing them. For example consider the two processes

$P \overset{\text{def}}{=} va.c!(\lambda a! \cdot 0).a?. \cdot 0 \quad Q \overset{\text{def}}{=} va.c!(\lambda a! \cdot 0).a?.c!. \cdot 0$

and an observer which only knows of the channel $c$. By inputting on $c$ it gains knowledge of the (suspended) process $a! \cdot 0$ although it does not gain any knowledge of the existence of the private channel $a$. Nevertheless by running this suspended process a difference can be detected between $P$ and $Q$: in one case output can be detected on channel $c$ after the execution of the suspended process.

However, even in the first-order case, it is necessary for the observer to independently generate new values with which to interrogate the process. For example consider a situation in which the observer is only aware of the channel name $a$. Then the only way for the observer to distinguish between the two processes

$a? (x). \cdot 0 \quad a? (x). \text{if } x = a \text{ then } \cdot 0 \text{ else } a! \cdot 0$
is to generate a new channel name, say \( b \), and send this as input along the known channel \( a \).

In pp-\( \pi \) it is also necessary for the observer to generate new higher-order values with which to interrogate the process, by sending them as inputs. However in our LTS these new higher-order values are simply abstract constants, ranged over by \( \alpha \), taken from a countable set \( \text{AConstant} \). On receiving such an abstract higher-order value \( \alpha \) the processes under interrogation has very little it can do with it; \( \alpha \) can only be transmitted as a value along other channels. However, as we will see, our LTS will also allow the process to apply \( \alpha \) in a trivial manner. To accommodate these abstract values we need to extend the syntax in Figure 2 to allow them to be used as values. We let \( \mathcal{P} \) and \( \mathcal{V} \) range over the extended syntax of abstract processes, \( \text{AProcess} \), and abstract values, \( \text{AValue} \), respectively; \( \text{acon} (\mathcal{P}) \) denotes the set of abstract constants occurring in \( \mathcal{P} \). Furthermore we extend the typing rules to apply to abstract processes and values by adding the following typing judgement for abstract constants:

\[
\Gamma \vdash \alpha : \text{Pr}
\]

In order to formalise these kinds of interactions our LTS needs to take into account both the process being interrogated and the current knowledge of the observer, or context. As we have indicated this knowledge is accumulated via interactions with the process, and consists either of (first-order) channel names or higher-order values. To tabulate the latter we use a countable set of concrete constants \( \text{CConstant} \), disjoint from other kinds of constants, and ranged over by \( \kappa \).

**Definition 3.1 (Knowledge environments).** A knowledge environment \( \Delta \) is a finite set of the kind

\[
\text{Name} \cup \text{AConstant} \cup (\text{CConstant} \rightarrow \text{fin} \text{AValue})
\]

with the property that it maps concrete constants to abstract values of type \( \text{Pr} \):

\[
\Delta (\kappa) = \mathcal{V} \text{ implies } \vdash \mathcal{V} : \text{Pr}
\]

We write \( \Delta (\kappa) = \mathcal{V} \) for \( (\kappa, \mathcal{V}) \in \Delta \); we also write \( \text{names}(\Delta) \), \( \text{acon}(\Delta) \), and \( \text{ccon}(\Delta) \) for the name component, the abstract constant component, and the domain of the functional component of \( \Delta \), respectively.

Our LTS will be defined between configurations of the form \( \nu \pi (\Delta, \mathcal{P}) \), where \( \pi \) are names the scope of which extends to \( \mathcal{P} \) and the processes indexed in \( \Delta \). \( \mathcal{P} \) is an abstract process and \( \Delta \) is a knowledge environment. Configurations are identified up to alpha-equivalence, are ranged over by \( C \), and are subject to the following well-formedness constraints:

**Definition 3.2 (Well-Formed Configuration).** A well-formed configuration is any configuration \( \nu \pi (\Delta, \mathcal{P}) \) with the properties:

(i) \( \pi \) are distinct bound names

(ii) \( \{ \pi \} \cap \text{names}(\Delta) = \emptyset \)

(iii) \( \forall \mathcal{P} : \text{OK} \text{ and } \text{fn}(\mathcal{P}) \subseteq \{ \pi \} \cup \text{names}(\Delta) \text{ and } \text{acon}(\mathcal{P}) \subseteq \text{acon}(\Delta) \)

(iv) \( \forall \mathcal{V} : \text{Pr} \text{ and } \text{fn}(\mathcal{V}) \subseteq \{ \pi \} \cup \text{names}(\Delta) \text{ and } \text{acon}(\mathcal{V}) \subseteq \text{acon}(\Delta) \text{ for every } \mathcal{V} \text{ in the codomain of } \Delta. \)

In a configuration \( \nu \pi (\Delta, \mathcal{P}) \) the environment \( \Delta \) represents the knowledge of the observer. The names \( \pi \) are those known to the process under investigation \( \mathcal{P} \), which are not known to the observer, motivating condition (ii); note however that these private names are shared between the process and the abstract values indexed in \( \Delta \), values sent to the observer from the
process. The remaining conditions guarantee that all processes and values must be well-typed and only use names which are in \( \Delta \) or are known to the environment, and abstract constants in \( \Delta \). For the remainder of this paper we only consider well-formed configurations.

The judgements for the LTS take the form

\[
\nu \Delta, \, \nu \mathcal{P} \xrightarrow{\eta} \nu \Delta', \, Q
\]

and the rules for generating them are given in Figure 4 and Figure 5. The label \( \eta \) can take one of the following forms:

(i) \textit{Internal action}, \( \tau \): these are the unobservable actions of the process (e.g. internal communication) and weakly correspond to the semantics of Figure 3.

(ii) \textit{First-order input}, \( \ell\!n \): input by the process along the channel \( \ell \), known to the observer, of the name \( n \); \( n \) is picked by the observer and (due to well-formedness conditions) it might already be recorded in the knowledge environment or is freshly generated—in both cases \( n \) is recorded in the knowledge of the observer after the transition.

(iii) \textit{Higher-order input}, \( \ell\!a \): input by the process of an abstract constant \( a \), which is always taken to be fresh.

(iv) \textit{First-order output}, \( \ell\!n \): output by the process along the known channel \( \ell \) of the name \( n \). Here the name \( n \) may be private to the process or known to the observer. In both cases, \( n \) is known to the observer after the transition.

(v) \textit{Higher-order output}, \( \ell\!k \): output by the process of some value along the channel \( \ell \). The concrete constant \( k \) is picked fresh and the actual value output by the process is indexed by \( k \) in \( \Delta \).

(vi) \textit{Abstract value application}, \( \text{app } \alpha \): signals the execution by the process of the abstract higher-order value \( \alpha \) supplied by the observer. The computational effect of this transition is effectively a noop.

---

<table>
<thead>
<tr>
<th>Name-In-Trans</th>
<th>Proc-In-Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c \in \text{names}(\Delta) )</td>
<td>( c \in \text{names}(\Delta) )</td>
</tr>
<tr>
<td>( \nu \Delta, , \nu \mathcal{P} \xrightarrow{\gamma} \nu \Delta[n], , \nu \mathcal{P}[n/x] )</td>
<td>( \nu \Delta, , \nu \mathcal{P} \xrightarrow{\gamma} \nu \Delta[\alpha], , \nu \mathcal{P}[\alpha/x] )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name-Out-Trans</th>
<th>Proc-Out-Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c \in \text{names}(\Delta) )</td>
<td>( c \in \text{names}(\Delta) )</td>
</tr>
<tr>
<td>( \nu \Delta, , \nu \mathcal{P} \xrightarrow{\gamma} \nu \Delta[n], , \nu \mathcal{P} )</td>
<td>( \nu \Delta, , \nu \mathcal{P} \xrightarrow{\gamma} \nu \Delta[\kappa \to V], , \nu \mathcal{P} )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comm-Name-Trans</th>
<th>Conc-Proc-Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu \mathcal{P}_1 )</td>
<td>( \nu \mathcal{P}_1 )</td>
</tr>
<tr>
<td>( \nu \mathcal{P}_2 )</td>
<td>( \nu \mathcal{P}_2 )</td>
</tr>
<tr>
<td>( \nu \mathcal{P}_3 )</td>
<td>( \nu \mathcal{P}_3 )</td>
</tr>
<tr>
<td>( \nu \mathcal{P}_4 )</td>
<td>( \nu \mathcal{P}_4 )</td>
</tr>
<tr>
<td>( \nu \mathcal{P}_5 )</td>
<td>( \nu \mathcal{P}_5 )</td>
</tr>
<tr>
<td>( \nu \mathcal{P}_6 )</td>
<td>( \nu \mathcal{P}_6 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Abs-App-Trans</th>
<th>Proc-Proc-Trans</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \nu \Delta, , \nu \mathcal{P} \xrightarrow{\text{app } \alpha} \nu \Delta, , \nu \mathcal{P} )</td>
<td>( \nu \Delta, , \nu \mathcal{P} )</td>
</tr>
</tbody>
</table>
Suppose Proposition 3.3.

An important aspect of our LTS is the handling of names and extrusion. A (sub-) process with a top-level $\nu$-binder can only take a $\tau$-transition, which lifts the binder to the level of the configuration (Nu-Trans). Since configurations are identified up to renaming bound names, this is essentially an extrusion step. When such a bound name is revealed to the observer via an output transition, the binder is removed and the name is added in knowledge environment (Name-Out-Trans). In this way we greatly simplify the labels of the transitions in our LTS, which allows us to give a propositional characterisation of weak bisimilarity, as we will see in Section 4.3.

We require that communication with the observer occurs over known channels, and that the observer never provides a private channel as an input. Hence, a transition

$$\nu\bar{\pi}\langle \Delta, \mathcal{P} \rangle \xrightarrow{\eta} \nu\bar{\pi}\langle \Delta', \mathcal{Q} \rangle$$

happens only if

$$s(\eta) \cap [\bar{a}] = \text{inp}(\eta) \cap [\bar{a}] = \emptyset$$

where $s(-)$ and $\text{inp}(-)$ return a singleton set containing, respectively, the subject of a communication action, the object of an input action; in all other cases they return the empty set.

Internal communication for first-order values is captured by Comm-Name-Trans, and for higher-order values by Comm-Proc-Trans. Such communication can take place over channels that are local to the process, hence the temporary addition of the local channels in $\Delta$ in the premises. Note that in Comm-Proc-Trans, the concrete constant $\kappa$ used to temporarily store the value being communicated is not included in the environment $\Delta$ in the conclusion.

In rule Par-L-Trans all the private names bound by the configuration are temporarily added in $\Delta$ to avoid their alpha renaming without the corresponding renaming of names in $\mathcal{Q}$. The side condition ensures that names exported to the observer are properly added to the knowledge environment: $\text{exp}(-)$ returns the object of an output action and the empty set otherwise.

The LTS only increases the knowledge of the observer.

**Proposition 3.3.** Suppose $\nu\bar{\pi}\langle \Delta_1, \mathcal{P} \rangle \xrightarrow{\eta} \nu\bar{\pi}\langle \Delta_2, \mathcal{Q} \rangle$; then $\Delta_1 \subseteq \Delta_2$. 
Proof. By straightforward rule induction on the transition relation. \hfill \Box

For the rest of this subsection we analyse in considerable detail the structure of the actions in the LTS. First we give an exhaustive analysis of the structure of configurations which are produced by these actions.

**Proposition 3.4.** The following properties are true.

(i) If $v\alpha_\Delta(P) \xrightarrow{cn} v\beta_\Delta(Q)$ then for some $P_1$ and $P_2$

$$P \models c!(n).P_1 \mid P_2 \quad Q \models P_1 \mid P_2 \quad \Delta_2 = \Delta_1 \cup \{n\} \quad \{\bar{b}\} = [\bar{a}] \setminus \{n\}$$

(ii) If $v\alpha_\Delta(P) \xrightarrow{ck} v\beta_\Delta(Q)$ then for some $P_1$ and $P_2$

$$P \models c!(V).P_1 \mid P_2 \quad Q \models P_1 \mid P_2 \quad \Delta_2 = \Delta_1 \cup \{\kappa \rightarrow V\} \quad \{\bar{b}\} = [\bar{a}]$$

(iii) If $v\alpha_\Delta(P) \xrightarrow{cn} v\beta_\Delta(Q)$ then for some $P_1$ and $P_2$

$$P \models c!(x:nm).P_1 \mid P_2 \quad Q \models P_1 \mid P_2 \quad \Delta_2 = \Delta_1 \cup \{n\} \quad \{\bar{b}\} = [\bar{a}]$$

(iv) If $v\alpha_\Delta(P) \xrightarrow{ck} v\beta_\Delta(Q)$ then for some $P_1$ and $P_2$

$$P \models c!(x:Pr).P_1 \mid P_2 \quad Q \models P_1 \mid P_2 \quad \Delta_2 = \Delta_1 \cup \{\kappa\} \quad \{\bar{b}\} = [\bar{a}]$$

(v) If $v\alpha_\Delta(P) \xrightarrow{app} v\beta_\Delta(Q)$ then for some $P_1$

$$P \models \text{app } a \mid P_1 \quad Q \models P_1 \quad \Delta_2 = \Delta_1 \quad \{\bar{b}\} = [\bar{a}]$$

(vi) If $v\alpha_\Delta(P) \xrightarrow{app} v\beta_\Delta(Q)$ then for some $V = \Delta_1(\kappa)$

$$Q \models \text{app } V \mid P \quad \Delta_2 = \Delta_1 \quad \{\bar{b}\} = [\bar{a}]$$

(vii) If $v\alpha_\Delta(P) \xrightarrow{\tau} v\beta_\Delta(Q)$ then $\Delta_2 = \Delta_1$ and $[\bar{a}] \subseteq \{\bar{b}\}$.

Proof. All properties are shown by rule induction on the transition relation. \hfill \Box

In a configuration $v\alpha_\Delta(P)$ there are two sources of knowledge, the environment’s knowledge in $\Delta$ and the internal knowledge of the process in $\bar{a}$. The next result shows that changes to this knowledge has no effect on many actions.

**Proposition 3.5.**

(i) Knowledge extension: If $v\alpha_\Delta(P) \xrightarrow{\eta} v\beta_\Delta(Q)$ and $v\alpha_\Delta,\bar{e}_\Delta(\Delta_0 \uplus \Delta_1, P)$ is well-formed, and $\text{names}(\eta) \cap \text{names}(\Delta_0) = \text{acon}(\eta) \cap \text{acon}(\Delta_0) = \text{ccon}(\eta) \cap \text{ccon}(\Delta_0) = \emptyset$ then

$$v\alpha_\Delta,\bar{e}_\Delta(\Delta_0 \uplus \Delta_1, P) \xrightarrow{\eta} v\beta_\Delta,\bar{e}_\Delta(\Delta_0 \uplus \Delta_2, Q)$$
(ii) Knowledge restriction: If \( \nu \bar{a}, \bar{c} (\Delta_0 \uplus \Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu \bar{b} (\Delta_0 \uplus \Delta_2, \mathcal{Q}) \) and \( \nu \bar{a} (\Delta_1, \mathcal{P}) \) is well-formed, and \( \eta \notin \{\text{app } \kappa \mid \kappa \in \Delta_0 \} \cup \{c\bar{c}n \mid n \in \Delta_0\} \) then

\[
\nu \bar{a} (\Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu \bar{b} (\Delta_2, \mathcal{Q})
\]

where \( \{\bar{b}\} = \{\bar{c}\}\).  

Proof. In both cases we use rule induction on the transition relation. \(\square\)

Information in \( \nu \bar{a} (\Delta, \mathcal{P}) \) can also be shifted between the observers knowledge \( \Delta \) and the processes knowledge \( \bar{a} \) without affecting actions, provided of course that information is not used in the actions.

**Proposition 3.6** (Unused Information).

(i) Hiding: Suppose \( \nu \bar{a} (\Delta_1 \cup \{b\}, \mathcal{P}) \xrightarrow{\eta} \nu \bar{a} (\Delta_2 \cup \{b\}, \mathcal{Q}) \) and \( b \) does not occur in \( \eta \). Then

\[
\nu b, \bar{a} (\Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu b, \bar{a} (\Delta_2, \mathcal{Q})
\]

(ii) Revealing: Conversely, suppose \( \nu b, \bar{a} (\Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu b, \bar{a} (\Delta_2, \mathcal{Q}) \) where again \( b \) does not occur in \( \eta \). Then

\[
\nu \bar{a} (\Delta_1 \cup \{b\}, \mathcal{P}) \xrightarrow{\eta} \nu \bar{a} (\Delta_2 \cup \{b\}, \mathcal{Q})
\]

Proof. Again, by rule induction. \(\square\)

With reference to this proposition there are actually very limited ways in which an action \( \eta \) from the configuration \( \nu b, \bar{a} (\Delta_1, \mathcal{P}) \) can use the name \( b \). Indeed the only possibility is an output action, which by Proposition 3.4 must have the form \( \nu b, \bar{a} (\Delta, \mathcal{P}) \xrightarrow{c b} \nu \bar{a} (\Delta \cup \{b\}, \mathcal{Q}) \); and this action can still be performed when the observer knows of the existence of \( b \):

**Proposition 3.7** (Extrusion). Provided \( c \) is different than \( b \),

\[
\nu b, \bar{a} (\Delta, \mathcal{P}) \xrightarrow{c b} \nu \bar{a} (\Delta \cup \{b\}, \mathcal{Q}) \text{ iff } \nu \bar{a} (\Delta \cup \{b\}, \mathcal{P}) \xrightarrow{c b} \nu \bar{a} (\Delta \cup \{b\}, \mathcal{Q})
\]

Proof. By rule induction, in both directions. \(\square\)

Abstract constants are significant only in application and communication actions that mention them—substituting a value for an abstract constant leaves all other actions unaffected. Similarly, values are significant only in application steps—abstracting away values leaves other actions unaffected.

**Proposition 3.8**.

(i) Substitution: Suppose \( \nu \bar{a} (\Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu \bar{b} (\Delta_2, \mathcal{Q}) \) and \( \nu \bar{a} (\Delta_1 \{V/\alpha\}, \mathcal{P}\{V/\alpha\}) \) is well-formed and \( \alpha \) does not occur in \( \eta \). Then

\[
\nu \bar{a} (\Delta_1 \{V/\alpha\}, \mathcal{P}\{V/\alpha\}) \xrightarrow{\eta} \nu \bar{b} (\Delta_2 \{V/\alpha\}, \mathcal{Q}\{V/\alpha\})
\]

(ii) Abstraction: Let \( \nu \bar{a} (\Delta_1 \{V/\alpha\}, \mathcal{P}\{V/\alpha\}) \xrightarrow{\eta} \nu \bar{b} (\Delta_2 \{V/\alpha\}, \mathcal{Q}\{V/\alpha\}) \) is well-formed and \( \eta \) is not a \( \tau \) action involving the rule APP-TRANS or an app a action. Then

\[
\nu \bar{a} (\Delta_1, \mathcal{P}) \xrightarrow{\eta} \nu \bar{b} (\Delta_2, \mathcal{Q})
\]

Proof. Both properties are shown by rule induction. \(\square\)
4 Bisimulations

In this section we give the definitions for strong and weak bisimulations. We prove that the limited structural equivalence (\(\equiv\)) is a strong bisimulation and the full structural equivalence (\(\cong\)) is a weak bisimulation over configurations. We also prove several useful weak bisimulations that encode properties of local and global names. Finally we give a characterisation of weak bisimilarity in terms of a propositional Hennessy-Milner Logic.

4.1 Strong Bisimulations

We start with the definition of strong bisimulation, a rather strict equivalence on configurations which will be useful later for deriving technical results.

We write binary relations on well-formed configurations as \(R\), \(X\), etc.

**Definition 4.1** (Strong Bisimulation). \(R\) is a strong bisimulation if and only if for all \(C R C'\):

(i) If \(C \xrightarrow{\eta} C_1\) then there exists \(C' \xrightarrow{\eta} C'_1\) and \(C_1 R C'_1\).

(ii) The converse of (i)

Strong bisimulations are closed under unions. Thus the union of all strong bisimulations is the largest strong bisimulation; it is also easy to see that it is an equivalence relation.

**Definition 4.2** (Strong Bisimilarity (\(\sim\))). (\(\sim\)) is the largest strong bisimulation.

The limited structural equivalence from Definition 2.1 can be extended to configurations in the obvious manner. First it is extended to abstract processes by applying the axioms and rules in Definition 2.1. Then we let \(\nu a \langle \Delta, P \rangle \equiv \nu a' \langle \Delta', P' \rangle\) whenever \(P \equiv P', a = a'\) and \(\Delta = \Delta'\).

**Proposition 4.3.** (\(\equiv\)) is a strong bisimulation over configurations.

**Proof.** By using induction on the rules of (\(\equiv\)); i.e. the rules shown in Definition 2.1 and the standard rules for an equivalence, we can show that all moves from related configurations can be appropriately matched. \(\square\)

4.2 Weak Bisimulations

Our theory of behavioural equivalence is based on weak bisimulations, which use weak actions from the LTS of the previous section. We write \(\xrightarrow{\eta}\) to mean the reflexive, transitive closure of \(\xrightarrow{\tau}\), when \(\eta = \tau\), and \(\xrightarrow{\eta}\), otherwise.

**Definition 4.4** (Weak Bisimulation). \(R\) is a bisimulation if and only if for all \(C R C'\):

(i) If \(C \xrightarrow{\eta} C_1\) then there exists \(C' \xrightarrow{\eta} C'_1\) and \(C_1 R C'_1\).

(ii) The converse of (i)

The collection of weak bisimulations is closed under unions, and thus the union of all weak bisimulations is the largest weak bisimulation; again it is straightforward to show that this is also an equivalence relation.

**Definition 4.5** (Weak Bisimilarity (\(\approx\))). (\(\approx\)) is the largest weak bisimulation.

**Lemma 4.6.** If \(\nu a \langle \Delta, P \rangle \approx \nu a' \langle \Delta', P' \rangle\) then \(ccon(\Delta) = ccon(\Delta')\).
Proof (by contradiction). Let $\kappa \in \text{ccon}($Δ$)$ and $\kappa \notin \text{ccon}(\Delta')$; then $\nu\alpha(\Delta, P)$ has an app-$\kappa$-transition to another configuration but $\nu\bar{\alpha}(\Delta', P')$ does not, which contradicts the premise.

Our primary concern is the ability for our bisimulations to support reasoning about process behaviour. To this end we extend weak bisimilarity to closed processes as follows:

**Definition 4.7.** We write $P \equiv P'$ if and only if there exist $b$ such that

$$\langle [\nu b], P \rangle \approx \langle [\nu b], P' \rangle$$

Note that since ($\equiv$) is only defined between well-formed configurations the names $\nu b$ in the above definition include the free names of $P$ and $P'$.

As with ($\equiv$), we extend the structural equivalence ($\equiv$) to abstract processes in the usual way, and to LTS configurations as follows; note that this is extension is slightly more general than that used for the limited structural equivalence ($\sim$).

**Definition 4.8 (($\equiv$) on LTS configurations).** We write $\nu\alpha(\Delta, P) \equiv \nu\bar{\alpha}(\Delta', P')$ if and only if

$$\nu\alpha. P \equiv \nu\bar{\alpha}. P' \quad \Delta = \Delta'$$

**Proposition 4.9.** ($\equiv$) is a weak bisimulation over configurations.

Proof (sketch). Suppose

$$\nu\alpha(\Delta_1, P) \xrightarrow{\eta} \nu\bar{\alpha}(\Delta_2, Q) \quad \text{and} \quad \nu\alpha(\Delta_1, P) \equiv \nu\bar{\alpha}(\Delta', P')$$

We show that

$$\nu\bar{\alpha}(\Delta_1', P') \xrightarrow{\eta} \nu\bar{\alpha}(\Delta_2', Q')$$

for some $\nu\bar{\alpha}(\Delta_2', Q') \equiv \nu\bar{\alpha}(\Delta_2, Q)$, and vice-versa.

We proceed by induction on the proof that $\nu\bar{\alpha}. P \equiv \nu\bar{\alpha}. P'$. The base cases are provided by the axioms for ($\equiv$) in Definition 2.1 and reflexivity. The only complication here involves any outermost $\nu$-binders in the processes of the axioms: for each binder we distinguish the case where it is a binder at the level of the configuration and the case where it is a binder in the process part of the configuration. In all cases, the behaviour of the related processes are identical, modulo the $\tau$-transitions that extrude a binder to the level of the configuration.

The cases for symmetry and transitivity are shown by straightforward applications of the induction hypothesis.

For the case of closure under ($-| -$) we have $\nu\bar{\alpha}. P = P_1 \mid P_2$ and $\nu\bar{\alpha}. P' = P'_1 \mid P'_2$, for some $P_1, P_2, P'_1,$ and $P'_2$, with $P_1 \equiv P'_1$ and $P_2 \equiv P'_2$. Hence $[\alpha] = [\bar{\alpha}] = 0$. We proceed by cases on the $\eta$-transition. The only applicable cases are Comm-Name-Trans, Comm-Proc-Trans, and Par-L-Trans, which are all proved by straightforward applications of the induction hypothesis and Proposition 3.4.

The case for closure under ($\nu n. -$) follows by the induction hypothesis and Propositions 3.6 and 3.7.

**Corollary 4.10.** ($\equiv \subseteq$ ($\equiv$)).

Extension of knowledge environments with identical names preserves weak bisimilarity.

**Lemma 4.11.** If $\nu\alpha(\Delta, P) \approx \nu\bar{\alpha}(\Delta', P')$ and $n \notin [\alpha, \bar{\alpha}]$ then

$$\nu\alpha(\Delta \cup \{n\}, P) \approx \nu\bar{\alpha}(\Delta' \cup \{n\}, P')$$
Proof. Let

\[ \mathcal{X} = \{(νa⟨Δ∪[n], P⟩, νa'⟨Δ'∪[n], P'⟩) \mid νa⟨Δ, P⟩ \approx νa'⟨Δ', P'⟩ \} \]

It is easy to show that \( \mathcal{X} \) is a weak bisimulation using Proposition 3.5. □

Hiding names also preserves weak bisimilarity.

Lemma 4.12. If \( νa⟨Δ [n], P⟩ \approx νa'⟨Δ' [n], P'⟩ \) then

\[ νa,n⟨Δ, P⟩ \approx νa',n⟨Δ', P'⟩ \]

Proof. Similar to the above proof. □

Furthermore, extrusion of private names to the level of the configuration is indistinguishable by weak bisimilarity.

Lemma 4.13. \( νa,b⟨Δ, P⟩ \approx νb⟨Δ, νa.P⟩ \)

Proof. Trivial. □

Lemma 4.14. \( νa,b⟨Δ, P⟩ \approx νa',b'⟨Δ', P'⟩ \) if \( νb⟨Δ, νa.P⟩ \approx νb'⟨Δ', νa'.P'⟩ \)

Proof. By Lemma 4.13 and transitivity of \( \approx \). □

4.3 Logical Characterisation

Weak bisimilarity is characterised by a propositional Hennessy-Milner Logic with the following syntax.

\[ F ::= \neg F \mid \bigwedge_{i \in I} F_i \mid \langle \eta \rangle F \]

where \( I \) is a (possibly infinite) indexing set.

These formulas define a set of basic properties satisfied by configurations of our LTS. The construct \( \neg F \) encodes negation and \( \bigwedge_{i \in I} F_i \) encodes (possibly infinite) propositional conjunction. The modal construct \( \langle \eta \rangle F \) encodes the property that there is a weak \( \eta \)-transition to a configuration that satisfies \( F \).

The semantics of this logic is given by a satisfaction relation \( C \models F \) between a configuration \( C \) and a formula \( F \).

Definition 4.15 (Satisfaction Relation \( (C \models F) \)).

\[ C \models \neg F \iff C \not\models F \]
\[ C \models \bigwedge_{i \in I} F_i \iff \forall i \in I, C \models F_i \]
\[ C \models \langle \eta \rangle F \iff \exists C'. C \xrightarrow{\eta} C' \text{ and } C' \models F \]

As usual, more predicates are derivable; e.g.:

\[ C \models tt \overset{\text{def}}{=} C \models \bigwedge_{i \in I} F_i \]
\[ C \models ff \overset{\text{def}}{=} C \models \neg tt \]
\[ C \models [\eta] F \overset{\text{def}}{=} C \models \neg(\langle \eta \rangle \neg F) \]
\[ C \models \bigvee_{i \in I} F_i \overset{\text{def}}{=} C \models \neg \bigwedge_{i \in I} \neg F_i \]
\[ C \models F_1 \land F_2 \overset{\text{def}}{=} C \models \bigwedge_{i \in \{1,2\}} F_i \]
\[ C \models F_1 \lor F_2 \overset{\text{def}}{=} C \models \bigvee_{i \in \{1,2\}} F_i \]
As the transition labels $\eta$ in our LTS contain actual (not extruded) names, i.e. constants, the above logic is similar to that of the CCS ([9], Chapter 10). Hence, we avoid the complications of extrusion and generation of fresh names in the logic.

The main theorem in this section is the characterisation of weak bisimilarity by the logic.

**Theorem 4.16.** $C \approx C'$ if and only if for all $F$

$$C \models F \iff C' \models F$$

**Proof.** For the forward direction we proceed by structural induction, taking cases on the formula $F$:

**Case** $F = \neg F_0$: By the induction hypothesis,

$$C \models F_0 \iff C' \models F_0$$

hence, by Definition 4.15, $C \models \neg F_0 \iff C' \models \neg F_0$.

$\diamondsuit F = \bigwedge_{i \in I} F_i$: by Definition 4.15,

$$C \models \bigwedge_{i \in I} F_i \iff (\forall i \in I. C \models F_i)$$

$$C' \models \bigwedge_{i \in I} F_i \iff (\forall i \in I. C' \models F_i)$$

By the induction hypothesis,

$$\forall i \in I. C \models F_i \iff C' \models F_i$$

and by the definition of the (possibly infinite) conjunction $C \models \bigwedge_{i \in I} F_i \iff C' \models \bigwedge_{i \in I} F_i$.

$\diamondsuit F = \langle \eta \rangle F_0$: If $C \models \langle \eta \rangle F_0$ then, by Definition 4.15, there exists $C_0$ such that

$$C \xrightarrow{\eta} C_0 \quad C_0 \models F_0$$

Because $C \approx C'$, there exists $C_0'$ such that

$$C' \xrightarrow{\eta} C_0' \quad C_0' \models C_0$$

Hence, by the induction hypothesis, it must be that $C_0' \models F_0$, and, by Definition 4.15, $C' \models \langle \eta \rangle F_0$. Similarly if $C' \models \langle \eta \rangle F_0$.

For the converse direction of the theorem we define the following relation.

$$R = \{ (C, C') \mid \forall F. C \models F \iff C' \models F \}$$

We show by contradiction that $R$ is a weak bisimulation:

We assume that $R$ is not a bisimulation. Because $R$ is obviously symmetric, w.l.o.g., this means that for some $(C, C') \in R$ there exists $C_1$ such that

$$C \xrightarrow{\eta} C_1 \quad \forall C'_i \in S. (C_1, C'_i) \notin R$$

where $S = \{ C'_i \mid C' \xrightarrow{\eta} C'_i \}$. By the definition of $R$, for every $C'_i \in S$ there exists $F_i$ such that

$$C_1 \models F_i \quad C'_i \not\models F_i$$

17
or vice-versa, but in this case we consider \( \neg F_i \). Hence, if \( I \) contains the indices of exactly these formulas,

\[
C_1 \models \bigwedge_{i \in I} F_i
\]

and therefore

\[
C \models \langle \eta \rangle \left( \bigwedge_{i \in I} F_i \right) \quad C' \not\models \langle \eta \rangle \left( \bigwedge_{i \in I} F_i \right)
\]

which contradicts the fact that \((C, C') \in \mathbb{R}\). \( \square \)

An immediate consequence of this theorem is that the logic is particularly useful in giving simple proofs of inequivalence. In Section 8.5 we prove such an inequivalence by providing an HML formula that is satisfied by one of the processes and not the other.

5 Soundness of Weak Bisimilarity

In this section we prove that weak bisimulation equivalence \( (\simeq) \) satisfies the defining properties of parallel contextual equivalence \( (\sim_{pcxt}) \) and therefore is included in it. For convenience it is divided into three sub-sections. The first establishes a close relationship between the reduction semantics of Section 2 and the \( \tau \)-moves in the LTS semantics of Section 3. The second sub-section proves that \((\simeq)\) is reduction-closed and preserves barbs, while in the final sub-section is devoted to the most difficult property, preservation by parallel contexts.

5.1 Reductions versus \( \tau \)-steps

To prove that \((\simeq)\) is reduction-closed we first need to show that \( \tau \)-transitions correspond to reduction steps.

**Lemma 5.1.** If \( \nu \sigma \langle \{c\}, P \rangle \xrightarrow{\tau} \nu \sigma \langle \{c\}, Q \rangle \) then \( \nu \sigma \cdot P \rightarrow^{*} \nu \sigma \cdot Q \).

**Proof.** By induction on the transition \( \nu \sigma \langle \{c\}, P \rangle \xrightarrow{\tau} \nu \sigma \langle \{c\}, Q \rangle \). The cases **Cond-True-Trans**, **Cond-False-Trans**, **Rec-Trans**, **Nu-Trans**, and **App-Trans** are trivial.

**Case** **Par-L-Trans:** we have

\[
\begin{align*}
\langle \Delta_1 \cup \{\sigma\}, P_1 \rangle \xrightarrow{\tau} & \nu \sigma \langle \Delta_2 \cup \{\sigma\}, P_2 \rangle \\
\langle \sigma \rangle = |\sigma| \exp(\tau) & \\

\nu \sigma \langle \Delta_1, P_1 \mid Q \rangle \xrightarrow{\tau} & \nu \sigma \langle \Delta_2, P_2 \mid Q \rangle
\end{align*}
\]

and want to show that \( \nu \sigma \cdot (P_1 \mid Q) \rightarrow^{*} \nu \sigma \cdot (P_2 \mid Q) \). By the induction hypothesis \( P_1 \rightarrow^{*} \nu \sigma \cdot (P_2 \mid Q) \), and, by the reduction rule **Par-L-Red** in Figure 3, \( P_1 \mid Q \rightarrow^{*} (\nu \sigma \cdot P_2) \mid Q \). By the properties of well-formed configurations and Proposition 3.3, \( \sigma \notin fn(Q) \). Hence, by **Par-Cong-Red** and **Nu-Red**, \( \nu \sigma \cdot (P_1 \mid Q) \rightarrow^{*} \nu \sigma \cdot (P_2 \mid Q) \).

**Case** **Comm-Name-Trans:** we have

\[
\begin{align*}
\langle \{\bar{c}, \bar{a}\}, P_1 \rangle \xrightarrow{\text{cn}} & \langle \Delta', P_2 \rangle \\
\langle \{\bar{c}, \bar{a}\}, Q_1 \rangle \xrightarrow{\text{cn}} & \langle \Delta'', Q_2 \rangle
\end{align*}
\]

\[
\nu \sigma \langle \{\bar{c}, \bar{a}\}, P_1 \mid Q_1 \rangle \xrightarrow{\tau} \nu \sigma \langle \{\bar{c}, \bar{a}\}, P_2 \mid Q_2 \rangle
\]

18
and want to show that \( \nu \alpha. P_1 | Q_1 \rightarrow^* \nu \alpha. P_2 | Q_2 \). By Proposition 3.4 (i) and (iii) we get that

\[
\begin{align*}
P_1 &= c(n;\text{Nm}). P_{11} | P_{12} \\
P_2 &= P_{11} | P_{12} \\
Q_1 &= c(n;\text{Nm}). Q_{11} | Q_{12} \\
Q_2 &= Q_{12}[n/x] | Q_{22}
\end{align*}
\]

Thus, by \textsc{Comm-Red}, \textsc{Cong-Red}, \textsc{Par-Red}, and \textsc{Nu-Red} we get \( \nu \alpha. (P_1 | Q_1) \rightarrow^* \nu \alpha. (P_2 | Q_2) \). Similarly for the case \textsc{Comm-Proc-Trans}. The rest of the cases are vacuously true. \( \square \)

**Lemma 5.2.** If \( P \rightarrow Q \), and \( \mathsf{ft}(P) \subseteq \{ \nu \} \) then there exist \( \alpha \) and \( Q_0 \) such that

\[
\langle \nu \alpha, P \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_0 \rangle \quad \nu \alpha. Q_0 \equiv Q
\]

**Proof.** By induction on \( P \rightarrow Q \). Cases \textsc{Comm-Red}, \textsc{App-Red}, and \textsc{Cond-Red} are straightforward.

**Case **\textsc{Par-Red}: we have

\[
\frac{P \rightarrow P}{P_1 \mid Q \rightarrow P_2 \mid Q}
\]

and want to show that there exist \( \alpha \) and \( Q_0 \) such that \( \langle \nu \alpha, P_1 \mid Q \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_0 \rangle \) and \( \nu \alpha. Q_0 \equiv P_2 \mid Q \). By the induction hypothesis there exist \( \alpha \) and \( P_{20} \) such that \( \langle \nu \alpha, P_1 \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, P_{20} \rangle \) and \( \nu \alpha. P_{20} \equiv P_2 \mid Q \). By \textsc{Par-L-Trans} and well-formedness of configurations

\[
\langle \nu \alpha, P_1 \mid Q \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, P_{20} \mid Q \rangle \quad \nu \alpha. (P_{20} \mid Q) \equiv (\nu \alpha. P_{20}) \mid Q \equiv P_2 \mid Q
\]

**Case **\textsc{Nu-Red}: we have

\[
\frac{P \rightarrow Q}{vb. P \rightarrow vb. Q}
\]

and want to show that there exist \( \alpha \) and \( Q_0 \) such that \( \langle \nu \alpha, vb. P \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_0 \rangle \) and \( \nu \alpha. Q_0 \equiv vb. Q \). By the induction hypothesis there exist \( \alpha \) and \( Q_1 \) such that \( \langle \nu \alpha, P \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_1 \rangle \) and \( \nu \alpha. Q_1 \equiv Q \). By \textsc{Nu-Trans} and Proposition 3.6 (Hiding) we have

\[
\langle \nu \alpha, vb. P \rangle \xrightarrow{\tau} vb \langle \nu \alpha, P \rangle \xrightarrow{\tau} \nu \alpha. vb \langle \nu \alpha, Q_1 \rangle 
\]

\[
\nu \alpha. vb. Q_1 \equiv vb. \nu \alpha. Q_1 \equiv vb. Q
\]

**Case **\textsc{Cong-Red}: we have

\[
\frac{P' \rightarrow Q'}{P \equiv P' \quad Q \equiv Q'}
\]

and want to show that there exist \( \alpha \) and \( Q_0 \) such that \( \langle \nu \alpha, P \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_0 \rangle \) and \( \nu \alpha. Q_0 \equiv Q \). By the induction hypothesis there exist \( \alpha' \) and \( Q_0' \) such that \( \langle \nu \alpha', P \rangle \xrightarrow{\tau} \nu \alpha' \langle \nu \alpha', Q_0' \rangle \) and \( \nu \alpha'. Q_0' \equiv Q' \). By Proposition 4.9 and because \( \langle \nu \alpha, P \rangle \equiv \langle \nu \alpha, P' \rangle \) there exist \( \alpha \) and \( Q_0 \) such that

\[
\langle \nu \alpha, P \rangle \xrightarrow{\tau} \nu \alpha \langle \nu \alpha, Q_0 \rangle 
\]

\[
\nu \alpha \langle \nu \alpha, Q_0 \rangle \equiv \nu \alpha' \langle \nu \alpha, Q_0' \rangle
\]

and by Definition 2.1 \( \nu \alpha. Q_0 \equiv \nu \alpha'. Q_0' \equiv Q' \equiv Q \). \( \square \)
5.2 Reduction-closure and preservation of barbs

We can now prove that \( \equiv \) is reduction-closed.

**Proposition 5.3** (Reduction Closure of \( \equiv \)). If \( P \equiv P' \) and \( P \rightarrow Q \) then there exists \( Q' \) such that:

\[
P' \rightarrow^* Q' \quad Q \equiv Q'
\]

and vice-versa.

**Proof.** We prove only the forward direction, the converse is symmetric. By the first premise and Definition 4.7, there exist \( \bar{b} \) (with \( \text{fn}(P, P') \subseteq \{\bar{b}\} \)) such that

\[
\langle \bar{b}, P \rangle \equiv \langle \bar{b}, P' \rangle
\]

By the second premise and Lemma 5.2 there exist \( \bar{a} \) and \( Q_0 \) such that

\[
\langle \bar{b}, P \rangle \xRightarrow{\tau} \bar{a} \langle \bar{b}, Q_0 \rangle \quad \bar{a} \cdot Q_0 \equiv Q
\]

Thus, by Definition 4.4 and (1), there exist \( \bar{a} ', \Delta ' , \) and \( Q'_0 \) such that

\[
\langle \bar{b}, P \rangle \xRightarrow{\tau} \bar{a} ' \langle \bar{b}, Q'_0 \rangle \quad \bar{a} ' \cdot Q'_0 \equiv Q
\]

and by Proposition 3.4 (vii) \( \Delta ' = \{\bar{b}\} \).

By (2) and Lemma 5.1, \( P' \rightarrow^* \bar{a} ' \cdot \Delta ' , Q'_0 \).

By (3) and Lemma 5.14 \( \langle \bar{b}, \bar{a}, Q_0 \rangle \equiv \langle \bar{b}, \bar{a} ' , Q'_0 \rangle \) and therefore \( \bar{a} \cdot Q_0 \equiv \bar{a} ' \cdot Q'_0 \).

Hence \( Q \equiv \bar{a} ' \cdot Q'_0 \), and by transitivity of \( \equiv \) and Corollary 4.10 we get \( Q \equiv \bar{a} ' \cdot Q'_0 \). \( \square \)

**Proposition 5.4** (Preservation of Barbs of \( \equiv \)). If \( P \equiv P' \) then \( P \not\equiv n \iff P' \not\equiv n \).

**Proof.** We prove only the forward direction, the converse is symmetric. By the second premise and Definition 2.6 we get that there exists \( Q \) such that \( P \rightarrow^* Q , Q \equiv \bar{a} \cdot n \cdot (V : t), Q_1 \mid Q_2 \), and \( n \notin \bar{a} \). By Proposition 5.3 there exists \( Q' \) such that \( P' \rightarrow^* Q' \) and \( Q \equiv Q' \).

By the first premise, transitivity of \( \equiv \), and Corollary 4.10 we get \( \bar{a} \cdot n \cdot (V : t), Q_1 \mid Q_2 \equiv Q' \). Thus, by Definition 4.7, there exist \( \bar{b} \) such that

\[
\langle \bar{b}, n \rangle , \bar{a} \cdot n \cdot (V : t) , Q_1 \mid Q_2 \rangle \equiv \langle \bar{b}, n \rangle , Q' \rangle
\]

and by the transition rules of the LTS we get

\[
\langle \bar{b}, n \rangle , \bar{a} \cdot n \cdot (V : t) , Q_1 \mid Q_2 \rangle \xRightarrow{\text{tm}} \bar{a} \langle \bar{b}, n \rangle \cup \{V\} , Q_1 \mid Q_2 \rangle
\]

if \( t = \text{tm} \) or

\[
\langle \bar{b}, n \rangle , \bar{a} \cdot n \cdot (V : t) , Q_1 \mid Q_2 \rangle \xRightarrow{\text{tm}} \bar{a} \langle \bar{b}, n , \kappa \rightarrow V \rangle , Q_1 \mid Q_2 \rangle
\]

if \( t = \text{Pr} \). By Definition 4.4 and (4), there exist \( \bar{a} ' , \Delta ' , \) and \( Q'_1 \) such that one of the following is true:

\[
\langle \bar{b}, n \rangle , Q' \rangle \xRightarrow{n \cdot \kappa} \bar{a} ' \langle \bar{b}, n \rangle , \Delta ' , Q'_1 \rangle
\]

or

\[
\langle \bar{b}, n \rangle , Q' \rangle \xRightarrow{n \cdot \kappa} \bar{a} ' \langle \bar{b}, n \rangle , \Delta ' , Q'_1 \rangle
\]

Therefore, by Proposition 3.4 (i) or (ii), and (vii), there exist \( Q'_1 \) and \( Q'_2 \) such that

\[
\langle \bar{b}, n \rangle , Q' \rangle \xRightarrow{\text{tm}} \bar{a} ' \langle \bar{b}, n \rangle , n \cdot (V ' : t) , Q'_1 \mid Q'_2 \rangle
\]

and by Lemma 5.1 \( Q' \rightarrow^* \bar{a} ' \cdot n \cdot (V ' : t) , Q'_1 \mid Q'_2 \rangle \) with \( n \notin \bar{a} ' \). Hence \( P' \not\equiv n \). \( \square \)
\textbf{Cxt-R} \\
\( \nu \alpha (\Delta, \mathcal{P}) \equiv \nu \alpha (\Delta', \mathcal{P}') \) \\
\( \text{acon}(\Delta) = \text{acon}(\Delta') \) \\
\( \text{names}(\Delta) = \text{names}(\Delta') \) \\
\( \nu \alpha (\Delta, \mathcal{P}) \equiv^\text{Cxt} \nu \alpha (\Delta', \mathcal{P}') \)

\textbf{Cxt-Par} \\
\( \nu \alpha (\Delta_1, \mathcal{P}_1) \equiv^\text{Cxt} \nu \alpha (\Delta'_1, \mathcal{P}'_1) \) \\
\( \nu \beta (\Delta_2, \mathcal{P}_2) \equiv^\text{Cxt} \nu \beta (\Delta'_2, \mathcal{P}'_2) \)

\( \nu \alpha (\Delta, \mathcal{P}) \equiv^\text{Cxt} \nu \alpha (\Delta', \mathcal{P}') \)

\textbf{Cxt-Hide} \\
\( \nu \alpha (\Delta \downarrow [a], \mathcal{P}) \equiv^\text{Cxt} \nu \alpha (\Delta' \downarrow [a], \mathcal{P}') \) \\
\( \alpha \notin \text{acon}(\mathcal{V}, \mathcal{V}') \)

\( \nu \alpha (\Delta'[\mathcal{V}/a], \mathcal{P}'[\mathcal{V}/a]) \equiv^\text{Cxt} \nu \alpha (\Delta'[\mathcal{V}/a], \mathcal{P}'[\mathcal{V}/a]) \)

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Parallel Context Closure.}
\end{figure}

\section{5.3 Parallel Contexts}

Here our intention is to show that

\[ P \equiv P' \quad \text{implies} \quad P \mid Q \equiv P' \mid Q \]  

\((\ast)\)

for any \( Q \). That is \((\equiv)\) satisfies property (iii) of Definition 2.7. However, the proof requires a generalisation of \((\ast)\) to configurations.

\textbf{Definition 5.5} (Parallel Context Closure of a Relation). If \( \equiv \) is a relation on well-formed configurations of the LTS then \( \equiv^{\text{Cxt}} \) is the smallest relation on well-formed configurations satisfying the rules of Figure 6.

Here \( \text{Cxt-Par} \) is the most significant closure property. We aim to show that \((\equiv)^{\text{Cxt}}\) is contained in \((\equiv)\), from which \((\ast)\) will follow.

\textbf{Lemma 5.6.} If \( \nu \alpha (\Delta, \mathcal{P}) \equiv^{\text{Cxt}} \nu \alpha (\Delta', \mathcal{P}') \) then

\[ \text{acon}(\Delta) = \text{acon}(\Delta') \quad \text{names}(\Delta) = \text{names}(\Delta') \]

\textit{Proof.} By a straightforward induction on the rules of Figure 6. \( \square \)

\textbf{Theorem 5.7} (Parallel Context Closure of \((\equiv)\)). \((\equiv)^{\text{Cxt}} \subseteq (\equiv)\).

\textit{Proof.} Appendix A \( \square \)

\textbf{Corollary 5.8} (Parallel Context Closure of \((\equiv)\)). If \( P \equiv P' \) then for any process \( Q \), \( Q \mid P \equiv Q \mid P' \).

\textit{Proof.} By the premise and Definition 4.7, there exist \( \tilde{\alpha} \) such that \( \langle [\tilde{\alpha}], P \rangle \equiv \langle [\tilde{\alpha}], P' \rangle \).

Let \( Q \) be a process with \( \text{fn}(Q) \subseteq [\tilde{\alpha}] \). By Lemma 4.11, \( \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], P \rangle \equiv \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], P' \rangle \) and \( \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q \rangle \equiv \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q' \rangle \). By rules Cxt-R and Cxt-Par of Figure 7 \( \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q \mid P \rangle \equiv^{\text{Cxt}} \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q \mid P \rangle \), and by Theorem 5.7 \( \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q \mid P \rangle \equiv \langle [\tilde{\alpha}] \cup [\tilde{\alpha}], Q \mid P' \rangle \). Hence, by Definition 4.7, \( Q \mid P \equiv Q \mid P' \). \( \square \)
Theorem 5.9 (Soundness). \((\simeq) \subseteq (\approx_{\text{pcxt}})\).

Proof. In Propositions 5.4 and 5.3 and Corollary 5.8 we have shown that \((\simeq)\) preserves barbs, is reduction-closed, and preserves parallel contexts. Thus \((\simeq)\) is included in the largest relation with these properties, namely \((\approx_{\text{pcxt}})\). \(\Box\)

6 Completeness of Weak Bisimilarity

Here we prove that \((\approx_{\text{pcxt}})\) is included in \((\simeq)\). To do this we give a translation of LTS configurations into concrete processes.

6.1 Concretion of Configurations

We start with the definition of the translation of LTS configurations to concrete processes.

Definition 6.1 (Concretion). Let \(\nu a \langle \Delta, P \rangle\) be a well-formed configuration, and \(f\) bijection that assigns fresh names (w.r.t. names(\(\Delta\)) and \(a\)) to the abstract and concrete constants in \(\Delta\). Then the concretion of \(P, \Delta, a\) configuration are defined as follows:

\[ P/f \; \overset{\text{def}}{=} \; P\{\lambda c?0/\alpha\} \quad \text{where } f(\alpha_i) = c_i, \text{ for all } i \]

\[ \Delta/f \; \overset{\text{def}}{=} \; \prod_{\Delta(\alpha) = V/f(\alpha) = c} *(c?\text{app}V/f) \]

\[ \nu a \langle \Delta, P \rangle / f \; \overset{\text{def}}{=} \; \nu a \langle \Delta/f | P/f \rangle \]

The purpose of the concretion of a configuration is to simulate the handling of higher-order inputs and outputs in the LTS. When the abstract process applies a higher-order value that has been provided by the context the LTS simply raises a signal to the observer. The corresponding concrete process signals the observer via a communication on a unique global channel. Similarly, at any point in the execution, the LTS allows the observer to run a value that has been provided by the process (and is indexed in the environment of the configuration). The corresponding concrete process allows the same behaviour by exposing a service listening on a global channel; communication on the channel runs the value.

We now show that the reductions of translated LTS configurations are simulated by \(\tau\)-transitions of the configurations.

Lemma 6.2. If \(P/f \equiv Q\) then there exists \(Q_0\) such that \(Q \equiv Q_0/f\).

Proof. By induction on the height of the derivation tree \(P/f \equiv Q\). \(\Box\)

Lemma 6.3. If \(\Gamma, x : t \vdash P/f : OK\) and \(\nu \nu \text{app}V/f : t\) then \(\Gamma \vdash P/f[(V/f)/x] = P[V/x]/f : OK\).

Proof. By induction on the height of the derivation tree \(\Gamma, x : t \vdash P/f : OK\). \(\Box\)

Lemma 6.4. If \(\nu a \langle \Delta, P \rangle \rightarrow Q\) and \(\nu a \langle \Delta, P \rangle\) is well-formed then exactly one of the following is true:

(i) there exist \(b\) and \(Q_0\) such that

\[ Q \equiv \nu b, Q_0/f \;
\nu a \langle \Delta, P \rangle \xrightarrow{\tau} \nu b \langle \Delta, Q_0 \rangle \]

(ii) there exist \(b, Q_0,\) and \(c\) such that

\[ Q \equiv \nu b, Q_0/f | c?\emptyset \;
\nu a \langle \Delta, P \rangle \xrightarrow{\tau} \nu b \langle \Delta, Q_0 | \text{app } a \rangle \quad f(\alpha) = c \]
Proof. By induction on the height of the derivation tree $\mathcal{P}/f \rightarrow Q$, using Proposition 4.9 and Lemma 6.3.

**Proposition 6.5.** If $\nu\alpha(\Delta, \mathcal{P})/f \rightarrow Q$ then exactly one of the following is true:

(i) there exist $\bar{b}$ and $Q_0$ such that

$$Q \equiv v\bar{b}(\Delta, Q_0)/f \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0)$$

(ii) there exist $\bar{b}, Q_0, c$ such that

$$Q \equiv v\bar{b}(\Delta, Q_0)/f \mid c?\emptyset \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0 \mid \text{app } \alpha) \quad f(\alpha) = c$$

Proof. Because $\Delta/f$ can not take any steps or communicate with $\mathcal{P}/f$, it must be that for some $Q_1$,

$$\nu\alpha(\Delta, \mathcal{P})/f \rightarrow v\alpha(\Delta/f \mid Q_1) \equiv Q \quad \nu\alpha.\mathcal{P}/f \rightarrow v\alpha.\bar{Q}_1$$

and by Lemma 6.4 there exist $\bar{b}$ and $Q_0$ such that

$$\nu\alpha.\bar{Q}_1 \equiv v\bar{b}.Q_0/f \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0)$$

or there exist $\bar{b}, Q_0, c$ such that

$$\nu\alpha.\bar{Q}_1 \equiv v\bar{b}.Q_0/f \mid c?\emptyset \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0 \mid \text{app } \alpha) \quad f(\alpha) = c$$

By Proposition 3.4 (v) and (vii) we have that $[\bar{a}] \subseteq [\bar{b}]$ in both cases. Hence, either

$$Q \equiv v\bar{b}.(\Delta/f \mid Q_1) \equiv v\bar{b}(\Delta, Q_0)/f \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0)$$

or there exist $\bar{b}, Q_0, c$ such that

$$Q \equiv v\bar{b}.(\Delta/f \mid Q_1) \mid c?\emptyset \equiv v\bar{b}(\Delta, Q_0)/f \mid c?\emptyset$$

$$\nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0 \mid \text{app } \alpha) \quad f(\alpha) = c$$

From the above we conclude that reductions of translated configurations correspond to $\tau$-transitions of the original configurations, possibly accumulating several app $\alpha$ processes.

**Corollary 6.6.** If $\nu\alpha(\Delta, \mathcal{P})/f \rightarrow^* Q$ then there exist $\bar{b}, c, \alpha, Q_0$ such that

$$Q \equiv v\bar{b}(\Delta, Q_0)/f \mid \prod_{c \in [\bar{a}]} c?\emptyset \quad \nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q_0 \mid \prod_{\alpha \in [\bar{a}]} \text{app } \alpha_i) \quad f(\alpha_i) = c_i$$

Conversely, $\tau$-transitions of configurations correspond to (zero or one) reductions of their corresponding translations.

**Proposition 6.7.** If $\nu\alpha(\Delta, \mathcal{P}) \overset{\tau}{\rightarrow} v\bar{b}(\Delta, Q)$ then one of the following holds.

(i) $\nu\alpha(\Delta, \mathcal{P})/f \equiv v\bar{b}(\Delta, Q)/f$, or

(ii) $\nu\alpha(\Delta, \mathcal{P})/f \rightarrow v\bar{b}(\Delta, Q)/f$.

Proof. By rule induction.

In what follows we will use an eta-expansion lemma:

**Lemma 6.8.** $P \equiv \text{ext app } \lambda P$

Proof. By considering the smallest relation on configurations containing the identity and satisfying the axiom $P = \text{app } \lambda P$, and by showing that it is a weak bisimulation.
6.2 Completeness

The completeness proof is based on the fact that the following relation on configurations is a weak bisimulation.

**Definition 6.9** ($\Xi$).

$$\Xi \overset{\text{def}}{=} \{(\nu\alpha (\Delta, \mathcal{P}),\nu\alpha' (\Delta', \mathcal{P}')) \mid \exists f. \nu\alpha (\Delta, \mathcal{P})/f \equiv_{\text{pcxt}} \nu\alpha' (\Delta', \mathcal{P}')/f\}$$

First we show that $\Xi$ is closed under $\tau$-transitions.

**Proposition 6.10.** If $\nu\alpha (\Delta, \mathcal{P}) \not\equiv \nu\alpha' (\Delta', \mathcal{P}')$ and $\nu\alpha (\Delta, \mathcal{P}) \xrightarrow{\tau} \nu\beta (\Delta, \mathcal{Q})$ then there exist $\nu\beta$ and $\nu\beta'$ such that

$$\nu\alpha' (\Delta', \mathcal{P}') \xrightarrow{\tau} \nu\beta' (\Delta', \mathcal{Q}') \quad \nu\alpha (\Delta, \mathcal{Q}) \not\equiv \nu\alpha' (\Delta', \mathcal{Q}')$$

**Proof.** By the first premise and Definition 6.9 we get that there exists $f$ such that

$$\nu\alpha (\Delta, \mathcal{P})/f \equiv_{\text{pcxt}} \nu\alpha' (\Delta', \mathcal{P}')/f$$

(5)

By the second premise and Proposition 6.7 we get that either $\nu\alpha (\Delta, \mathcal{P})/f \equiv \nu\beta (\Delta, \mathcal{Q})/f$, or $\nu\alpha (\Delta, \mathcal{P})/f \to \nu\beta (\Delta, \mathcal{Q})/f$. In the former case the proof is completed because, by Proposition 4.9 and Theorem 5.9, $\equiv \subseteq (\approx) \subset (\equiv_{\text{pcxt}})$, hence $\nu\alpha (\Delta, \mathcal{Q})/f \equiv_{\text{pcxt}} \nu\alpha' (\Delta', \mathcal{P}')/f$ and $\nu\beta (\Delta, \mathcal{Q}) \not\equiv \nu\alpha' (\Delta', \mathcal{Q}')$. In the latter case, by (5) and Definition 2.7, we have that there exists $Q'$ such that

$$\nu\alpha' (\Delta, \mathcal{P}')/f \to^* Q'$$

(6)

$$\nu\beta (\Delta, \mathcal{Q})/f \equiv_{\text{pcxt}} Q'$$

(7)

By (6) and Corollary 6.6, there exist $\nu\beta$, $c$, $\alpha$, and $Q'_0$ such that $\nu\beta (\alpha) = c$ and

$$Q' \equiv \nu\beta (\Delta', \mathcal{Q}'_0)/f \mid \prod_{c \in \tau} c^? \cdot \theta$$

(8)

$$\nu\alpha' (\Delta', \mathcal{P}') \xrightarrow{\tau} \nu\beta' (\Delta', \mathcal{Q}'_0) \mid \prod_{\alpha \in \tau} \text{app } \alpha$$

By (7), (8), the fact that $\equiv \subseteq (\approx) \subset (\equiv_{\text{pcxt}})$, and Lemma 6.8 we have

$$\nu\beta (\Delta, \mathcal{Q})/f \equiv_{\text{pcxt}} \nu\beta (\Delta', \mathcal{Q}'_0)/f \mid \prod_{c \in \tau} c^? \cdot \theta$$

$$\equiv_{\text{pcxt}} \nu\beta (\Delta', \mathcal{Q}'_0)/f \mid \prod_{c \in \tau} \text{app } \lambda c^? \cdot \theta$$

$$= \nu\beta (\Delta', \mathcal{Q}'_0) \mid \prod_{\alpha \in \tau} \text{app } \alpha$$

thus

$$\nu\beta (\Delta, \mathcal{Q}) \not\equiv \nu\beta (\Delta', \mathcal{Q}'_0) \mid \prod_{\alpha \in \tau} \text{app } \alpha$$

$\square$
Proposition 6.11. If $\forall \alpha \langle \Delta_1, P \rangle \not\equiv \forall \alpha \langle \Delta'_1, P' \rangle$ and for some $\eta \neq \tau$, $\forall \alpha \langle \Delta_1, P \rangle \xrightarrow{\eta} \forall \alpha \langle \Delta_2, Q \rangle$ then there exist $\overline{\alpha} \langle \Delta'_1, P' \rangle$ and $\overline{\alpha} \langle \Delta'_2, Q' \rangle$ such that

$$
\forall \alpha \langle \Delta'_1, P' \rangle \xrightarrow{\eta} \forall \alpha \langle \Delta'_2, Q' \rangle \quad \forall \alpha \langle \Delta_2, Q \rangle \not\equiv \forall \alpha \langle \Delta'_2, Q' \rangle
$$

Proof. We proceed by cases on $\eta$.

Case $\eta = \text{app } \alpha$: By the second premise and Proposition 3.4 (v),

$$\Delta_1 = \Delta_2 \quad P \equiv Q \mid \text{app } \alpha \quad \overline{\alpha} = \overline{\beta}$$

Thus by the first premise and Definition 6.9 we get that there exists $f$ such that

$$\forall \alpha \langle \Delta_1, Q \mid \text{app } \alpha \rangle / f \equiv_{\text{pcxt}} \forall \alpha \langle \Delta'_1, P' \rangle / f$$

Because ($\equiv_{\text{pcxt}}$) preserves parallel contexts we pick the context $C = [\cdot] \mid c! \cdot \theta$, where $f(\alpha) = c$. We have

$$\forall \alpha \langle \Delta_1, Q \mid \text{app } \alpha \rangle / f \mid c! \cdot \theta = \forall \alpha \langle \Delta_1, Q \rangle / f \mid c! \cdot \theta \equiv_{\text{pcxt}} \forall \alpha \langle \Delta'_1, P' \rangle / f \mid c! \cdot \theta$$

Thus, by Definition 2.7 and because

$$\forall \alpha \langle \Delta_1, Q \rangle / f \mid c!. \theta \rightarrow \forall \alpha \langle \Delta_1, Q \rangle / f \quad \forall \alpha \langle \Delta_1, Q \rangle / f \nmid c! \cdot \theta$$

there must be $Q'$ such that

$$\forall \alpha \langle \Delta_1, Q \rangle / f \mid c!. \theta \rightarrow \forall \alpha \langle \Delta_1, Q \rangle / f \equiv_{\text{pcxt}} Q'$$

(9)

Therefore, there exists $Q'_1$ such that

$$\forall \alpha \langle \Delta'_1, P' \rangle / f \rightarrow^* Q'_1 \mid c?. \theta$$

$$\forall \alpha \langle \Delta'_1, P' \rangle / f \mid c!. \theta \rightarrow \forall \alpha \langle \Delta'_1, Q'_0 \rangle / f \mid c?. \theta \mid \prod_{c \in [\tau]} c?. \theta$$

(11)

and by Corollary 6.6 there exist $\overline{\alpha}, \overline{\alpha}, Q'_0$ such that $f(\alpha) = c_i$ and

$$Q'_0 \mid c?. \theta \equiv \forall \alpha \langle \Delta'_1, Q'_0 \rangle / f \mid c?. \theta \mid \prod_{c \in [\tau]} c?. \theta$$

(12)

By (10) and (11),

$$\forall \alpha \langle \Delta'_1, Q'_0 \rangle / f \mid \prod_{c \in [\tau]} c?. \theta \rightarrow^* Q'$$

and thus

$$\forall \alpha \langle \Delta'_1, Q'_0 \rangle / f \mid \prod_{c \in [\tau]} \text{app } \alpha_i \rangle / f \rightarrow^* \forall \alpha \langle \Delta'_1, Q'_0 \rangle / f \mid \prod_{c \in [\tau]} c?. \theta \rightarrow^* Q'$$

25
By Corollary 6.6 there exist $\overline{t}$, $c'$, $\overline{r}$, $Q'_2$ such that $f(\alpha'_1) = c'_1$ and

$$Q' \equiv \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / f \mid \prod_{c'_i \in \overline{r}} c'_i? \cdot \theta$$

(13)

$$\nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha_i \in \overline{r}} \text{app } \alpha_i \quad \overset{r}{\longrightarrow} \quad \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha'_i \in [\overline{r}]} \text{app } \alpha'_i$$

(14)

Hence by (12) and (14)

$$\nu \overline{t} \langle \Delta'_1, R' \rangle \overset{r}{=} \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha_i \in \overline{r}} \text{app } \alpha_i \quad \overset{app \alpha}{\longrightarrow} \quad \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha_i \in \overline{r}} \text{app } \alpha_i$$

$$\overset{r}{\longrightarrow} \quad \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha'_i \in [\overline{r}]} \text{app } \alpha'_i$$

Furthermore, by (9) and (13), the fact that $(\equiv) \subseteq (\approx) \subseteq (\approx_{\text{pcxt}})$, and Lemma 6.8 we have

$$\nu \overline{t} \langle \Delta_1, Q \rangle / f \equiv_{\text{pcxt}} \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{c'_i \in \overline{r}} c'_i? \cdot \theta$$

$$\equiv_{\text{pcxt}} \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{c'_i \in \overline{r}} \text{app } \lambda c'_i? \cdot \theta$$

$$= \quad \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha'_i \in [\overline{r}]} \text{app } \alpha'_i / f$$

Hence,

$$\nu \overline{t} \langle \Delta_1, Q \rangle \not\equiv \nu \overline{t} \langle \Delta'_1, Q'_2 \rangle / \prod_{\alpha'_i \in [\overline{r}]} \text{app } \alpha'_i$$

The rest of the cases are proved similarly using the following contexts (in which $r$ is a fresh channel):

- For $\eta = c!n$ we use the context

  $$c? (x). \text{if } x = n \text{ then } (r!. \theta | r?. \theta) \text{ else } \theta$$

  when $n \in \text{names}(\Delta_1)$, and

  $$c? (x). \text{if } x \in \text{names}(\Delta_1) \text{ then } \theta \text{ else } (r!. \theta | r?. \theta)$$

  otherwise. Here if $x \in \text{names}(\Delta_1)$ then $P \text{ else } Q$ is expressible in terms of if $x = n_i \text{ then } P \text{ else } Q$ because names$(\Delta_1)$ is a finite set of names. Moreover, $r$ is a barb of the context in parallel with the process. After the communication between the process and the context on $c$ and the communication on $r$ it is no longer a barb—in this way we “force” the communication on channel $c$.

- For $\eta = c?n$ we use the context

  $$c? (n). (r!. \theta | r?. \theta)$$

- For $\eta = c!k$ we use the context

  $$c? (X). (\ast (c_k?. \text{app } X) | r!. \theta | r?. \theta)$$

  where $c_k$ is a fresh name.
• For \( \eta = c? \alpha \) we use the context
\[
e ! (\lambda c_pr. \theta). (r! . r? . \theta)
\]
where \( c_pr \) is a fresh name.

• For \( \eta = \text{app } \kappa \) we use the context
\[
e ! \langle \lambda c ! | cfr ! > . (r! . r? | r! . r?)
\]
where \( cfr \) is a fresh name.

Proposition 6.12. \( \mathcal{X} \) is a weak bisimulation.

Proof. By Propositions 6.10 and 6.11 and by symmetry, \( \mathcal{X} \) satisfies the conditions of Definition 4.4 for a weak bisimulation. \( \square \)

Theorem 6.13 (Completeness). \((\cong_{\text{pcxt}}) \subseteq (\cong)\).

Proof. If \( P \cong_{\text{pcxt}} P' \) then for names \( n \subseteq fn(P, P') \) we have
\[
P = \langle n, P \rangle / \emptyset \cong_{\text{pcxt}} \langle n, P' \rangle / \emptyset = P'
\]
By Definition 6.9 \( \langle n, P \rangle \not\cong \langle n, P' \rangle \) and by Proposition 6.12, \( \langle n, P \rangle \equiv \langle n, P' \rangle \). Thus, by Definition 4.7, \( P \equiv P' \). \( \square \)

7 Full Barbed Congruence

In this section we study reduction-closed barbed congruence \((\equiv_{\text{cxt}})\) with arbitrary contexts. We show that weak bisimilarity implies reduction-closed barbed congruence and therefore, as with the first-order \( \pi \)-calculus, parallel contextual equivalence coincides with reduction-closed barbed congruence for pp-\( \pi \).

\[
(\cong_{\text{cxt}}) \subseteq (\equiv_{\text{pcxt}}) = (\equiv) \subseteq (\equiv_{\text{cxt}})
\]

First we give the definition for contexts.

Definition 7.1 (Contexts). A context \( C \) is derived from the following grammar:
\[
C ::= [\cdot] \mid \emptyset \mid V_C :: (V_C : t). C \mid V_C ?(x : t). C \mid | C \mid C | V_C \mid v n. C \mid \text{app } V_C \mid v(C) \mid \text{if } V_C = V_C \text{ then } C \text{ else } C
\]
\[
V_C ::= x \mid \lambda C \mid n \mid \text{bv}
\]

We write \( C[P] \) (resp. \( V_C[P] \)) to mean the replacement of all holes in \( C \) (resp. \( V_C \)) with \( P \).

Definition 7.2 (Reduction-Closed Barbed Congruence \((\equiv_{\text{cxt}})\)). \((\equiv_{\text{cxt}})\) is the largest congruence on closed processes that preserves barbs and is reduction closed; i.e. \( P \equiv_{\text{cxt}} P' \) if and only if

(i) Barb preserving: for all \( b \), \( P \uparrow b \) iff \( P' \uparrow b \).

(ii) Reduction closed: for all \( P_1 \) with \( P \rightarrow P_1 \) there exists \( P'_1 \) such that \( P' \rightarrow^* P'_1 \) and \( P_1 \equiv_{\text{cxt}} P'_1 \), and vice-versa, and

(iii) Preserves contexts: for all \( C \), \( C[P] \equiv_{\text{cxt}} C[P'] \).

The conditions of \((\equiv_{\text{cxt}})\) are stronger than those of \((\equiv_{\text{pcxt}})\), hence the following Proposition.
Proposition 7.3. \( (\approx_\text{ext}) \subseteq (\approx_\text{ctx}) \).

We will show that \( (\approx) \subseteq (\approx_\text{ext}) \). As we proved in Section 5.2, \( (\approx) \) is reduction-closed and barb-preserving. Therefore it suffices to show that \( (\approx) \) preserves contexts. For this proof we extend the definition of Parallel Context Closure (Definition 5.5) by adding the rule of Figure 7. Bisimilarity is closed under this new \( (\approx) \).

Theorem 7.4 (Context Closure of \( (\approx) \)). \( (\approx_\text{ext}) \subseteq (\approx) \).

Proof. We show that \( (\approx) \) is a weak bisimulation up to limited structural equivalence \( (\hat{\approx}) \). The proof proceeds as the one of Theorem 5.7, by induction on the rules of \( (\approx) \). The only new proof obligation is the case \( \text{Cxt-C} \) shown in Figure 7.

We proceed by induction on \( C \). The cases 0, vs. \( C \), \( c? (x: t). C \), \( c!(n: t). C \), \( c! (V_c: t). R \), \( c?(V_c: t). C \), \( + (C) \), \( \text{app} a, \text{app} i; C \), and \( 1 \text{ if } b = c \text{ then } C_1 \text{ else } C_2 \), are easy.

Let \( \Delta = \{ \kappa \mapsto V_c \} \). \( \Delta' = \{ \kappa \mapsto V_c' \} \), and \( \langle \sigma, \pi \rangle = \langle \pi, \pi' \rangle \).

Case \([\vdash]\): we need to show that if \( \nu \bar{\sigma} \langle \Delta \uplus \{ \sigma \}, \pi \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta_2, \pi' \rangle \) then there exist \( \nu \bar{\sigma} \langle \Delta_2, \pi' \rangle \) and \( \pi' \) such that

\[ \nu \bar{\sigma} \langle \Delta_2 \uplus \{ \sigma \}, \pi' \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta_2', \pi' \rangle \]

By Proposition 3.3, there exist \( \Delta_1 \) and \( \Delta_1' \) such that \( \Delta_2 = \Delta \uplus \Delta_1 \) and \( \Delta_2' = \Delta' \uplus \Delta_1' \).

We distinguish three cases for \( \eta \).

\( \bullet \) \( \eta = \text{app} \ k_i \) and \( k_i \in \Delta \): we have

\[ \nu \bar{\sigma} \langle \Delta \uplus \{ \sigma \}, \pi \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta \uplus \{ \sigma \}, \pi \mid \text{app} V_i[P] \rangle \]

\[ \nu \bar{\sigma} \langle \Delta' \uplus \{ \sigma \}, \pi \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta' \uplus \{ \sigma \}, \pi' \mid \text{app} V_i[P'] \rangle \]

\[ \nu \bar{\sigma} \langle \Delta \uplus \{ \sigma \}, \pi \mid \text{app} V_i[P] \rangle \xrightarrow{(\approx) \text{ext}} \nu \bar{\sigma} \langle \Delta' \uplus \{ \sigma \}, \pi' \mid \text{app} V_i[P'] \rangle \]

\( \bullet \) \( \eta = n_i \) or \( n_i \in \text{names}(\Delta) \): vacuously true since \( \Delta \) does not contain any names.

\( \bullet \) Otherwise: by Proposition 3.5 (ii) there exist \( \bar{\sigma} \) such that \( \bar{\sigma} \bar{\sigma} = \bar{\sigma} \bar{\sigma} \) and

\[ \langle \sigma \bar{\sigma}, \pi \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta, \pi \rangle \]

and because \( \langle \sigma, \pi \rangle \approx \langle \sigma, \pi \rangle \) there exist \( \bar{\sigma} \), \( \Delta' \), and \( \pi' \) such that

\[ \langle \sigma, \pi' \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta', \pi' \rangle \]

\[ \nu \bar{\sigma} \langle \Delta, \pi \rangle \approx \nu \bar{\sigma} \langle \Delta, \pi \rangle \]

By Proposition 3.5 (i)

\[ \nu \bar{\sigma} \langle \Delta' \uplus \{ \sigma \}, \pi' \rangle \xrightarrow{\eta} \nu \bar{\sigma} \langle \Delta' \uplus \Delta_1', \pi' \rangle \]

and because \( \nu \bar{\sigma} \langle \Delta, \pi \rangle \approx \nu \bar{\sigma} \langle \Delta', \pi \rangle \) we get

\[ \nu \bar{\sigma} \langle \Delta \uplus \Delta_1 \rangle \approx (\approx) \approx \nu \bar{\sigma} \langle \Delta' \uplus \Delta_1', \pi' \rangle \]
Case $C_1 | C_2$: by Lemma 4.11

$⟨[n, a], P⟩ ≈ ⟨[n, a], P'⟩$

By the induction hypothesis

$⟨\Delta \uplus [n, a], C_1[P]⟩ ≈ ⟨\Delta' \uplus [n, a], C_1[P']⟩$

$⟨\Delta \uplus [n, a], C_2[P]⟩ ≈ ⟨\Delta' \uplus [n, a], C_2[P']⟩$

and by Theorem 5.7

$⟨\Delta \uplus [n, a], C_1[P] | C_2[P]⟩ ≈ ⟨\Delta' \uplus [n, a], C_1[P'] | C_2[P']⟩$

and again by the same theorem

$\nu\sigma ⟨\Delta \uplus [n], C_1[P] | C_2[P]⟩ ≈ ν\sigma ⟨\Delta' \uplus [n], C_1[P'] | C_2[P']⟩ \quad □$

From the above theorem we conclude that $(\simeq)$ preserves arbitrary contexts.

Theorem 7.5 (Compositionality). If $P \simeq P'$ then for any context $C$, $C[P] \simeq C[P']$.

Theorem 7.6 (Soundness w.r.t. $(\simeq_{\text{cont}})$). $(\simeq) \subseteq (\simeq_{\text{cont}})$

Proof. In Propositions 5.4 and 5.3 and Corollary 7.5 we have shown that $(\simeq)$ preserves barbs, is reduction-closed, and preserves arbitrary contexts. Thus, $(\simeq)$ is included in the largest relation with these properties, namely $(\simeq_{\text{cont}})$.

Theorem 7.7. $(\simeq_{\text{pcxt}}) = (\simeq_{\text{cont}})$.

Proof. By the definitions of $(\simeq_{\text{cont}})$ and $(\simeq_{\text{pcxt}})$ we have $(\simeq_{\text{cont}}) \subseteq (\simeq_{\text{pcxt}})$, by Theorem 6.13 we have $(\simeq_{\text{pcxt}}) \subseteq (\simeq)$, and by Theorem 7.6 we have $(\simeq) \subseteq (\simeq_{\text{cont}})$. Hence,

$(\simeq_{\text{cont}}) \subseteq (\simeq_{\text{pcxt}}) \subseteq (\simeq) \subseteq (\simeq_{\text{cont}})$

and therefore $(\simeq_{\text{pcxt}}) = (\simeq_{\text{cont}})$.

This latter result states that the observational power of arbitrary contexts can be adequately captured by the very restricted class of parallel contexts. It also states that our proof technique is both sound and complete with respect to the touchstone behavioural equivalence $(\simeq_{\text{cont}})$.

8 Examples

Here we illustrate the effectiveness of our theory by giving simple proofs of equivalence using first-order weak bisimulation, and of inequivalence using the Hennessy-Milner Logic. Many of our examples involve ping servers and triggers, which in our opinion get to the heart of the challenges of reasoning about higher-order concurrent processes.

All equivalences can be proved using the standard weak bisimulation. However, to improve presentation, we develop a lightweight up-to $\beta$ and the limited structural equivalence $(\approx)$ technique similar to that in [3], Chapter 6. $\beta$-moves are $\tau$-transitions that are confluent with all other transitions.

Definition 8.1 ($\beta$-move ($\tau_\beta \rightarrow$)). A $\tau$-transition $C_1 \xrightarrow{\tau_\beta} C_2$ is a $\beta$-move and we write $C_1 \xrightarrow{\tau_\beta} C_2$ if and only if for all transitions $C_1 \xrightarrow{\eta} C_3$ one of the following is true:
(i) \( \eta = \tau \) and \( C_2 = C_3 \), or  
(ii) there exists \( C_4 \) such that \( C_2 \xrightarrow{\eta} C_4 \) and \( C_3 \xrightarrow{\tau} C_4 \).

**Definition 8.2** (Weak Bisimulation up-to \( \beta \) and \( \hat{=} \)). A relation \( R \) on configurations is a weak bisimulation up-to \( \beta \) and \( \hat{=} \) if and only if for all \( C \equiv C' \),

(i) if \( C \xrightarrow{\eta} C_1 \) then, for some \( C_1' \),

\[
\begin{align*}
C' & \xrightarrow{\eta} C_1' \quad C_1 \xrightarrow{\tau} \hat{=} \equiv R \equiv C_1'
\end{align*}
\]

(ii) the converse of (i)

**Proposition 8.3.** The relation \( (\tau \beta \rightarrow \ast \hat{=} \equiv R \equiv \tau \beta \leftarrow \ast) \) is transitive.

**Proof.** By induction on the rules of \( (\hat{=}) \). \( \square \)

It is easy to verify that \( (\hat{=}) \) is a weak bisimulation up-to \( \beta \) and \( (\hat{=}) \). Any weak bisimulation up-to \( \beta \) and \( (\hat{=}) \) is included in \( (\hat{=}) \):

**Proposition 8.4.** If \( R \) is a weak bisimulation up-to \( \beta \) and \( (\hat{=}) \), then \( R \subseteq (\tau \beta \rightarrow \ast \hat{=} \equiv R \equiv \tau \beta \leftarrow \ast) \subseteq (\hat{=}) \).

**Proof.** Because \( (\tau \beta \rightarrow \ast) \) and \( (\hat{=}) \) contain the identity, \( R \subseteq (\tau \beta \rightarrow \ast \hat{=} \equiv R \equiv \tau \beta \leftarrow \ast) \). Thus, it suffices to show that \( (\tau \beta \rightarrow \ast \hat{=} \equiv R \equiv \tau \beta \leftarrow \ast) \) is a weak bisimulation.

Let

\[
\begin{align*}
C_1 \xrightarrow{\tau \beta \rightarrow \ast} C_2 \equiv C_3 \equiv C_3' \equiv C_4 \xleftarrow{\tau \beta \leftarrow \ast} C_1'
\end{align*}
\]

then

\[
\begin{align*}
\text{implies} & \quad C_2 \xrightarrow{\eta} C_5 \wedge C_4 \xrightarrow{\tau \beta \rightarrow \ast} C_5 & \text{(for some} \ C_5, \ \text{by definition of} \ (\tau \beta \rightarrow \ast)) \\
\text{implies} & \quad C_3 \xrightarrow{\eta} C_6 \wedge C_5 \equiv C_6 & \text{(for some} \ C_6, \ \text{because} \ (\hat{=}) \ \text{is a strong bisimulation)} \\
\text{implies} & \quad C_3' \xrightarrow{\eta} C_6' \wedge C_6 \xrightarrow{\tau \beta \rightarrow \ast} \equiv R \equiv C_6' & \text{(for some} \ C_6', \ \text{because} \ R \ \text{is a weak bisimulation up-to} \ \beta \ \text{and} \ (\hat{=}) \) \\
\text{implies} & \quad C_3' \equiv C_5' \wedge C_6 \equiv C_5' & \text{(for some} \ C_5', \ \text{because} \ (\hat{=}) \ \text{is a strong bisimulation)} \\
\text{implies} & \quad C_1' \xrightarrow{\eta} C_5' & \text{(because} \ \tau \beta \text{-moves are} \ \tau \text{-steps)}
\end{align*}
\]

Hence, we have that for some \( C_5', C_1' \xrightarrow{\eta} C_5' \) and

\[
\begin{align*}
C_4 \xrightarrow{\tau \beta \rightarrow \ast \hat{=} \equiv R \equiv \tau \beta \leftarrow \ast} C_5'
\end{align*}
\]

and by Proposition 8.3 and the fact that \( (\tau \beta \leftarrow \ast) \) contains the identity we get

\[
\begin{align*}
C_4 \xrightarrow{\tau \beta \rightarrow \ast \hat{=} \equiv \tau \beta \leftarrow \ast} C_5'
\end{align*}
\]

Similarly we prove the converse condition of Definition 4.4. \( \square \)

Using weak bisimulation up-to \( \beta \) and \( (\hat{=}) \) we prove several interesting equivalences in the following sections.
8.1 Implementation of Replication

For our first example we consider an encoding of replication via higher-order communication. The following process receives a suspended process on channel \( p \) which then replicates and runs.

\[
\text{Rec} \overset{\text{def}}{=} p?(X).\text{va.}(R | a!((\lambda \text{app } X | R).\theta)) \\
R \overset{\text{def}}{=} a?(X).(\text{app } X | a!(X), \theta)
\]

We show that this is weakly bisimilar to

\[
\text{Rec}' \overset{\text{def}}{=} p?(X).\ast(\text{app } X)
\]

Namely, we prove that \( \text{Rec} \approx \text{Rec}' \), which by definition amounts to proving

\[
\langle \langle p \rangle, \text{Rec} \rangle \approx \langle \langle p \rangle, \text{Rec}' \rangle
\]

To prove this we will provide a relation \( R \) on configurations that relates \( \langle \langle p \rangle, \text{Rec} \rangle \) and \( \langle \langle p \rangle, \text{Rec}' \rangle \) and show that it is a bisimulation up-to (\( \hat{=} \)).

Let us first consider the configurations reachable from \( \langle \langle p \rangle, \text{Rec} \rangle \) that are relevant to our proof. These can be partitioned to the following families of configurations.

\[
C_1 \overset{\text{def}}{=} \langle \langle p \rangle, \text{Rec} \rangle \\
C_2(\alpha) \overset{\text{def}}{=} \langle \langle p, \alpha \rangle, \text{va. } R | a!((\lambda \text{app } a | R).\theta) \rangle \\
C_3(\alpha, i) \overset{\text{def}}{=} \text{va } \langle \langle p, \alpha \rangle, R | a!((\lambda \text{app } a | R).\theta) \mid [\alpha], \text{app } a \rangle \\
C_4(\alpha, i) \overset{\text{def}}{=} \text{va } \langle \langle p, \alpha \rangle, \text{app } \lambda \text{app } a | R | a!(\lambda \text{app } a | R).\theta \mid [\alpha], \text{app } a \rangle
\]

Similarly, all configurations reachable from \( \langle \langle p \rangle, \text{Rec}' \rangle \) are members of one of the two families of configurations

\[
C_1' \overset{\text{def}}{=} \langle \langle p \rangle, \text{Rec}' \rangle \\
C_2'(\alpha, i) \overset{\text{def}}{=} \langle \langle p, \alpha \rangle, \ast(\text{app } a) \mid [\alpha], \text{app } a \rangle
\]

In this and following sections we visualise the structure of each LTS involved in a bisimulation proof by a more abstract Kripke-like structure that uses families of configurations. Each node in the structure represents a family of configurations; the parameters of each family are quantified at each state. A labelled arrow between families of configurations exists if there is a configuration belonging to the originating family that has an LTS transition with the same label to a configuration in the target family. We sometimes identify transitions with the same originating and target families using metavariables.

Here the possible-worlds structure that corresponds to \( \text{Rec} \) is

\[
\begin{array}{c}
\text{app } a \\
C_1 \overset{p?\alpha}{\longrightarrow} C_2(\alpha) \overset{\tau}{\longrightarrow} C_3(\alpha, i) \overset{\tau}{\longrightarrow} C_4(\alpha, i) \\
\text{app } a
\end{array}
\]

This picture has an arrow labelled \( p?\alpha \) from \( C_1 \) to \( C_2 \) because of the LTS transition \( C_1 \overset{p?\alpha}{\longrightarrow} C_2(\alpha) \) that inputs an abstract constant \( \alpha \) on channel \( p \). The \( \tau \)-labelled arrow from \( C_2 \) to \( C_3 \) is because of the the transition \( C_2(\alpha) \overset{\tau}{\longrightarrow} C_3(\alpha, 0) \) that extrudes the private name \( a \) to the level of the configuration. The \( \tau \)-labelled arrow from \( C_3 \) to \( C_4 \) is due to the transitions \( C_3(\alpha, i) \overset{\tau}{\longrightarrow} C_4(\alpha, i) \) that communicate the value \( \lambda \text{app } a | R \) over the channel \( a \). The remaining \( \tau \)-arrow is a result of the application of \( \lambda \text{app } a | R \): \( C_4(\alpha, i) \overset{\tau}{\longrightarrow} C_3(\alpha, i + 1) \) that produces one more
process \text{app} \alpha. The self-loops on the configurations \(C_3\) and \(C_4\), labelled \text{app} \alpha, are because of the transitions that apply a process \text{app} \alpha:

\[
C_3(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C_3(\alpha, i) \quad C_4(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C_4(\alpha, i)
\]

Similarly the LTS that corresponds to \(\text{Rec}'\) can be abstracted by the following picture.

Here we have the input \(p'\alpha\) due to the transition \(C'_1 \xrightarrow{p'\alpha} C'_2(\alpha, 0)\), and the \(\tau\)-labelled loop on \(C'_2\) due to an unfolding of the replication

\[
C'_2(\alpha, i) \xrightarrow{\tau} C'_2(\alpha, i + 1)
\]

The \text{app} \alpha-labelled loop is because of applications of process \text{app} \alpha:

\[
C'_2(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C'_2(\alpha, i)
\]

We validate that the following relation is a weak bisimulation up-to \(\hat{=}\).

\[
\mathcal{R} = \{(C_1, C'_1), (C_2(\alpha), C'_2(\alpha, 0)), (C_3(\alpha, i), C'_3(\alpha, i)), (C_3(\alpha, i), C'_3(\alpha, i)) \mid \alpha, i\}
\]

Indeed, the transition \(C_1 \xrightarrow{p\alpha} C_2(\alpha)\) is matched by \(C'_1 \xrightarrow{p\alpha} C'_2(\alpha, 0)\) The \(\tau\)-transitions \(C_2(\alpha) \xrightarrow{\tau} C_2(\alpha, 0)\) and \(C_3(\alpha, i) \xrightarrow{\tau} C_3(\alpha, i)\) are matched by zero \(\tau\)-transitions from \(C'_2(\alpha, 0)\) and \(C'_3(\alpha, i)\), respectively. The transition \(C_3(\alpha, i) \xrightarrow{\tau} C_3(\alpha, i + 1)\) is matched by \(C'_3(\alpha, i) \xrightarrow{\tau} C'_2(\alpha, i + 1)\). Finally both the transitions \(C_3(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C_3(\alpha, i)\) and \(C_4(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C_4(\alpha, i)\) are matched by \(C'_3(\alpha, i + 1) \xrightarrow{\text{app} \alpha} C'_2(\alpha, i)\). All resulting configurations of matching transitions are related in \(\mathcal{R}\).

\[\text{8.2 A Trigger-Installing Ping Service}\]

We now consider a ping service that receives a suspended process on channel \text{png} and sends back on the same channel a trigger. When the context applies the trigger a copy of the suspended process is run.

\[
\text{Ping}_1 \overset{\text{def}}{=} \! \langle \nu \text{tr}. P_1(\text{tr}) \rangle \\
\text{P}_1(\text{tr}) \overset{\text{def}}{=} \text{png}?(X:\text{Pr}).\text{png}!(\lambda \text{tr}!. \emptyset). \langle \text{tr}?. \text{app} X \rangle
\]

We show that this service is weakly bisimilar to the trivial ping service

\[
\text{Ping}_2 \overset{\text{def}}{=} \langle \text{png}?(X:\text{Pr}).\text{png}!(X). \emptyset \rangle
\]

Because \((\approx)\) is a full congruence (Theorem 7.5), it suffices to show that for the processes under the replication

\[
M_1 \overset{\text{def}}{=} \nu \text{tr}. P_1(\text{tr}) \quad M_2 \overset{\text{def}}{=} \text{png}?(X:\text{Pr}).\text{png}!(X). \emptyset
\]

it is the case that \(M_1 \approx M_2\) or, by definition, \(\langle \text{png} \rangle, M_1 \rangle \approx \langle \text{png} \rangle, M_2 \rangle.\) We prove this by providing a relation \(\mathcal{R}\) that contains \(M_1\) and \(M_2\) and show that it is a weak bisimulation up-to \(\beta\) and \((\approx)\).
We identify the following families of configurations reachable from \( \langle \text{png}, M_1 \rangle \).

\[
C_1 \doteq \langle \text{png}, M_1 \rangle \\
C_2 \doteq \text{vtr} \langle \text{png}, \text{P}_1(\text{tr}) \rangle \\
C_3(\alpha) \doteq \text{vtr} \langle \langle \text{png}, \alpha \rangle, \text{png}!\langle \alpha \rangle, \text{png}!\langle \alpha \rangle, \text{png}!\langle \alpha \rangle \rangle, \langle \text{tr}? \cdot \text{app} \alpha \rangle \\
C_4(\alpha, \kappa, i, j) \doteq \text{vtr} \langle \langle \text{png}, \alpha \rightarrow \text{tr}\rangle, \alpha \rightarrow \text{tr}\rangle, \langle \text{tr}? \cdot \text{app} \alpha \rangle | \prod_i \text{tr}? \cdot \text{app} \alpha | \prod_k \text{app} \alpha \rangle
\]

An abstraction of the LTS for \( M_1 \) is the following.

\[
\begin{array}{c}
\xymatrix{ C_1 \ar[r]^\tau & C_2 \ar[r]^{\text{png}?\alpha} & C_3(\alpha) \ar[r]^{\text{png}\lambda\kappa} & C_4(\alpha, \kappa, i, j) \ar[r]^\tau & \text{app} \alpha, (\text{app} \kappa) \tau \beta \tau \beta }
\end{array}
\]

Transition \( C_1 \xrightarrow{\tau} C_2 \) is the extrusion of the local name \( \text{tr} \) to the level of the configuration.

Transition \( C_2 \xrightarrow{\text{png}?\alpha} C_3(\alpha) \) is the input on channel \( \text{png} \) of an abstract constant \( \alpha \) and transition \( C_3(\alpha) \xrightarrow{\text{png}\lambda\kappa} C_4(\alpha, \kappa, 0, 0) \) is the subsequent output of the concrete constant \( \kappa \) on the same channel. The remaining transitions are between configurations in the family \( C_4(\alpha, \kappa, i, j) \) that only affect the value of the parameters:

(i) \( C_4(\alpha, \kappa, i, j) \xrightarrow{\tau} C_4(\alpha, \kappa, i + 1, j) \) is an unfolding of the replication.
(ii) \( C_4(\alpha, \kappa, i + 1, j) \xrightarrow{\text{app}\kappa \tau \delta \tau \beta} C_4(\alpha, \kappa, i, j + 1) \) is an \( \kappa \)-transition, which puts \( \text{app} \lambda \text{tr}! \cdot \theta \) in parallel with the current process, followed by two \( \beta \)-moves. The first \( \beta \)-move is the application of \( \text{app} \lambda \text{tr}! \cdot \theta \) that produces the process \( \text{tr}? \cdot \theta \), and the second is the communication over the channel \( \text{tr} \) that releases one more process \( \text{app} \alpha \).
(iii) \( C_4(\alpha, \kappa, i, j) \xrightarrow{\text{app}\alpha} C_4(\alpha, \kappa, i, j - 1) \) is an observable application of the abstract constant \( \alpha \).

Similarly we find the families of configurations reachable from \( \langle \text{png}, M_2 \rangle \),

\[
C'_1 \doteq \langle \text{png}, M_2 \rangle \\
C'_2(\alpha) \doteq \langle \text{png}, \alpha, \text{png}!\langle \alpha \rangle, \text{png}!\langle \alpha \rangle \rangle \\
C'_3(\alpha, \kappa, j) \doteq \langle \langle \text{png}, \alpha \rightarrow \alpha \rangle, \prod_i \text{app} \alpha \rangle
\]

and the corresponding abstraction of the LTS

\[
\begin{array}{c}
\xymatrix{ C'_1 \ar[r]^{\text{png}?\alpha} & C'_2(\alpha) \ar[r]^{\text{png}\lambda\kappa} & C'_3(\alpha, \kappa, j) \ar[r] & \text{app} \kappa, \text{app} \alpha }
\end{array}
\]

Here we have no \( \tau \)-transitions, only the input and output on channel \( \text{png} \) and the transitions \( \text{app} \kappa \) and \( \text{app} \alpha \), which increase and decrease, respectively, the number of \( \text{app} \alpha \) in the process. The following relation is a weak bisimulation up-to \( \beta \) and \( \equiv \).

\[
\mathbb{R} = \{ (C_1, C'_1), (C_2, C'_2), (C_3(\alpha), C'_3(\alpha)), (C_4(\alpha, \kappa, i, j), C'_4(\alpha, \kappa, j)) | \alpha, \kappa, i, j \}
\]

The proof is straightforward. All \( \tau \)-transitions on the LHS are matched by zero transitions on the RHS; the transitions \( \text{app} \kappa \) and \( \text{app} \alpha \) on the LHS are matched with the same transitions on the RHS. Conversely, any transition on the RHS is matched with a corresponding weak transition on the LHS. Furthermore, all configurations resulting from matching moves are related by \( \mapsto^{\tau} \equiv \mathbb{R} \equiv \)
8.3 A Trigger-Promoting Ping Service

We now consider a ping service that, instead of locally installing a trigger service for each suspended process it receives, it wraps this trigger service in the response sent to the client. The trigger service is installed in only one of the clients, after an application of the response. If V is the suspended process received by the service then the response will be

$$U(V, \text{inst}, tr) \overset{\text{def}}{=} \lambda(t(r! \cdot \emptyset | \text{inst}?.\ast(t(r!.\text{app} V)))$$

where \(\text{inst}\) is a private channel controlling the installation of the trigger service and \(tr\) is a private channel that invokes it. The ping service is encoded as

$$\text{Ping}_3 \overset{\text{def}}{=} \ast(\text{vinst}.\text{vtr}. P_3(\text{inst}, tr))$$

$$P_3(\text{inst}, tr) \overset{\text{def}}{=} \text{png}?(X:\text{Pr}).\text{png}!(U(X, \text{inst}, tr)).\text{inst}!.\emptyset$$

We prove that \(\text{Ping}_3\) is weakly bisimilar to the trivial ping service \(\text{Ping}_2\), defined in the previous section. As before we will use the property of congruence for \(\equiv\) to simplify the proof. Hence we only have to prove that

$$M_3 \overset{\text{def}}{=} \text{vinst}.\text{vtr}.\text{png}?(X:\text{Pr}).\text{png}!(U(X, \text{inst}, tr)).\text{inst}!.\emptyset$$

is weakly bisimilar to \(M_2\), also defined in the previous section. We show this by providing a relation \(R\) that relates the two processes and showing that it is a weak bisimulation up-to \(\beta\) and \(\equiv\).

First, we identify the following families of configurations reachable from \((\text{png}), M_3\).

Here we omit subscripts to parallel products that do not affect the equivalence between configurations.

$$C_1 \overset{\text{def}}{=} \langle \text{png}, M_3 \rangle$$
$$C_2 \overset{\text{def}}{=} \text{vinst}.\text{tr} \langle \text{png}, P_3(\text{inst}, tr) \rangle$$
$$C_3(\alpha) \overset{\text{def}}{=} \text{vinst}.\text{tr} \langle \langle \text{png}, \alpha \rangle, \text{png}!(U(\alpha, \text{inst}, tr)).\text{inst}!.\emptyset \rangle$$
$$C_4(\alpha, \kappa) \overset{\text{def}}{=} \text{vinst}.\text{tr} \langle \langle \text{png}, \alpha \kappa \ast U(\alpha, \text{inst}, tr) \rangle, \text{inst}!.\emptyset \rangle$$
$$C_5(\alpha, \kappa, i) \overset{\text{def}}{=} \text{vinst}.\text{tr} \langle \langle \text{png}, \alpha \kappa \ast U(\alpha, \text{inst}, tr) \rangle, (\ast(t(r!.\text{app} \alpha) | \text{inst}?.\ast(t(r!.\text{app} \alpha)))) \rangle$$

The corresponding abstraction of the LTS is:

![LTS Diagram]

The first \(\tau_{\beta}\) transition extrudes the private names \(\text{inst}\) and \(tr\). The transitions \(\text{png}?.\alpha\) and \(\text{png}!.\kappa\) are the input and output of the trigger, respectively. The transition \(C_4(\alpha, \kappa) \overset{\text{app} \ast t_{\beta}}{\longrightarrow} C_5(\alpha, \kappa, i)\) is the application of the concrete constant \(\kappa\) by the context, followed by a communication on channel \(\text{inst}\) that will install the service \(\ast(t(r!.\text{app} \alpha))\), at least one unfolding of the replication, and a communication on channel \(tr\) that will produce the process \(\text{app} \alpha. C_5(\alpha, \kappa, i)\) has a \(\tau\)-loop that unfolds the replication, as well as the transitions \(C_5(\alpha, \kappa, i) \overset{\text{app} \ast t_{\beta}}{\longrightarrow} C_5(\alpha, \kappa, i + 1)\) and \(C_5(\alpha, \kappa, i + 1) \overset{\text{app} \alpha}{\longrightarrow} C_5(\alpha, \kappa, i + 1)\).

The families of configurations and the abstraction of the LTS for \(M_2\) are given in the previous section.
We can easily show that the following relation on configurations\(^1\) is a weak bisimulation up-to \(\beta\) and (\(\doteq\)).

\[ R = \{ (C_1, C_1'), (C_2, C_2'), (C_3(\alpha), C_3'(\alpha)), (C_4(\alpha, \kappa), C_4'(\alpha, \kappa, 0)), (C_5(\alpha, \kappa, i), C_5'(\alpha, \kappa, i)) \mid \alpha, \kappa, i \}\]

### 8.4 Composition of Triggers with Replication

In previous examples we used the fact that \((\approx)\) is a congruence to factor out the common contexts and simplify the proofs of equivalence. This is not always possible. To illustrate this we prove the equivalence

\[
Ping_3 \equiv Ping_4
\]

where \(Ping_3\) is the ping service defined in Section 8.3, and

\[
Ping_4 \overset{\text{def}}{=} \text{rec}(M_4) \quad M_4 \overset{\text{def}}{=} \text{png}?(X;\text{Pr}).\text{png}!(X).\emptyset
\]

\[
\text{rec}(P) \overset{\text{def}}{=} \text{va}.(R | a!(\lambda P | R).\emptyset) \quad R \overset{\text{def}}{=} a?X.(\text{app } X | a!(X).\emptyset)
\]

Because of the use of different replication constructs, there is no common context between \(Ping_3\) and \(Ping_4\) that we can factor out to reduce the proof obligation. Hence, we have to provide a relation \(R\) such that \(\langle \text{png}, Ping_3 \rangle R \langle \text{png}, Ping_4 \rangle\) and show that it is a weak bisimulation up-to \(\beta\) and (\(\doteq\)).

We devise the following family of configurations that describes the configurations that are reachable from \(\langle \text{png}, Ping_3 \rangle\) and relevant to the bisimulation proof.

\[
C(\overline{\alpha}, \overline{\kappa}, I, J, K, L, \overline{m}) \overset{\text{def}}{=} v\overline{\alpha} \langle \text{png}, \overline{\alpha}, \overline{\kappa}, \overline{\kappa} \rangle U(\alpha, \text{inst}, tr)^{K,L} \rangle, \ast(M_3)
\]

\[
\quad \mid \prod_{\alpha} P_{\text{inst}}(\text{inst}, tr)\\
\quad \mid \prod_{\alpha} \text{png}!(U(\alpha_j, \text{inst}_j, tr_j)).\text{inst}_j!\emptyset\\
\quad \mid \prod_{\alpha} \text{inst}_j!\emptyset\\
\quad \mid \prod_{\alpha} (\ast(tr_j?. \text{app } \alpha_i) \mid \prod_{\alpha} tr_j?. \text{app } \alpha_i)\\
\quad \mid \prod_{\alpha} \text{app } \alpha_i \mid \prod_{\alpha} \text{inst}_j!\ast(tr_j?. \text{app } \alpha_i))
\]

Here \(I, J, K, L\), and \(L\) are finite sets of natural numbers. We also use the notation \(\overline{A}^S\) to mean that the length of the sequence is the cardinality of the set \(S\), and the subscripts of the metavariables in \(A\) are drawn from the elements of \(S\).

The reader may observe that \(C\) contains the parallel composition of the process \(\ast(M_3)\) with an arbitrary number of the processes in configurations \(C_2, C_3, C_4, \text{ and } C_5\) of Section 8.3. This is because the ping service may be invoked multiple times by the context, and each invocation will create a separate set of states \(C_2\) to \(C_5\). The sets \(I, J, K, L\) contain the indices of local trigger channels, abstract constants, and concrete constants that correspond to the different instances of \(C_2\) to \(C_5\), respectively. Moreover, the transitions that in Section 8.3 are between configurations in \(C_i\) and \(C_j\) now only change the parameters of \(C\).

The abstraction of the LTS in this case has only one state.

\[
\tau_{\alpha} \tau_{\beta}, \\
\text{png}?!\alpha_i, \text{png}!\kappa_i, \\
(app \kappa_i)\tau_{\alpha_i}, \text{app } \alpha_i
\]

\[
C(\overline{\alpha}, \overline{\kappa}, I, J, K, L, \overline{m})
\]

\(^1\)Because of omitted indices in the definition of \(C_6\), the expression \(C_5(\overline{\alpha}, \alpha, \kappa, i)\) is a set of configurations. Here we abuse notation to mean any configuration in that set.
The configurations reachable from \(\langle\text{png}, P_{\text{ing}}\rangle\) and relevant to this bisimulation proof belong in the following families of configurations.

\[
C'_1 \triangleq \langle\text{png}, \text{rec}(M_4)\rangle
\]

\[
C'_2(\overline{\alpha}, \overline{\kappa}, I, J, K, L, \overline{m}) = \nu a \{ \langle\text{png}, \overline{\alpha} \overrightarrow{\overset{\kappa}{\alpha}} \rangle, R \mid a! (\lambda M_4 \mid R). \emptyset \\
\mid \prod M_4 \mid \prod_{j \in J} \text{png}!(\alpha_j). \emptyset \mid \prod_{i \in I} (\prod_{m_i} \text{app} \alpha_i)\}
\]

Notice that \(C'_2\) is similar to configuration \(C_1\) of Section 8.1. \(C_4\) is not necessary here because of the use of the up-to \(\beta\) technique. These are also the two states of the abstract LTS.

\[
\begin{aligned}
&\overset{\tau \beta, \text{app} \kappa, \text{app} \lambda}{\tau \beta, \text{app} \kappa, \text{app} \lambda} \\
&\overset{\tau \beta, \text{app} \kappa, \text{app} \lambda}{\tau \beta, \text{app} \kappa, \text{app} \lambda} \\
\end{aligned}
\]

8.5 The Processes in Figure 1

For our last two examples we consider the two pairs of processes in Figure 1, discussed in the introduction. We prove that the processes in \((\dagger)\) are indeed weakly bisimilar, while the processes in \((\ddagger)\) are not.

We extend the language with internal choice by adding the following reduction and transition rules (and their symmetric ones).

\[
P \oplus Q \rightarrow P \quad \nu a \langle \Delta, P \oplus Q \rangle \rightarrow \nu a \langle \Delta, P \rangle
\]

Adding these rules does not change our theory since internal choice can be encoded using communication:

\[
P \oplus Q \overset{\text{def}}{=} \nu a \mid a? . P \mid a? . Q \quad a \not\in \text{fn}(P, Q)
\]

The equivalence. First we consider the equivalence \((\dagger)\) in Figure 1. We will show that \(P = P'\); i.e. we will show \(\langle c, P \rangle \sim \langle c, P' \rangle\), where

\[
P \overset{\text{def}}{=} c? (X) . c? (Y) . \nu t . (c! (\nu V_1 (X, Y)) . \emptyset \oplus c! (\nu V_2 (X)) . \emptyset) \\
P' \overset{\text{def}}{=} c? (X) . c? (Y) . \nu t . (c! (\nu V_1 (X, Y)) . \emptyset \oplus c! (\nu V_2 (Y)) . \emptyset)
\]

\[
\nu V_1 (X, Y) \overset{\text{def}}{=} \lambda ((\lambda c (X) \oplus \lambda c (Y)) . \lambda c (X)) \\
\nu V_2 (X) \overset{\text{def}}{=} \lambda (t! . \lambda c (X))
\]

The relevant configurations reachable from \(\langle c, P \rangle\) can be described by the following families of configurations.
The following replacements of the boldfaced parts in $C_i$.

\[ C_1 \overset{\tau}{\Rightarrow} \langle \{c\}, P \rangle \]
\[ C_2(a_1) \overset{\tau}{\Rightarrow} \langle \{c, a_1\}, c^?\langle Y \rangle \rangle \text{ v.t. } (c^!\langle V_1(a_1, Y) \rangle . \theta @ c^!\langle V_2(a_1) \rangle . \theta)
\]
\[ \text{ v.t. } ((\text{app } a_1 @ \text{app } Y))
\]
\[ C_3(a_1, a_2) \overset{\tau}{\Rightarrow} \langle \{c, a_1, a_2\}, (c^!\langle V_1(a_1, a_2) \rangle . \theta @ c^!\langle V_2(a_1) \rangle . \theta)
\]
\[ \text{ v.t. } ((\text{app } a_1 @ \text{app } a_2))
\]
\[ C_4(a_1, a_2, \kappa, i, j, k)
\]
\[ \overset{\tau}{\Rightarrow} \langle \{c, a_1, a_2, \kappa \Rightarrow V_1(a_1, a_2)\}, (\text{app } a_1 @ \text{app } a_2))
\]
\[ \text{ v.t. } ((\text{app } a_1 @ \text{app } a_2))
\]
\[ C_5(a_1, a_2, \kappa, i, j, k)
\]
\[ \overset{\tau}{\Rightarrow} \langle \{c, a_1, a_2, \kappa \Rightarrow V_1(a_1)\}, (\text{app } a_1 @ \text{app } a_2))
\]
\[ \text{ v.t. } ((\text{app } a_1 @ \text{app } a_2))
\]

The corresponding abstraction of the LTS is:

\[ \tau, \text{app } \alpha_i, \text{app } \kappa
\]

We obtain the families $C_i$ to $C_5$ of configurations reachable from $\langle \{c\}, P' \rangle$ by performing the following replacements of the boldfaced parts in $C_1$ to $C_5$:

(i) in $C_1$ we replace $P$ with $P'$,
(ii) in $C_2$ we replace the $V_1(a_1, Y)$ and $V_2(a_1)$ with $V_1(Y, a_1)$ and $V_2(Y)$, respectively,
(iii) in $C_3$ to $C_5$ we replace $V_1(a_1, a_2)$ and $V_2(a_1)$ with $V_1(a_2, a_1)$ and $V_2(a_2)$, respectively.

The corresponding abstraction of the LTS is the same as the one shown above.

The intuition of this equivalence is that the $\tau$-transition

\[ C_3(a_1, a_2) \overset{\tau}{\Rightarrow} C_4(a_1, a_2, \kappa, 0, 0, 0) \]

on the left-hand side is matched by the $\tau$-transition

\[ C_3(a_1, a_2) \overset{\tau}{\Rightarrow} C_4(a_1, a_2, \kappa, 0, 0, 0) \]

on the right-hand side, and vice-versa. Hence, from that point onward, a transition

\[ C_4(a_1, a_2, \kappa, i, j, k) \overset{\text{app } \kappa}{\Rightarrow} C_4(a_1, a_2, \kappa, i + 1, j + 1, k) \]

37
is matched by the transitions

\[ C_5(\alpha_1, \alpha_2, \kappa, i, j, k) \xrightarrow{\tau_0} \tau_0 \xrightarrow{\tau_0} C_5(\alpha_1, \alpha_2, \kappa, i + 1, j + 1, k) \]

where the first \( \tau \)-step unfolds the replication once, and the second is the internal communication on channel \( \ell \).

The proof of this equivalence concludes by verifying that the following relation is a weak bisimulation up-to \( \beta \) and \( (\doteq) \).

\[ \mathbb{R} = \{ (C_1, C'_1), (C_2(\alpha_1), C'_2(\alpha_1)), (C_3(\alpha_1, \alpha_2), C'_3(\alpha_1, \alpha_2)), (C_4(\alpha_1, \alpha_i, \kappa, i, j, k), C'_4(\alpha_1, \alpha_i, \kappa, i, j, k)), [\alpha_1, \alpha_2, \kappa, i, j, k] \} \]

**The inequivalence.** We now consider the processes \( (\doteq) \) in Figure 1 and prove they are not equivalent. These processes, written in pp-\( \pi \) syntax, are:

\[
Q \overset{\text{def}}{=} c!(X).c?(Y).\langle \begin{array}{c}
\text{\(c!(\lambda((\text{app } X | \text{ app } Y) @ \text{ app } X)).\emptyset\)} \\
\text{\(\oplus c!(\lambda(!! \text{ app } Y)).\emptyset\)} \\
\text{\(| \ast (\ell?.(\text{app } X | \text{ app } Y))\)}
\end{array} \rangle
\]

\[
Q' \overset{\text{def}}{=} c!(X).c?(Y).\langle \begin{array}{c}
\text{\(c!(\lambda((\text{app } X | \text{ app } Y) @ \text{ app } Y)).\emptyset\)} \\
\text{\(\oplus c!(\lambda(!! \text{ app } X)).\emptyset\)} \\
\text{\(| \ast (\ell?.(\text{app } X | \text{ app } Y))\)}
\end{array} \rangle
\]

Let us assume that we match the output

\[ c!(\lambda((\text{app } X | \text{ app } Y) @ \text{ app } X)).\emptyset \]

on the left-hand side with the output

\[ c!(\lambda(!! \text{ app } X)).\emptyset \]

on the right-hand side. Then, after an \text{app} \( \kappa \)-transition, we could eventually arrive at related configurations where the one on the left-hand side would be able to apply the value bound to \( X \), but not the value bound to \( Y \); the configuration on the right-hand side would always be able to apply both the values bound to \( X \) and \( Y \), therefore, these configurations would not be weakly bisimilar (and would be distinguishable by an observer).

Of course there are more choices of relating configurations on the left- and right-hand side that might lead to a bisimulation. The Hennessy-Milner Logic that we used to characterise weak bisimilarity is useful in proving that none of these choices would be successful. It suffices to find an HML formula that is satisfied by the configuration \( \langle c, Q \rangle \) but not by \( \langle c, Q' \rangle \). This formula \( F \) is

\[ (c?\alpha_1)(c?\alpha_2)(c!\kappa)\text{(app } \alpha_1)(\text{app } \alpha_2) \text{ tt } \land \text{ [app } \alpha_2] \text{ ff } \]

It is the case that \( \langle c, Q \rangle \models F \), because after the inputs \( c?\alpha_1 \) and \( c?\alpha_2 \) the process can pick the output

\[ c!(\lambda((\text{app } \alpha_1 | \text{ app } \alpha_2) @ \text{ app } \alpha_1)).\emptyset \]

to perform an \text{c!}\( \kappa \)-transition. A subsequent \text{app} \( \kappa \)-transition followed by a \( \tau \)-transition of the internal choice releases the process \text{app} \( \alpha_1 \), which can perform an \text{app} \( \alpha_1 \)-transition but not an \text{app} \( \alpha_2 \)-transition.
On the other hand, \(\langle c', Q' \rangle \not\in F\) because none of the outputs
\[
c!((\text{app } a_1 \mid \text{app } a_2) \oplus \text{app } a_1)).0
\]
leads to a configuration satisfying \((\text{app } a_1)tt \land [\text{app } a_2]ff\).

9 First-Order Processes

The first-order \(\pi\)-calculus, from Chapter 1 of [14], can be considered to be a sub-language of \(\text{pp-}\pi\); let us refer to this sub-language as \(\text{fo-}\pi\) and use \(p, q\) to range over closed processes from \(\text{fo-}\pi\). The more standard theory for this sub-language is given in terms of the standard LTS in which the nodes are processes and the actions have labels of the form \(c?n\) – input, \(c!n\) – free output, \((\nu)n\) – bound output, or \(\tau\) for internal activity; note in particular the use of extrusion in the bound outputs. For these first-order (closed) processes we have the following equivalences:

(i) \(p \equiv_{\text{ext}} p'\) from Definition 7.2: intuitively this means that the first-order processes \(p\) and \(p'\) can not be distinguished by any higher-order context.

(ii) \(p = p'\) from Definition 4.7: this means that processes \(p\) and \(p'\) are weakly bisimilar when viewed as (degenerate) configurations in the LTS described in Section 3. Notice that the LTS generated by such first-order configurations only contains actions whose labels take the form \(c?n, c!n\), or \(\tau\).

(iii) \(p \equiv_{\text{fo}} p'\): meaning that \(p\) and \(p'\) are weakly bisimilar in the standard LTS alluded to above, as given in [5, 14].

For the purpose of analysis let us now introduce a fourth [5].

**Definition 9.1** (First-order p-contextual equivalence (\(\equiv_{\text{fo}}\)). \(\equiv_{\text{fo}}\) is the largest relation on closed \(\text{fo-}\pi\) processes that preserves barbs, is reduction closed, and is preserved by first-order parallel contexts.

It is known from the literature that \(\equiv_{\text{fo}}\) coincides with \(\equiv_{\text{fo}}\) ([14], Chapter 2); we show that \(\equiv\) also coincides with \(\equiv_{\text{fo}}\):

**Theorem 9.2.** In \(\text{fo-}\pi p \equiv p'\) if and only if \(p \equiv_{\text{fo}} p'\)

**Proof.** (Outline) A very easy adaptation of Theorems 5.9 and 6.13

The above theorem completes the link between contextual equivalence in \(\text{pp-}\pi\) and weak bisimilarity in the standard LTS for \(\text{fo-}\pi\) as shown in Figure 8. We can now derive the following interesting consequences:

**Corollary 9.3.** In \(\text{fo-}\pi\),

(i) \(p \equiv p'\) iff \(p \equiv_{\text{fo}} p'\)
to bound communication. For such a language there is no fully-abstract or practical translation calculi, we outline its adaptation to an extension of pp- 

As evidence for the applicability of the theory developed so far to distributed higher-order processes. This latter result has significant implications for verification; if we prove an equivalence between two first-order processes using the first-order theory, this equivalence remains true even when these first-order processes are used in a higher-order setting.

10 Extension to Distributed Localities

As evidence for the applicability of the theory developed so far to distributed higher-order calculi, we outline its adaptation to an extension of pp-π with localities that dynamically bound communication. For such a language there is no fully-abstract or practical translation to π-calculus [19].

\[ P ::= \ldots \mid [P]|\overline{u} \]

Roughly, the semantics of \([P]|\overline{u}\) is to dynamically disallow communication of \(P\) with its context over the channels in \(\overline{u}\), or bound to the variables in \(\overline{u}\). This construct is a blocking operator similar to those studied in [19, 20], where the authors argue for the usefulness of such operators for security analysis and web services, respectively.

We extend the type system to guarantee that localities are only top-level: they do not appear in suspended processes and are never nested (Figure 10). A locality that restricts no names is identified with its inner process. The communication rule now becomes

\[
[P_1 \mid c!(V,x).P_2]|\overline{m} \quad \rightarrow [P_1 \mid P_2]|\overline{m} \quad \rightarrow [Q_1 \mid Q_2]|\overline{m}
\]

provided that \(c, V \notin [\overline{m}, \overline{n}]\). The condition on \(V\) provides a way of determining whether a name \(n\) is restricted around a process. For example, if \(e_\alpha\) is fresh then the following reduction
will occur if and only if \( n \notin [\bar{n}] \).

\[
Q_1 \mid [Q_2 \mid c_p!(n).P] \bar{m} \mid c_p ?(x).r!.\emptyset \rightarrow Q_1 \mid [Q_2 \mid P] \bar{m} \mid r!.\emptyset
\]

Reductions are allowed inside localities

\[
\frac{P \rightarrow Q}{[P]\bar{m} \rightarrow [Q]\bar{n}}
\]

The structural equivalence (\( \equiv \)) is extended with the axiom:

\[
\nu n. ([P]\bar{m}) = [\nu n. P]\bar{m} \quad \text{if } n \notin [\bar{m}]
\]

### 10.1 Adaptation of the LTS

We now outline the adaptation of the LTS-based semantics of Section 3 to account for the new language constructor and the new interactions between the process and the observer. This adaptation requires the introduction of named holes in the language, written as \( \langle h \rangle \), and the following new actions of the LTS:

\[
\eta ::= \ldots | \text{app } \alpha @ h | \text{app } \kappa @ h | \xi @ h | c @ h | [h] \bar{m}
\]

where \( \xi \) ranges over communication actions. Figure 9 shows the added and modified rules of the LTS which we now explain.

With the addition of localities the transitions \text{ABS-APP-TRANS} and \text{CONC-APP-TRANS} of Figure 4 are no longer adequate and are replaced by the first two rules of Figure 9. The effect of executing a suspended process now depends on the locality in which the execution happens. For example the processes

\[
c!(x:Pr).[\text{app } x]\bar{n} \quad c!(x:Pr).\text{app } x
\]

have different behaviour and are distinguished by the observer \( c!(\lambda n!.\emptyset) \mid n?, m!. \emptyset \).

Hence, in this language, the effect of the process executing a higher-order value originally supplied by the observer (denoted by an abstract constant \( \alpha \) in the LTS) is not merely a signal to the observer. It also amounts to installing a part of the observer (i.e. an agent) at the place where the execution happens. This is encoded in our LTS by the introduction of a fresh named hole \( \langle h \rangle \) by a new \text{ABS-APP-TRANS} rule, which can further interact with the process. The label \text{app } \alpha @ h of this transition records both the name \( \alpha \) of the abstract value being executed and the name \( h \) of the generated hole. The name of the hole is also added in the knowledge environment \( \Delta \).

Each named hole can perform two kinds of transitions:
(i) **Concrete constant application**, $\text{app } \kappa @ h$: the hole $\langle h \rangle$ runs a higher-order value previously emitted from the process and indexed by $\kappa$.

(ii) **Generic communication**, $c @ h$: the hole $\langle h \rangle$ signals its ability to perform either an input or an output action on a known channel $c$. For this action, $s(c @ h) = \{c\}$ and $\text{inp}(c @ h) = \exp(c @ h) = \emptyset$.

An action $c @ h$, emitted by hole $\langle h \rangle$, can communicate with an actual input or output action $\xi$ over channel $c$, emitted by the process, via the rule $\text{Comm-Hole-Trans}$. The result is a transition $\xi @ h$ which indicates that the communication action $\xi$ was detected by the hole $\langle h \rangle$. For these actions, $s(\xi @ h) = s(\xi)$, $\text{inp}(\xi @ h) = \exp(\xi)$, and $\exp(\xi @ h) = \exp(\xi)$.

The rule $\text{New-Loc-Trans}$ allows the observer to create new localities containing a single fresh hole, in which it can later choose to execute higher-order values emitted by the process. The names that these localities restrict are either known to the observer or fresh and are added in the knowledge environment. For this action $\text{inp}(\langle h \rangle) | m = \{m\}$.

The remaining $\text{Loc-Trans}$ rule is a congruence rule which disallows localities to take input, output, or generic communication transitions over restricted channels.

Propositions 3.3, 3.6, 3.7, and 3.8 from Section 3, including located actions when allowed by the premises, are still valid. In addition, Proposition 3.7 is extended to located actions:

**Proposition 10.1** (Extrusion). Provided $c$ is different than $b$,

$$\text{vb} , \overline{\alpha} (\Delta , \mathcal{P}) \xrightarrow{c^n b \alpha h} \text{v} \overline{\alpha} (\Delta \cup \{b\}, \mathcal{Q}) \iff \text{v} \overline{\alpha} (\Delta \cup \{b\}, \mathcal{P}) \xrightarrow{c^n b \alpha h} \text{v} \overline{\alpha} (\Delta \cup \{b\}, \mathcal{Q})$$

Propositions 3.4 and 3.5 are adapted as follows.

**Proposition 10.2.** The following properties are true.

(i) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{c^n \alpha} \text{v} \overline{\beta} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1 , \mathcal{P}_2$ and $\mathcal{P}_3$

$$\mathcal{P} \equiv [c! (n) . \mathcal{P}_1 | \mathcal{P}_2] | \overline{m} | \mathcal{P}_3 \quad \mathcal{Q} \equiv [\mathcal{P}_1 | \mathcal{P}_2] | \overline{m} | \mathcal{P}_3 \quad \Delta_2 = \Delta_1 \cup \{n\} \quad [\overline{b}] = [\overline{a}] \setminus \{n\} \quad c \notin [\overline{m}]$$

(ii) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{c^n \alpha} \text{vb} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1 , \mathcal{P}_2$ and $\mathcal{P}_3$

$$\mathcal{P} \equiv [c!(\nu . \mathcal{P}_1 | \mathcal{P}_2)] | \overline{m} | \mathcal{P}_3 \quad \mathcal{Q} \equiv [\mathcal{P}_1 | \mathcal{P}_2] | \overline{m} | \mathcal{P}_3 \quad \Delta_2 = \Delta_1 \cup \nu \{k \rightarrow \nu\} \quad [\overline{b}] = [\overline{a}] \quad c \notin [\overline{m}]$$

(iii) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{c^n \alpha} \text{vb} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1 , \mathcal{P}_2$ and $\mathcal{P}_3$

$$\mathcal{P} \equiv [c!(x . \mathcal{P}_1 | \mathcal{P}_2)] | \overline{m} | \mathcal{P}_3 \quad \mathcal{Q} \equiv [\mathcal{P}_1 | \mathcal{P}_2] | \overline{m} | \mathcal{P}_3 \quad \Delta_2 = \Delta_1 \cup \{x\} \quad [\overline{b}] = [\overline{a}] \quad c \notin [\overline{m}]$$

(iv) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{c^n a} \text{vb} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1 , \mathcal{P}_2$ and $\mathcal{P}_3$

$$\mathcal{P} \equiv [c!(x. \mathcal{P}_1 | \mathcal{P}_2)] | \overline{m} | \mathcal{P}_3 \quad \mathcal{Q} \equiv [\mathcal{P}_1 | \mathcal{P}_2] | \overline{m} | \mathcal{P}_3 \quad \Delta_2 = \Delta_1 \cup \nu \{a\} \quad [\overline{b}] = [\overline{a}] \quad c \notin [\overline{m}]$$

(v) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{\text{app } \alpha @ h} \text{vb} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1$ and $\mathcal{P}_2$

$$\mathcal{P} \equiv [\text{app } \alpha | \mathcal{P}_1] | \overline{m} | \mathcal{P}_2 \quad \mathcal{Q} \equiv [\langle h \rangle | \mathcal{P}_1] | \overline{m} | \mathcal{P}_2 \quad \Delta_2 = \Delta_1 \cup \{h\} \quad [\overline{b}] = [\overline{a}]$$

(vi) If $\text{v} \overline{\alpha} (\Delta_1 , \mathcal{P}) \xrightarrow{\text{app } \kappa @ h} \text{vb} (\Delta_2 , \mathcal{Q})$ then for some $\mathcal{P}_1$ and $\mathcal{P}_2$

$$\mathcal{P} \equiv [\langle h \rangle | \mathcal{P}_1] | \overline{m} | \mathcal{P}_2 \quad \mathcal{Q} \equiv [\langle h \rangle | \text{app } \nu V | \mathcal{P}_1] | \overline{m} | \mathcal{P}_2 \quad \Delta_2 = \Delta_1 \quad [\overline{b}] = [\overline{a}]$$
(viii) If \( v\vartriangleright \langle \Delta_1, P \rangle \overset{\epsilon}{\longrightarrow} v\vartriangleright \langle \Delta_2, Q \rangle \) then \( \Delta_2 = \Delta_1 \) and \( \{\bar{a}\} \subseteq \{\bar{b}\} \).

(ix) If \( v\vartriangleright \langle \Delta_1, P \rangle \overset{\{h\}|\bar{m}}{\longrightarrow} v\vartriangleright \langle \Delta_2, Q \rangle \) then for some \( \mathcal{P}_1 \) and \( \mathcal{P}_2 \)

\[
\mathcal{P} = Q \overset{\{\langle h \rangle\}|\bar{m}}{=} \langle \mathcal{P}_1 \rangle | \bar{m} | \mathcal{P}_2 \quad \Delta_2 = \Delta_1 \quad \{\bar{b}\} = \{\bar{a}\} \quad c \notin \{\bar{m}\}
\]

(x) If \( v\vartriangleright \langle \Delta_1, P \rangle \overset{\xi \otimes h}{\longrightarrow} v\vartriangleright \langle \Delta_2, Q \rangle \) then for some \( \mathcal{P}_1, Q_1, \mathcal{P}_2 \), and \( \bar{m} \)

\[
v\vartriangleright \langle \Delta_1, P_1 \rangle \overset{\xi}{\longrightarrow} v\vartriangleright \langle \Delta_2, Q_1 \rangle
\]

and either

\[
\mathcal{P} \overset{\{\mathcal{P}_1 | \langle \rho \rangle\}|\bar{m}}{=} \langle \mathcal{P}_2 \rangle | \bar{m} | \mathcal{P}_1 \quad \mathcal{Q} \overset{\{\langle \rho \rangle\}|\bar{m}}{=} \langle \mathcal{Q}_1 \rangle | \bar{m} | \mathcal{Q}_2
\]

or

\[
\mathcal{P} \overset{\{\mathcal{P}_1 | \langle \rho \rangle\}|\bar{m}}{=} \langle \mathcal{P}_2 \rangle | \bar{m} \quad \mathcal{Q} \overset{\{\langle \rho \rangle\}|\bar{m}}{=} \langle \mathcal{Q}_1 \rangle | \bar{m} | \mathcal{Q}_2
\]

Proof: By rule induction. \( \square \)

**Proposition 10.3.**

(i) Knowledge extension: If \( v\vartriangleright \langle \Delta_1, P \rangle \overset{\eta}{\longrightarrow} v\vartriangleright \langle \Delta_2, Q \rangle \) and \( v\vartriangleright \varnothing \langle \Delta_0 \cup \Delta_1, P \rangle \) is well-formed, and \( \text{names}(\eta) \cap \text{names}(\Delta_0) = \text{acon}(\eta) \cap \text{acon}(\Delta_0) = \text{ccon}(\eta) \cap \text{ccon}(\Delta_0) = \emptyset \) then

\[
v\vartriangleright \langle \Delta_0 \cup \Delta_1, P \rangle \overset{\eta}{\longrightarrow} v\vartriangleright \varnothing \langle \Delta_0 \cup \Delta_2, Q \rangle
\]

(ii) Knowledge restriction: If \( v\vartriangleright \varnothing \langle \Delta_0 \cup \Delta_1, P \rangle \overset{\eta}{\longrightarrow} v\vartriangleright \varnothing \langle \Delta_0 \cup \Delta_2, Q \rangle \) and \( v\vartriangleright \langle \Delta_1, P \rangle \) is well-formed, and \( \eta \notin \{\text{app}\, \kappa \otimes h | \kappa \in \Delta_0\} \cup \{c?n, c\otimes h | n \in \Delta_0\} \) then

\[
v\vartriangleright \langle \Delta_1, P \rangle \overset{\eta}{\longrightarrow} v\vartriangleright \varnothing \langle \Delta_2, Q \rangle
\]

where \( \{\bar{b}'\} = \{\bar{b}\} \setminus \{\bar{c}\} \).

Proof: By rule induction. \( \square \)

**10.2 Full-abstraction**

A barb is the ability to output on a given channel. In the language extended with localities this can be defined as follows.

**Definition 10.4 (Barbs).** We write \( P \downarrow_b \) if and only if there exist \( \varnothing, V, t, P_1, P_2, P_3 \), and \( m \) with \( b \notin \{\bar{c}, \bar{m}\} \), such that \( P \equiv v\vartriangleright (\{b! (V : t) \mid \mathcal{P}_1 | \mathcal{P}_2 \} | \bar{m} | \mathcal{P}_3) \).

We write \( P \downarrow_{\bar{b}} \) if and only if there exists \( Q \) such that \( P \rightarrow^* Q \) and \( Q \downarrow_{\bar{b}} \).

We verify that the propositions of Section 4 are valid also in the extended language.
Theorem 10.7. \((\equiv) \subseteq (\approx)\).

---

**10.2.1 Soundness**

With the addition of transition Hole-IO-TRANS in the LTS, the names and holes in the environment are now observable.

**Proposition 10.5.** If \(\nu \Delta \approx \nu \Delta'\) then \(\text{names}(\Delta) = \text{names}(\Delta')\) and \(\text{holes}(\Delta) = \text{holes}(\Delta')\).

**Proof.** By contradiction, using the transition New-Loc-Trans of Figure 9 to create a new locality and Hole-IO-TRANS to observe when a name or a hole is in \(\text{names}(\Delta)\) but not in \(c \notin \text{names}(\Delta')\). \(\square\)

Similarly, the restricted names around a hole that are in the knowledge environment can be observed.

**Proposition 10.6.** If

\[
\nu \Delta \approx \nu \Delta' \quad \text{then} \quad \nu \Delta \approx \nu \Delta' \quad \text{\(\text{holes}(\Delta')\)}
\]

The changes to the LTS do not affect the behavior of \(\tau\)-transitions significantly. Thus \(\tau\)-transitions still weakly correspond to reduction steps and Lemmas 5.1 and 5.2 can be easily shown to be valid.

We define the context closure of Figure 11 and prove that bisimilarity is context closed.

Figure 11: Parallel Context Closure.
Proof. By induction on the rules of Figure 11, similarly to the proof of Theorem 5.7. □

From the above we have that bisimulation on source-level terms (=) is barb-preserving, reduction-closed, and preserves contexts. Hence it implies parallel reduction barbed congruence (≡pcxt).

Theorem 10.8. (=) ⊆ (≡pcxt).

10.3 Completeness

We prove completeness of the extended language by the same technique as that in Section 6. We update the definition of concretion of configurations as follows.

Definition 10.9 (Concretion). Let νa(Δ, P) be a well-formed configuration, and f bijection that assigns fresh names (w.r.t. names(Δ) and a) to the abstract and concrete constants, and the holes in Δ. Then the concretion of P, Δ, and a configuration are defined as follows:

\[ \mathcal{P}/f \overset{\text{def}}{=} \mathcal{P}'(λc?y.(y?(X).app X))/a,*κ(\mathcal{P}'(X)])/h) \]

where \( f(a_i) = c_i, f(h_j) = d_j \), for all \( i, j \)

\[ \Delta/f \overset{\text{def}}{=} \prod_{(\Delta, f, h) = c} *κ(app V/f).θ \]

\[ νa(Δ, P)/f \overset{\text{def}}{=} νa(Δ/f \mid P/f) \]

We prove that the following relation is a weak bisimulation.

Definition 10.10 (\( \mathcal{X} \)).

\[ \mathcal{X} \overset{\text{def}}{=} \{(\nu\Delta, P), \nu\Delta' (P', \mathcal{P}') \mid \exists f. \nu\Delta, P)/f \equivpcxt \nu\Delta', P')/f, \]

\[ \text{names}(\Delta) = \text{names}(\Delta'), \text{acon}(\Delta) = \text{acon}(\Delta'), \]

\[ \text{ccon}(\Delta) = \text{ccon}(\Delta'), \text{holes}(\Delta) = \text{holes}(\Delta') \]}

Proposition 10.11. If \( \nu\Delta, P \notin \mathcal{X} \nu\Delta' (P', \mathcal{P}') \) and \( \nu\Delta, P \overset{η}{\Rightarrow} \nu\Delta, Q \)

then there exist \( \bar{P}, \Delta_2 \), and \( Q' \) such that

\[ \nu\Delta' (P', Q') \overset{η}{\Rightarrow} \nu\bar{P}, \Delta_2, Q' \]

\[ \nu\bar{P}, \Delta_2, Q \notin \mathcal{X} \nu\bar{P}, \Delta_2, Q' \]

Proof. As the proofs of Propositions 6.10 and 6.11. For non-τ transitions we use the following contexts:

- For \( η = c!n \) and \( η = c?n \) we use the same contexts as in Proposition 6.11.
- For \( η = c!k \) we use the context

\[ c!(X).(c!p!(X).θ) \]

where \( c_p \) and \( r \) are fresh.

- For \( η = c?αr \) we use the context

\[ c!(λc_p?y.(y?(X).app X)).(r!θ | r?θ) \]

where \( c_p \) and \( r \) are fresh.
For $\eta = \text{app}\, \kappa @ h$ we use the context
\[
d!(\lambda c?!(X). (\text{app} X | r! . \theta | r?. \theta)). \theta
\]
and a concretion function $f$ with $f(\kappa) = c$ and $f(h) = d$, and fresh $r$.

For $\eta = \text{app}\, \alpha @ h$ we use the context
\[
c!(d_f). (r! . \theta | r?. \theta)
\]
and a concretion function $f$ with $f(\alpha) = c$ and fresh $d_f, r$.

For $\eta = c!n @ h$ we use the context
\[
d!(\lambda c?!(x). \text{if } x = n \text{ then } (r! . \theta | r?. \theta) \text{ else } \theta). \theta
\]
when $n \in \text{names}(\Delta_1)$, and
\[
d!(\lambda c?!(x). \text{if } x \in \text{names}(\Delta_1) \text{ then } \theta \text{ else } (r! . \theta | r?. \theta)). \theta
\]
when $n \not\in \text{names}(\Delta_1)$, and a concretion function $f$ with $f(h) = d$, and fresh $r$.

For $\eta = c?\alpha @ h$ we use the context
\[
d!(\lambda c!(n). (r! . \theta | r?. \theta)). \theta
\]
and a concretion function $f$ with $f(h) = d$, and fresh $r$.

For $\eta = c!\kappa @ h$ we use the context
\[
d!(\lambda c?!(X). c'!(X). \theta | c'?(X). (+ (c_f ! (X). \theta) | r! . \theta | r?. \theta))
\]
and a concretion function $f$ with $f(h) = d$, $f(\kappa) = c$, and fresh $r, c'$ and $c_f$.

For $\eta = c!?\alpha @ h$ we use the context
\[
d!(\lambda c!(y). (\lambda c_f?!(y). (+ (y?!(\text{app} X)) | r! . \theta | r?. \theta)). \theta)
\]
and a concretion function $f$ with $f(h) = d$, and fresh $r$ and $c_f$.

For $\eta = c!h @ h$ we use the context
\[
d!(\lambda r!c. \theta) | r?. \theta
\]
and a concretion function $f$ with $f(h) = d$, and fresh $r, r_1$.

For $\eta = [h]\Pi$, since by definition $\text{names}(\Delta_1) = \text{names}(\Delta'_1)$ and $\text{holes}(\Delta_1) = \text{names}(\Delta'_1)$, $[h]\Pi$ is a matching transition of $\nu (\Delta', \Delta'_1)^P'$.

A consequence of the above proposition is that parallel reduction barbed congruence ($\equiv_{\text{pcxt}}$) is a weak bisimulation on processes ($\Rightarrow$).

**Theorem 10.12** (Completeness). ($\equiv_{\text{pcxt}} \subseteq \Rightarrow$).

**Proof.** As the proof of Theorem 6.13. 

46
10.4  A Delay Ping Service

In the extended language the equivalences between the ping services of Sections 8.2 and 8.3 are no longer valid. The observer

\[ vt, ping!(\lambda t\. \emptyset). ping?(X). \]
\[ \{ \text{app } X | t? \cdot r_1!. \emptyset | [\text{app } X | t? \cdot r_2!. \emptyset | t \} \]

paired with Ping_1 does not signal on \( r_1 \) or \( r_2 \), paired with Ping_2 signals on both \( r_1 \) and \( r_2 \), and paired with Ping_3 signals on only one of \( r_1 \) and \( r_2 \).

Let us now consider the following implementation of a ping service:

\[ Ping_5 \overset{\text{def}}{=} (vt, P_3) \]
\[ P_3 \overset{\text{def}}{=} \text{ping?(X: Pr). ping!(\lambda t?\langle Y \rangle). app Y}.*(t!(\langle X \rangle). \emptyset) \]

The service first creates a private channel \( t \) and inputs a higher-order value on channel \( ping \). It then sends back on \( ping \) a suspended process which, when run, will receive over the channel \( t \) and execute the value originally sent to the service.

Using our theory we can prove that \( Ping_5 \) is equivalent to the trivial ping service \( Ping_2 \). The intuition of this equivalence is that the private channel \( t \) is never revealed to the observer and therefore the observer cannot restrict communication over it.

If \( M_4 \overset{\text{def}}{=} vt, P_4 \), it suffices to show that \( M_4 \equiv M_2 \). The families of configurations reachable from \( \langle ping \rangle, M_4 \) in the extended language are the following:

\[ C_1(\overline{h}, s) \overset{\Delta}{=} \langle \Delta \omega[ping], M_4 | Q(\overline{h}, s) \rangle \]
\[ C_2(\overline{h}, s) \overset{vt}{=} \langle \Delta \omega[ping], P_4 | Q(\overline{h}, s) \rangle \]
\[ C_3(\alpha, \Delta, \overline{h}, s) \overset{vt}{=} \langle \Delta \omega[ping, \alpha], ping!(\lambda t?\langle Y \rangle). app Y).*(t!(\alpha). \emptyset) | Q(\overline{h}, s) \rangle \]
\[ C_4(\alpha, \kappa, \overline{h}, s, J, K, L) \overset{vt}{=} \langle \Delta \omega[ping, \alpha, \kappa \mapsto \alpha], \overline{s} | \{ t!(\alpha). \emptyset | Q(\overline{h}, s) | \}
\[ \prod \{ \text{app } \alpha | \{ \text{app } \alpha \} | \{ \text{app } \alpha \} \} \rangle \]

where \( \overline{s} \) are sequences of names, \( Q(\overline{h}, s) = \prod \{ \langle h \rangle \} \setminus s \), and \( \Delta \) closes the free names and holes of each configuration.

Similarly, the families of configurations reachable from \( \langle ping \rangle, M_2 \) are:

\[ C'_1(\overline{h}, s) \overset{\Delta}{=} \langle \Delta \omega[ping], M_2 | Q(\overline{h}, s) \rangle \]
\[ C'_2(\alpha, \overline{h}, s) \overset{\Delta}{=} \langle \Delta \omega[ping, \alpha], ping!(\alpha). \emptyset | Q(\overline{h}, s) \rangle \]
\[ C'_3(\alpha, \kappa, \overline{h}, s, J, K, L) \overset{\Delta}{=} \langle \Delta \omega[ping, \alpha, \kappa \mapsto \alpha], Q(\overline{h}, s) | \]
\[ \prod \{ \text{app } \alpha | \{ \text{app } \alpha \} | \{ \text{app } \alpha \} \} \rangle \]

By enumerating the transitions from configurations in the above families we can verify that the following relation is a bisimulation:

\[ R = \{ (C_1(\overline{h}, s), C'_1(\overline{h}, s)), (C_2(\overline{h}, s), C'_2(\overline{h}, s)), (C_3(\alpha, \overline{h}, s), C'_3(\alpha, \overline{h}, s)), (C_4(\alpha, \kappa, \overline{h}, s, J, K, L), C'_4(\alpha, \kappa, \overline{h}, s, J, K, L)) \mid \alpha, \kappa, \overline{h}, s, J, K, L \} \]

11  Conclusions

The main achievement of this paper is a simple and effective proof technique for equivalence in a higher-order and a distributed setting. Our technique extends the standard theory of
weak bisimulations, and its corresponding Hennessy-Milner Logic to these settings. Previous
work on logics for higher-order concurrency [1, 2] aims at characterisations of higher-order
bisimilarity and relies on the use of constructive implication.

Our proof technique combines and improves existing theories, particularly those in [6]
and [13], and employs a novel treatment of extrusion. Compared to [13, 15], our bisimulations
have significantly weaker conditions, do not quantify over input contexts, and do not rely on
an up-to context or other up-to techniques to effectively reason about higher-order processes.

Compared to theories based on triggers [10, 11, 6], our use of knowledge environments
simplifies the labelled transition system by removing extrusion from its labels, thus allowing
the characterisation of weak bisimulation by a propositional Hennessy-Milner Logic. More-
over, our theory scales to languages with distribution where names can be locally scoped,
among others. For such a language there is no fully-abstract or practical translation to trig-
gers [19]. To our knowledge this is the first fully-abstract and practical theory for such a
language.

References

1995.

[2] M. Baldamus and J. Dingel. Modal characterization of weak bisimulation for higher-


LMCS, 1(1:4), 2005.

[7] A. Jeffrey and J. Rathke. Full abstraction for polymorphic pi-calculus. In FoSSaCS,
2005.

[8] V. Koutavas and M. Wand. Small bisimulations for reasoning about higher-order imperative


[10] D. Sangiorgi. Expressing mobility in process algebras: first-order and higher-order
paradigms. PhD thesis CST–99–93, Dept. of Computer Science, Univ. of Edinburgh,

[11] D. Sangiorgi. From pi-calculus to higher-order pi-calculus – and back. In TAPSOFT,
1993.


[13] D. Sangiorgi, N. Kobayashi, and E. Sumii. Environmental bisimulations for higher-


A Proof of Theorem 5.7 ((\sim)^{\text{cxt}} \subseteq (\sim))

By induction on the rules of Fig. 6.

**Case Cxt-R:** immediate.

**Case Cxt-Par:** the induction hypothesis gives

\[
\nu \bar{a} \langle \Delta_1, P_1 \rangle \approx \nu \bar{a} \langle \Delta'_1, P'_1 \rangle \quad (15)
\]
\[
\nu \bar{b} \langle \Delta_2, P_2 \rangle \approx \nu \bar{b} \langle \Delta'_2, P'_2 \rangle \quad (16)
\]

Therefore, by Lemmas 4.6 and 5.6 \(acorn(\Delta_1 \cup \Delta_2) = acorn(\Delta'_1 \cup \Delta'_2)\), \(ccorn(\Delta_1 \cup \Delta_2) = ccorn(\Delta'_1 \cup \Delta'_2)\), and \(names(\Delta_1 \cup \Delta_2) = names(\Delta'_1 \cup \Delta'_2)\). Moreover, by well-formedness of \(\nu \bar{a}, \bar{b} \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle\) and \(\nu \bar{a}, \bar{b} \langle \Delta'_1 \cup \Delta'_2, P'_1 | P'_2 \rangle\), there exist \(\Delta_{10} \subseteq \Delta_1, \Delta_{20} \subseteq \Delta_2, \Delta_{10}' \subseteq \Delta'_1,\) and \(\Delta_{20}' \subseteq \Delta'_2\) such that \(\Delta_1 \cup \Delta_2 = \Delta_{10} \cup \Delta_{20}, \Delta_1 \cup \Delta_2 = \Delta_{10}' \cup \Delta_{20}',\) and \(\Delta'_1 \cup \Delta'_2 = \Delta'_1 \cup \Delta'_2\). Here \(\Delta_{10}\) and \(\Delta_{20}\) encode the knowledge not in \(\Delta_2\) and \(\Delta_1\) respectively. It remains to show that if

\[
\nu \bar{a}, \bar{b} \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \xrightarrow{\eta} \nu \bar{c} \langle \Delta_3, Q \rangle
\]

then there exist \(\bar{c}, \Delta'_3, Q'\) such that

\[
\nu \bar{a}, \bar{b} \langle \Delta'_1 \cup \Delta'_2, P'_1 | P'_2 \rangle \xrightarrow{\eta} \nu \bar{c} \langle \Delta'_3, Q' \rangle \quad \nu \bar{c} \langle \Delta_3, Q \rangle \approx \nu \bar{c} \langle \Delta'_3, Q' \rangle
\]

and the symmetric.

Let \(\nu \bar{a}, \bar{b} \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \xrightarrow{\eta} \nu \bar{c} \langle \Delta_3, Q \rangle\). We consider the cases of this transition.

\(\blacklozenge\) **Conc-App-Trans:** we have

\[
(\Delta_1 \cup \Delta_2)(\kappa) = \nu \bar{c} \langle \Delta_3, Q \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{a} \langle \Delta_1, P_1 | \text{app } \nu \bar{c} \rangle
\]

W.l.o.g. let \(\kappa \in ccon(\Delta_1)\) and, thus, \(\kappa \in ccon(\Delta'_1)\). Then

\[
\nu \bar{a} \langle \Delta_1, P_1 \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{a} \langle \Delta_1, P_1 | \text{app } \nu \bar{c} \rangle
\]

and by (15), Definition 4.4, and Proposition 3.4 (vii), there exist \(\bar{c}, \Delta'_3, Q'_3\) such that

\[
\nu \bar{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{c} \langle \Delta'_3, Q'_3 \rangle \quad \nu \bar{a} \langle \Delta_1, P_1 | \text{app } \nu \bar{c} \rangle \approx \nu \bar{c} \langle \Delta'_3, Q'_3 \rangle
\]

Thus, by Proposition 3.5 (i),

\[
\nu \bar{a}, \bar{b} \langle \Delta'_1 \cup \Delta_{20}, P'_1 \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{c} \langle \Delta'_1 \cup \Delta_{20}, Q'_3 \rangle
\]

and by Proposition 3.6 (ii)

\[
\langle \Delta'_1 \cup \Delta_{20} | \nu [\bar{a}, \bar{b}], P'_1 \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{c} \langle \Delta'_1 \cup \Delta_{20}, \nu [\bar{a}, \bar{b}], Q'_3 \rangle
\]

hence, by rule Par-L-Trans,

\[
\nu \bar{a}, \bar{b} \langle \Delta'_1 \cup \Delta_2, P'_1 | P'_2 \rangle \xrightarrow{\text{app-\kappa}} \nu \bar{c} \langle \Delta'_1 \cup \Delta_2, Q'_1 | P'_2 \rangle
\]
and by (16), rules Cxt-R and Cxt-Par, and because \((\equiv) \subseteq (\sim ) \subseteq (\approx)\)
\[
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \cup a P \forall Q_1 (\equiv) \nu a, b \langle \Delta_1' \cup \Delta_2', P_1' | P_2' \rangle
\]

\bull \text{Par-L-Trans: using Proposition 3.3 we have}
\[
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 \rangle \rightarrow \nu \nu a, b \langle \Delta_3 \cup \Delta_2, Q_1 \rangle \\
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \rightarrow \nu \nu a, b \langle \Delta_3 \cup \Delta_2, Q_1 | P_2 \rangle
\]

We distinguish three cases for \(\eta\):
\begin{itemize}
  \item \(\eta = \text{app } \kappa \) with \(\kappa \in \text{ccom}(\Delta_2)\): for some \(V, \Delta_2(\kappa) = V, \Delta_2 = \Delta_1, Q_1 = P_1 | \text{app } V\), and \(\bar{\varepsilon} = \overline{a, b}\). Moreover,
  \[
  \nu b \langle \Delta_2, P_2 \rangle \xrightarrow{\text{app } \kappa} \nu b \langle \Delta_2, P_2 | \text{app } V\rangle
  \]

By (16), Definition 4.4, and Proposition 3.4 (vii) there exist \(\nu\) and \(Q_2\) such that
\[
\nu b \langle \Delta_2', P_2' \rangle \xrightarrow{\text{app } \kappa} \nu \nu b \langle \Delta_2', Q_2 \rangle \approx \nu \nu b \langle \Delta_2', Q_2 \rangle
\]

Thus, as before, by Propositions 3.5 (i) and 3.6 (ii), and rule Par-L-Trans,
\[
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \xrightarrow{\text{app } \kappa} \nu \nu a, b \langle \Delta_1' \cup \Delta_2', P_1' | Q_2 \rangle
\]
and by (15), rules Cxt-R and Cxt-Par, and because \((\equiv) \subseteq (\sim ) \subseteq (\approx)\)

\[
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 | \text{app } V | P_2 \rangle \approx \nu \nu b \langle \Delta_1' \cup \Delta_2', P_1' | Q_2 \rangle
\]

\bull \(\eta = c?n\) and \(n \in \text{names}(\Delta_2)\): by Proposition 3.4 (iii) we have \(\Delta_2 = \Delta_1 \cup \{n\} \cup (\Delta_2 \setminus \{n\}) = \Delta_1 \cup \Delta_2, [\bar{\varepsilon}] = \overline{a, b}\). By well-formedness of \(\nu a \langle \Delta_1, P_1 \rangle\) we get \(c \in \text{names}(\Delta_1)\) and, using Proposition 3.5 (ii),
\[
\nu a \langle \Delta_1 \cup \Delta_2, P_1 \rangle \xrightarrow{c?n} \nu \nu a \langle \Delta_1 \cup \Delta_2, P_1 \rangle
\]

By (15) and Lemma 4.11
\[
\nu a \langle \Delta_1 \cup \Delta_2, P_1 \rangle \approx \nu \nu b \langle \Delta_1' \cup \Delta_2', P_1' \rangle
\]

Thus, by Definition 4.4 and Proposition 3.4 (iii) there exist \(\nu\) and \(Q_1\) such that
\[
\nu \nu b \langle \Delta_1' \cup \{n\}, P_1' \rangle \xrightarrow{c?n} \nu \nu b \langle \Delta_1' \cup \{n\}, Q_1 \rangle \approx \nu \nu b \langle \Delta_1' \cup \{n\}, Q_1 \rangle
\]

By Lemma 5.6, \(\Delta_1' \cup \{n\} \cup (\Delta_2' \setminus \{n\}) = \Delta_1' \cup \Delta_2'.\) Thus, by Propositions 3.5 (i) and 3.6 (ii), and rule Par-L-Trans,
\[
\nu a, b \langle \Delta_1 \cup \Delta_2, P_1 | P_2 \rangle \xrightarrow{c?n} \nu \nu b \langle \Delta_1' \cup \Delta_2', Q_1 | P_2' \rangle
\]
and by (16), rules Cxt-R and Cxt-Par, and because \((\equiv) \subseteq (\sim ) \subseteq (\approx)\)

\[
\nu a, b \langle \Delta_1 \cup \Delta_2, Q_1 | P_2 \rangle \approx \nu \nu b \langle \Delta_1' \cup \Delta_2', Q_1 | P_2' \rangle
\]

\bull \(\eta \notin \{\text{app } \kappa | \kappa \in \text{ccom}(\Delta_2)\} \cup \{c?n | n \in \text{names}(\Delta_2)\}\): In this case, using the well-formedness of \(\nu a \langle \Delta_1, P_1 \rangle\) and Proposition 3.4 on the transition \(\nu b \langle \Delta_1 \cup \Delta_2, P_1 \rangle \rightarrow \nu \nu b \langle \Delta_1 \cup \Delta_2, P_1 \rangle\), we have
\[
\text{names}(\eta) \cap \text{names}(\Delta_2) = \text{acomm}(\eta) \cap \text{acomm}(\Delta_2) = \text{ccom}(\eta) \cap \text{ccom}(\Delta_2) = \emptyset
\]
and by Lemmas 4.6 and 5.6
\[ names(\eta) \cap names(\Delta'_{2n}) = acon(\eta) \cap acon(\Delta'_{2n}) = ccon(\eta) \cap ccon(\Delta'_{2n}) = \emptyset \]
Furthermore, by Proposition 3.5 (ii), for \([\overline{c}]\}_{1} = [\overline{c}]_{2}
\[ v\overline{a} \langle \Delta_1, P_1 \rangle \xrightarrow{\eta} v\overline{c}_{0} \langle \Delta_1, Q_1 \rangle \]
By (15) and Definition 4.4 there exist \(c', Q'_1\) such that
\[ v\overline{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\eta} v\overline{c} \langle \Delta'_1, Q'_1 \rangle \]
\[ v\overline{c}_{0} \langle \Delta_3, Q_1 \rangle = v\overline{c'} \langle \Delta'_3, Q'_3 \rangle \]
Thus, by Propositions 3.5 (i) and 3.6 (ii), and rule Par-L-Trans
\[ v\overline{c'}, \overline{b} \langle \Delta'_1 \cup \Delta'_2, P'_1 | P'_2 \rangle \xrightarrow{\eta} v\overline{c'}, \overline{b} \langle \Delta'_1 \cup \Delta'_2, Q'_1 | P'_2 \rangle \]
and by (16) and rules Cxt-R and Cxt-Par
\[ v\overline{c} \langle \Delta_1 \cup \Delta_2, Q_1 | P_2 \rangle \Rightarrow \text{trans} v\overline{c'}, \overline{b} \langle \Delta'_1 \cup \Delta'_2, Q'_1 | P'_2 \rangle \]
\[ \text{Comm-Name-Trans: using Proposition 3.4 (i) and (iii) we have} \]
\[ \langle \Delta_1 \cup [\overline{b}], \overline{P}_1 \rangle \xrightarrow{c_{\eta}} \langle \Delta_1 \cup [\overline{b}] \cup \Delta_2 \cup [\overline{b}], Q_1 \rangle \]
\[ \langle \Delta_1 \cup [\overline{b}], \overline{P}_1 \rangle \xrightarrow{c_{\eta}} \langle \Delta_1 \cup [\overline{b}] \cup \Delta_2 \cup [\overline{b}], Q_2 \rangle \]
\[ v\overline{a}, \overline{b} \langle \Delta_1 \cup \Delta_2, P_1 \cup P_2 \rangle \xrightarrow{r} v\overline{a}, \overline{b} \langle \Delta_1 \cup \Delta_2, Q_1 | Q_2 \rangle \]
By well-formedness of \(v\overline{a} \langle \Delta_1, P_1 \rangle\) and \(v\overline{b} \langle \Delta_2, P_2 \rangle\), and Lemma 5.6, it must be that \(c \in names(\Delta_1) \cap names(\Delta_2) = names(\Delta'_1) \cap names(\Delta'_2)\) and \(n \in names(\Delta_1) = names(\Delta'_1)\), and, using Proposition 3.5 (ii),
\[ \langle \Delta_1 \cup [\overline{b}], \overline{P}_1 \rangle \xrightarrow{c_{\eta}} \langle \Delta_1 \cup [\overline{b}] \cup \Delta_2 \cup [\overline{b}], Q_1 \rangle \]
\[ \langle \Delta_2 \cup [\overline{n}], P_2 \rangle \xrightarrow{c_{\eta}} \langle \Delta_2 \cup [\overline{n}], Q_2 \rangle \]
\[ v\overline{a} \langle \Delta_1, P_1 \rangle \xrightarrow{c_{\eta}} v\overline{c} \langle \Delta_1 \cup [\overline{n}], Q_1 \rangle \]
\[ v\overline{b} \langle \Delta_2, P_2 \rangle \xrightarrow{c_{\eta}} v\overline{b} \langle \Delta_2 \cup [\overline{n}], Q_2 \rangle \]
for \([\overline{c}] = [\overline{c}]_{1} \cup [\overline{c}]_{2}\). Therefore, by (15), (16), and Lemma 4.11, there exist \(c', c''\), \(Q'_1\), and \(Q'_2\) such that
\[ v\overline{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\tau} v\overline{c'} \langle \Delta'_1 \cup [\overline{n}], Q'_1 \rangle \]
\[ v\overline{b} \langle \Delta'_2 \cup [\overline{n}], Q'_2 \rangle \xrightarrow{\tau} v\overline{c''} \langle \Delta'_2 \cup [\overline{n}], Q'_2 \rangle \]
By Proposition 3.4 (i) and (iii), there exist \(\overline{a}, \overline{a}', \overline{b}', P'_3, P'_4, P'_5,\) and \(P'_6\) with \([\overline{a}]_2 = [\overline{a}]_{1} \cup [\overline{n}]\) such that
\[ v\overline{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\tau} v\overline{a} \langle \Delta'_1, c!(n), P'_3 | P'_4 \rangle \]
\[ v\overline{b} \langle \Delta'_2 \cup [\overline{n}], P'_5 \rangle \xrightarrow{\tau} v\overline{c'} \langle \Delta'_2 \cup [\overline{n}], Q'_1 \rangle \]
\[ v\overline{b} \langle \Delta'_2 \cup [\overline{n}], P'_5 \rangle \xrightarrow{\tau} v\overline{b} \langle \Delta'_2 \cup [\overline{n}], c!(x), P'_5 | P'_6, P'_7 \rangle \]
\[ v\overline{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\tau} v\overline{a} \langle \Delta'_1, c!(n), P'_3 | P'_4 \rangle \]
\[ v\overline{b} \langle \Delta'_2 \cup [\overline{n}], P'_5 \rangle \xrightarrow{\tau} v\overline{b} \langle \Delta'_2 \cup [\overline{n}], c!(x), P'_5 | P'_6, P'_7 \rangle \]
\[ v\overline{c'} \langle \Delta'_1 \cup [\overline{n}], Q'_1 \rangle \]
\[ v\overline{c''} \langle \Delta'_2 \cup [\overline{n}], Q'_2 \rangle \]
52
By Propositions 3.6 and 3.5 (i), and by well-formedness of $v\overline{a}, b \langle \Delta'_1 \cup \Delta'_2, P'_1 \mid P'_2 \rangle$, using the rules $\text{Par-L-Trans}$ and its symmetric, and $\text{Comm-Name-Trans}$, we get

$$v\overline{a}, b \langle \Delta'_1 \cup \Delta'_2, P'_1 \mid P'_2 \rangle \xrightarrow{\tau} v\overline{a}, b \langle \Delta'_1 \cup \Delta'_2, c!(n), P'_3 \mid P'_4 \mid \text{c?}(x).P'_5 \mid P'_6 \rangle$$

$$\xrightarrow{\tau} v\overline{a}, b \langle \Delta'_1 \cup \Delta'_2, P'_4 \mid P'_3 \mid P'_6 \rangle$$

$$\xrightarrow{\tau} v\overline{c}_3, c \langle \Delta'_1 \cup \Delta'_2, Q'_1 \mid Q'_2 \rangle$$

for some $c_3$ with $[c_3] = [c'_3] \setminus \{n\}$. Moreover, by rules $\text{Cxt-R}$ and $\text{Cxt-Par}$,

$$v\overline{a}, b \langle \{1 + \Delta_1 \cup \Delta_2\}, Q_1 \mid Q_2 \rangle (\equiv) v\overline{c}_3, c' \langle \{1 + \Delta_1 \cup \Delta_2\}, Q'_1 \mid Q'_2 \rangle$$

and we consider the following two cases:

• $n \in \text{names}(_1) = \text{names}(_1')$: We have $n \notin \overline{a}$ and $n \notin \overline{c_3}$, and thus, $[\overline{a}] = [\overline{a} \setminus \{n\}] = [\overline{a}]$ and $[\overline{c_3}] = [\overline{c_3} \setminus \{n\}] = [\overline{c_3}]$, and the above can be written as

$$v\overline{a}, b \langle \Delta_1 \cup \Delta_2, Q_1 \mid Q_2 \rangle (\equiv) v\overline{c}_3, c' \langle \Delta_1 \cup \Delta_2, Q'_1 \mid Q'_2 \rangle$$

• $n \notin \text{names}(_1) = \text{names}(_1')$: we have $[\overline{a}] = [\overline{a} \cup \{n\}]$ and $[\overline{c_3}] = [\overline{c_3} \cup \{n\}]$. Thus, by Lemma 4.12

$$v\overline{a}, b \langle \Delta_1 \cup \Delta_2, Q_1 \mid Q_2 \rangle (\equiv) v\overline{c}_3, c' \langle \Delta_1 \cup \Delta_2, Q'_1 \mid Q'_2 \rangle$$

• $\text{Comm-Proc-Trans}$: using Proposition 3.3 we have

$$\langle \Delta_1 \cup [\overline{a}], P_1 \rangle \xrightarrow{\text{cl}_{\alpha}} \langle \Delta_1 \cup [\overline{a}], P_1 \cup [\overline{a} \rightarrow V], Q_1 \rangle$$

$$\langle \Delta_1 \cup [\overline{b}], P_1 \rangle \xrightarrow{\text{cl}_{\alpha}} \langle \Delta_1 \cup [\overline{b}], P_1 \cup [\overline{a} \rightarrow V], Q_1 \rangle$$

By well-formedness of $v\overline{a} \langle \Delta_1, P_1 \rangle$ and $v\overline{b} \langle \Delta_2, P_2 \rangle$, and Lemma 5.6, it must be that $c \in \text{names}(_1) \cap \text{names}(_2) = \text{names}(_1') \cap \text{names}(_2')$. Moreover, $\alpha, \overline{a}, \overline{b} \notin \Delta_1 \cup \Delta_2 = \Delta'_1 \cup \Delta'_2$. Using Proposition 3.5 (ii),

$$\langle \Delta_1 \cup [\overline{a}], P_1 \rangle \xrightarrow{\text{cl}_{\alpha}} \langle \Delta_1 \cup [\overline{a}], P_1 \cup [\overline{a} \rightarrow V], Q_1 \rangle$$

and by Proposition 3.6 (i)

$$v\overline{a} \langle \Delta_1, P_1 \rangle \xrightarrow{\text{cl}_{\alpha}} v\overline{a} \langle \Delta_1 \cup [\overline{a} \rightarrow V], Q_1 \rangle$$

Therefore, by (15), (16), Definition 4.4, and Proposition 3.4 there exist $c_1, c_2, V'$, $Q'_1$, and $Q'_2$ such that

$$v\overline{a} \langle \Delta'_1, P'_1 \rangle \xrightarrow{\text{cl}_{\alpha}} v\overline{c}_1 \langle \Delta'_1 \cup [\overline{a} \rightarrow V'], Q'_1 \rangle$$

$$v\overline{a} \langle \Delta_1 \cup [\overline{a} \rightarrow V], Q_1 \rangle \approx v\overline{c}_1 \langle \Delta'_1 \cup [\overline{a} \rightarrow V'], Q'_1 \rangle$$

$$v\overline{b} \langle \Delta'_2, P'_2 \rangle \xrightarrow{\text{cl}_{\alpha}} v\overline{c}_2 \langle \Delta'_2 \cup [\overline{a}], Q'_2 \rangle$$

$$v\overline{b} \langle \Delta_2 \cup [\overline{a}], Q_2 \rangle \approx v\overline{c}_2 \langle \Delta'_2 \cup [\overline{a}], Q'_2 \rangle$$

53
Moreover, there exist $\bar{a}, \bar{a}, \bar{b}, \bar{P}_3, \bar{P}_4, \bar{P}_5$, and $\bar{P}_6$, with $|\bar{a}| = |\bar{a}| \setminus \{n\}$ such that

$$\nu\bar{a} \langle \Delta_1, \bar{P}_1 \rangle \overset{\text{ck}}{\Rightarrow} \nu\bar{a} \langle \Delta'_1, c!(V'), \bar{P}_3 | \bar{P}_4 \rangle$$
$$\nu\bar{a} \langle \Delta'_1 \cup \{\alpha\} \circ \Rightarrow \nu\bar{a} \langle \Delta'_1 \cup \{\alpha\} \circ, \bar{P}_3 | \bar{P}_4 \rangle$$

$$\nu\bar{b} \langle \Delta_2, \bar{P}_2 \rangle \overset{\text{ck}}{\Rightarrow} \nu\bar{b} \langle \Delta'_2, c!(x), \bar{P}_5 \rangle$$
$$\nu\bar{b} \langle \Delta'_2 \cup \{\alpha\}, \bar{P}_3 | \bar{P}_4 \rangle$$

By Propositions 3.5 (i) and 3.6 (ii), and because $\nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, \bar{P}_1 | \bar{P}_2 \rangle$ is well-formed, using the rule Par-L-Trans and its symmetric,

$$\nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, \bar{P}_1 | \bar{P}_2 \rangle \overset{\tau}{\Rightarrow} \nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, (c!(V'), \bar{P}_3) | \bar{P}_4 \rangle$$

by rule Comm-Proc-Trans

$$\nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, \bar{P}_1 | \bar{P}_2 \rangle \overset{\tau}{\Rightarrow} \nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, (c!(V'), \bar{P}_3) | \bar{P}_4 \rangle$$

and by applying Propositions 3.5 (i) and 3.6 (ii), rule Par-L-Trans and its symmetric, Proposition 3.8 (i), and Proposition 3.5 (ii)

$$\nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, \bar{Q}_1 \rangle \overset{\tau}{\Rightarrow} \nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, \bar{Q}_1 \rangle$$

Moreover, by rules Cxt-R and Cxt-Par,

$$\nu\bar{a}, \nu\bar{b} \langle \Delta \cup \Delta_2, Q_1 | Q_2 \rangle \overset{(\nu)\text{ext}}{\Rightarrow} \nu\bar{a}, \nu\bar{b} \langle \Delta_1 \cup \Delta_2, Q_1 | Q_2 \rangle$$

and because $\alpha \notin \Delta_1 \cup \Delta_2$ (thus $\alpha \notin ac\text{(V',V')}$), by rule Cxt-Subst,

$$\nu\bar{a}, \nu\bar{b} \langle \Delta_1 \cup \Delta_2, Q_1 | Q_2(V'/\alpha) \rangle \overset{(\nu)\text{ext}}{\Rightarrow} \nu\bar{a}, \nu\bar{b} \langle \Delta'_1 \cup \Delta'_2, Q_1 | Q_2(V'/\alpha) \rangle$$

**Case Cxt-Hide:** By the induction hypothesis

$$\nu\bar{a} \langle \Delta \cup \{\alpha\}, \bar{P} \rangle \approx \nu\bar{a} \langle \Delta' \cup \{\alpha\}, \bar{P}' \rangle$$

and by Lemma 4.12

$$\nu\bar{a}, n \langle \Delta, \bar{P} \rangle \approx \nu\bar{a}, n \langle \Delta', \bar{P}' \rangle$$

**Case Cxt-Subst:** By the induction hypothesis

$$\nu\bar{a} \langle \Delta \cup \{\alpha, \kappa \rightarrow V\}, \bar{P} \rangle \approx \nu\bar{a} \langle \Delta' \cup \{\alpha, \kappa \rightarrow V'\}, \bar{P}' \rangle$$

(17)

It remains to show that if

$$\nu\bar{a} \langle \Delta(V/\alpha), \bar{P}(V/\alpha) \rangle \overset{\eta}{\Rightarrow} \nu\bar{b} \langle \Delta_1(V/\alpha), Q'(V/\alpha) \rangle$$

then there exist $\bar{b}', \Delta'_1, Q'$ such that

$$\nu\bar{a} \langle \Delta'(V'/\alpha), \bar{P}'(V'/\alpha) \rangle \overset{\eta}{\Rightarrow} \nu\bar{b} \langle \Delta'_1(V'/\alpha), Q'(V'/\alpha) \rangle$$

$$\nu\bar{a} \langle \Delta_1(V'/\alpha), Q'(V'/\alpha) \rangle \overset{(\nu)\text{ext}}{\Rightarrow} \nu\bar{a} \langle \Delta'_1(V'/\alpha), Q'(V'/\alpha) \rangle$$
Proving the symmetric is analogous.

Let

\[ \nu \bar{a} (\Delta' [V/\alpha], \mathcal{P}[V/\alpha]) \xrightarrow{\eta} \nu \bar{b} (\Delta_1 [V/\alpha], Q[V/\alpha]) \]

We distinguish three cases:

1. \( \eta \) is a \( \tau \)-action involving the rule \textit{App-Trans}: It is easy to see that such an action is possible by applying rule \textit{Par-L-Trans} (and its symmetric) an arbitrary number of times and then rule \textit{App-Trans}. Hence,

\[ \mathcal{P}[V/\alpha] \triangleq \text{app } \lambda \mathcal{P}_1 | \mathcal{P}_2[V/\alpha] \]

Moreover, by Proposition 3.4 (vii), \( \Delta_1 = \Delta, \bar{a} = \bar{b} \). By the properties of substitution, either \( \mathcal{P} \triangleq \text{app } \lambda \mathcal{P}_0 | \mathcal{P}_2 \) and \( \mathcal{P}_1 = \mathcal{P}_0[V/\alpha] \), or \( \mathcal{P} \triangleq \text{app } \alpha | \mathcal{P}_1 \) and \( V = \lambda \mathcal{P}_1 \).

- \( \mathcal{P} \triangleq \text{app } \lambda \mathcal{P}_0 | \mathcal{P}_2 \) and \( \mathcal{P}_1 = \mathcal{P}_0[V/\alpha] \): We have \( Q = \mathcal{P}_0 | \mathcal{P}_2 \) and

\[ \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], \mathcal{P}) \xrightarrow{\tau} \nu \bar{b} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q) \]

By (17), Definition 4.4, and Proposition 3.4 (vii), there exist \( \bar{b}', Q' \) such that

\[ \nu \bar{a} (\Delta' \uplus [\alpha, \kappa \rightarrow V'], \mathcal{P}') \xrightarrow{\tau} \nu \bar{b}' (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q') \]

\[ \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q) \Rightarrow \nu \bar{b}' (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q') \]

By applying Propositions 3.8 (i) and 3.5 (ii)

\[ \nu \bar{a} (\Delta' [V'/\alpha], \mathcal{P}'[V'/\alpha]) \xrightarrow{\tau} \nu \bar{b}' (\Delta' [V'/\alpha], Q'[V'/\alpha]) \]

and by rule Cxt-Subst

\[ \nu \bar{a} (\Delta [V'/\alpha], Q[V'/\alpha]) \Rightarrow \nu \bar{b}' (\Delta' [V'/\alpha], Q'[V'/\alpha]) \]

- \( \mathcal{P} \triangleq \text{app } \alpha | \mathcal{P}_2 \) and \( V = \lambda \mathcal{P}_1 \): We have \( Q \triangleq \mathcal{P}_1 | \mathcal{P}_2 \), and for some \( Q_1 \triangleq \mathcal{P}_2 \) and \( Q_2 \triangleq \text{app } V | Q \)

\[ \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], \mathcal{P}) \xrightarrow{\text{app } \alpha} \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q_1) \]

\[ \xrightarrow{\text{app } \kappa} \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q_2) \]

\[ \xrightarrow{\tau} \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q) \]

By (17), Definition 4.4, and Proposition 3.4, there exist \( \bar{a}', \bar{b}', Q_0', Q', Q^1, Q^2, Q', Q_0' \), and \( Q_1' \) such that \( Q_1' \triangleq \text{app } \alpha | Q_2', Q_0' \triangleq \text{app } V | Q_1' \), and

\[ \nu \bar{a} (\Delta' \uplus [\alpha, \kappa \rightarrow V'], \mathcal{P}') \xrightarrow{\tau} \nu \bar{a}_1 (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q_1') \]

\[ \xrightarrow{\text{app } \alpha} \nu \bar{a}_1 (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q_2') \]

\[ \xrightarrow{\tau} \nu \bar{a}_1 (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q') \]

\[ \nu \bar{a} (\Delta \uplus [\alpha, \kappa \rightarrow V], Q) \Rightarrow \nu \bar{b}' (\Delta' \uplus [\alpha, \kappa \rightarrow V'], Q') \]

55
It is easy to show that there also exist $\overline{\alpha}'$, $Q_i$, $Q_i'$, and $Q_i''$ such that $Q_i'' \equiv \text{app } \alpha | Q_i'$, $Q_i \equiv \text{app } \alpha | Q_i'$, and 

\[
\forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P}' \rangle \overset{r}{\rightarrow} \forall \overline{\alpha}_2 \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q_i' \rangle
\]

By applying Propositions 3.8 (i) and 3.5 (ii), and rule App-Trans

\[
\forall \overline{\alpha}_2 \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q_i' \rangle \overset{r}{\rightarrow} v \overline{\beta} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q_i' \rangle
\]

and by rule Cxt-Subst

\[
\forall \alpha \equiv \text{app } \alpha_1: \text{Because } \alpha \text{ is substituted by } \nu' \text{ and } \alpha \not\in \text{acon}(\nu', \nu''), \text{ it must be } \alpha_1 \neq \alpha; \text{by Proposition } 3.4 \text{ (v)}
\]

\[
\mathcal{P}[\nu'/\alpha] \equiv \text{app } \alpha_1 | Q[\nu'/\alpha]
\]

Then either $\mathcal{P} \equiv \text{app } \alpha_1 | Q$ or $\mathcal{P} \equiv \text{app } \alpha | Q$ and $\nu' = \alpha_1$.

- $\mathcal{P} \equiv \text{app } \alpha_1 | Q$: We have 

\[
\forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P} \rangle \overset{\text{app-}}{\rightarrow} \forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, Q \rangle
\]

By (17), Definition 4.4, and Proposition 3.4, there exist $\overline{\beta}$ and $Q$ such that

\[
\forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P}' \rangle \overset{r}{\rightarrow} \forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]

\[
\forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, Q \rangle \approx v \overline{\beta} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]

By Propositions 3.8 (i) and 3.5 (ii), and by rule Cxt-Subst

\[
\forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P}' \rangle \overset{\text{app-}}{\rightarrow} \forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]

\[
\forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, Q \rangle \approx v \overline{\beta} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]

- $\mathcal{P} \equiv \text{app } \alpha | Q$ and $\nu' = \alpha_1$: We have 

\[
\forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P} \rangle \overset{\text{app-}}{\rightarrow} \forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, Q \rangle
\]

By (17), Definition 4.4, and Proposition 3.4, there exist $\overline{\alpha}_1'$, $\overline{\beta}'$, $\mathcal{P}_1'$, $\mathcal{P}_2'$, and $Q'$ such that $\mathcal{P}_1' \equiv \text{app } \alpha | \mathcal{P}_2'$ and

\[
\forall \overline{\alpha} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, \mathcal{P}' \rangle \overset{r}{\rightarrow} \forall \overline{\alpha}_2 \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]

\[
\forall \overline{\alpha} \langle \Delta \cup \{\alpha, \kappa \rightarrow \nu' \}, Q \rangle \approx v \overline{\beta} \langle \Delta' \cup \{\alpha, \kappa \rightarrow \nu' \}, Q' \rangle
\]
By Propositions 3.8 (i) and 3.5 (ii), and by rule Abs-App-Trans

\[
\nu a′\langle \Delta′[\nu′/a′], P′[\nu′/a′]\rangle \xrightarrow{\text{app}} \nu a′\langle \Delta′[\nu′/a′], P′_1[\nu′/a′]\rangle \\
\xrightarrow{\text{app}} \nu a′\langle \Delta′[\nu′/a′], Q′[\nu′/a′]\rangle
\]

and by rule Cxt-Subst

\[
\nu a\langle \Delta[\nu/\alpha], Q[\nu/\alpha]\rangle \xrightarrow{\text{ext}} \nu b\langle \Delta′[\nu′/\alpha], Q′[\nu′/\alpha]\rangle
\]

♦ Otherwise: By Propositions 3.8 (ii) and 3.5 (i), and because by definition of ( )^ext for the case where \( \eta = c? a_1 \) or \( \eta = c!\alpha \) it must be \( \alpha_1 \neq \alpha \) or \( \kappa_1 \neq \kappa \), respectively,

\[
\nu a\langle \Delta \cup [\alpha, \kappa \mapsto \nu V], P \rangle \xrightarrow{\eta} \nu b\langle \Delta_2 \cup [\alpha, \kappa \mapsto \nu V], Q \rangle
\]

By (17) and Definition 4.4, there exist \( \nu b, \Delta′_2 Q′ \) such that

\[
\nu a\langle \Delta_1 \cup [\alpha, \kappa \mapsto \nu V], P′ \rangle \xrightarrow{\eta} \nu b\langle \Delta_2 \cup [\alpha, \kappa \mapsto \nu V], Q′ \rangle
\]

By Propositions 3.8 (i) and 3.5 (ii), and rule Cxt-Subst

\[
\nu a\langle \Delta_1[\nu′/\alpha], P′[\nu′/\alpha]\rangle \xrightarrow{\text{ext}} \nu b\langle \Delta_2′[\nu′/\alpha], Q′[\nu′/\alpha]\rangle \\
\nu b\langle \Delta_2[\nu/\alpha], Q[\nu/\alpha]\rangle \xrightarrow{\text{ext}} \nu b\langle \Delta_2′[\nu′/\alpha], Q′[\nu′/\alpha]\rangle
\]