Weighted barbed congruence

Iwan Quémerais
ENS de Lyon
Lyon, France
Università di Bologna
Bologna, Italy
iwan.quemerais@ens-lyon.fr

In concurrent languages, a common way of defining the reference behavioural equivalence is barbed congruence, as the context closure of barbed bisimilarity. In barbed bisimilarity, a bisimulation game is played on reductions and the only observables are the barbs indicating the possibility of emitting an output at a given name. To reason about barbed congruence one however looks for direct characterisations, without quantification over contexts. In common process calculi such as plain CCS or π-calculus, or their asynchronous variants, these characterisations yield the familiar labeled bisimilarity.

In calculi where processes have finite size, when abstracting from internal reductions, and therefore adopting weak forms of behavioural equivalence, these characterizations of barbed congruence as bisimilarity are proved on restricted classes of processes, usually image-finite processes. The question of whether barbed congruence coincides with labeled bisimilarity has remained long open, and then solved by Fournet and Gonthier on asynchronous calculi. Their proof is clever but also long and complex; indeed up to now it has not been replicated in other calculi, as far as we know.

This paper proposes the use of *weighted barbs* in place of ordinary barbs. This refinement allows for simpler comparisons with labeled bisimilarities while resulting in the same barbed congruence. We consider four such refinements, for asynchronous and synchronous process calculi, in CCS and π-calculus.

1 Introduction

In programming language semantics, the reference behavioural equivalence is usually contextually defined, exploiting a notion of success. In a sequential setting, the success predicate is usually inductive, yielding the so-called Morris-style contextual equivalence. In a concurrent setting, success is often defined coinductively using (the context closure of) barbed bisimilarity, called *barbed congruence* [6, 5]. Contextually-defined relations are hard to handle because of the heavy quantification over all contexts. Hence one looks for direct characterizations, using labelled transition systems. For Morris-style contextual equivalence, one usually thus obtain trace equivalence. For barbed congruence one usually obtains ordinary (strong) bisimilarity.

For concrete applications, it is important to abstract from internal process reductions, and therefore adopting weak forms of barbed congruence and bisimilarity. In π-calculus, the usual method to prove characterization of barbed congruence as bisimilarity is to use a stratification of the bisimilarity and show that each layer includes barbed congruence [4, 8]. However, bisimilarity only coincides with the intersection of its stratifications on a restricted class of processes, such as the image-finite processes. It is not a limitation for strong bisimilarity as all processes of finite size are image-finite under strong transitions, this is not the case for weak transitions, hence with this method the characterization of barbed congruence as weak bisimilarity can only be shown on image-finite processes.

Submitted to:
SOS 2007

In paradigmatical process calculi such as π-calculus and asynchronous π-calculus [8], without infinitary features such as infinite sums [7] or infinite recursive definitions (e.g. [1]), the question of whether, in the weak case, barbed congruence coincides with labeled bisimilarity on the whole language has been long open. The question was settled by Fournet and Gonthier [2] for asynchronous calculi. However, their proof, though very clever, is long and complex; indeed it has not been replicated for any other calculi, as far as we know. Moreover, their proof relies on an encoding of the integers, therefore the proof cannot directly apply to languages that do not have such expressive power like asynchronous CCS [4].

In this paper, we propose an alternative to the usual barbs, namely *weighted barbs*. In the usual definition, a process P has barb at a, written $P \Downarrow_a$, when P is capable of emitting an output at a, i.e., for asynchronous CCS we have $P \Rightarrow \equiv \overline{a} \mid Q$, for some Q, where $\equiv$ designates the structural congruence. Weighted barbs allow us to test the multiplicity of the barbs, for example $P \Downarrow_a^3$ holds if $P \Rightarrow \equiv \overline{a} \mid \overline{a} \mid \overline{a} \mid Q$ for some Q. In other words, we view barbs as a multiset rather than a plain set. We show that the modification of the notion of barb does not affect the resulting barbed congruence. However, we show that the additional observational power can be used to obtain an easier and more robust method to prove coincidence with labeled bisimilarity.

Observing the multiplicity of actions in concurrency usually brings into the realm of true-concurrency semantics; however, in an asynchronous setting in which output particles have no continuations, observing barb multisets does not conflict with the interleaving semantics. In synchronous calculi, in contrast, we cannot use the above definition for weighted barbs. For example it would distinguish the terms $\overline{a} \mid \overline{a}$ and $\overline{a}.\overline{a}$, which are bisimilar. For these calculi, we propose a different refinement of barbs: we allow communication of natural numbers along the names used to produce barbs, and we observe such natural numbers.

In order to prove characterizations of barbed congruence as bisimilarity we will build universal distinguishing contexts, meaning contexts that can simulate any bisimulation game for a certain class of processes; we then show that if two processes are barbed bisimilar under a certain universal distinguishing context then they are bisimilar. Using the weighted barbs, these universal distinguishing contexts will be able to track the depth at which the corresponding bisimulation game is played (and thus force synchronization between the two processes during the barbed-bisimulation game) as well as which actions each context is allowing in the bisimulation game.

We showcase our proof method on different process calculi, both asynchronous and synchronous. We start by the simple language of *asynchronous CCS* where a very concise process is able to play the role of universal distinguishing observer. Then we show how the method is adapted to synchronous calculi by considering *synchronous CCS with value-passing*. Then we show how the method can handle the creation of new names during the execution by considering the *asynchronous π-calculus*. And finally we show how both adaptations can be combined by considering the *synchronous π-calculus*.

1.1 Preliminary notions

In this paper we consider different process calculi. Processes will be ranged over by $P, Q, R, \ldots$; these processes manipulate names (or channel names) in the countable set $\mathtt{Names} = \{a, b, \ldots, f, g, \ldots, x, y, \ldots, p, q, \ldots\}$. Each calculus uses a countable set of actions $\mathbf{Act} = \{\alpha, \beta, \gamma, \ldots\}$ for its transition semantics. In some of the calculi we consider the processes also manipulate natural numbers, ranged over by $n, m, \ldots, i, j, \ldots$. We present all these calculi here as they have many similarities. In later sections only one calculus will be considered in each section so that the sorting of names stays clear.

Asynchronous CCS The first process calculus we consider is *asynchronous CCS* [4]. In CCS, processes communicate through outputs and inputs with or without value-passing. The variant of CCS considered here is asynchronous and does not have value passing. Asynchrony means that the output particles do not have continuations; they can be consumed whenever an input on the same channel name is available.

The processes of asynchronous CCS follow this syntax:

$$\begin{aligned} P,Q &:= \overline{a} \,|\, M \,|\, P \mid Q \,|\, \boldsymbol{\nu} a P \,|\, !P \\ M,N &:= \mathbf{0} \,|\, a.P \,|\, \tau.P \,|\, M+N \end{aligned}$$

Its semantics is described by a transition relation $\xrightarrow{\alpha}$ where α is an action. Instead of recursive definitions that are often used in CCS, we chose to use the replication operator $!P$. This choice was made to keep proximity with the π-calculus and because we do not need the expressive power offered by recursive definitions. A replicated process $!P$ is meant to be always available even after performing some action, i.e, if $P \xrightarrow{\alpha} Q$ then $!P \xrightarrow{\alpha} Q \mid !P$. The other transitions are standard and left to Appendix A.1.

In asynchronous calculi, the choice operator can only be used for processes prefixed by an input or a τ-action (or another choice, or $\mathbf{0}$). In the following we will need internal choice between processes having outputs at top level. Internal choice $P \oplus Q$ is syntactic sugar for $\tau.P + \tau.Q$. This means that instead of reducing directly as $P+Q \xrightarrow{\alpha} R$ if $P \xrightarrow{\alpha} R$ the choice will be done internally with a τ action before performing α, $P \oplus Q \xrightarrow{\tau} P \xrightarrow{\alpha} R$.

Synchronous CCS with value-passing Another variant of CCS we consider is *synchronous CCS with value-passing* [4]. Compared to the previous variant, output particles are now allowed to have continuations and channels can communicate natural numbers.

The processes of synchronous CCS with value-passing follow this syntax:

$$\begin{aligned} \pi &:= \overline{a}\langle n \rangle \,|\, a(n) \,|\, \tau \\ P,Q &:= M \,|\, P \mid Q \,|\, \boldsymbol{\nu} a P \,|\, !P \,|\, \texttt{if}\, n = m \,\texttt{then}\, P \,\texttt{else}\, Q \\ M,N &:= \mathbf{0} \,|\, \pi.P \,|\, M+N \end{aligned}$$

Its semantics is described by a transition relation $\xrightarrow{\alpha}$ where α is an action. The actions now have the form $\overline{a}\langle n \rangle, a\langle n \rangle$. When adding value-passing, we also add an `if then else` operator to branch depending on natural numbers. The transition relations are similar to the ones in asynchronous CCS (see Appendix A.2 for the rules of the transition relation).

Asynchronous π-calculus Another process calculus we consider is the *asynchronous π-calculus* [8]. π-calculus is more complex than CCS in that the channels can be used to communicate channel names and a process can generate new names throughout its execution. The variant of π-calculus we consider is asynchronous, hence we have the same constraint as for asynchronous CCS that output particles do not have continuations.

The processes of asynchronous π-calculus follow this syntax:

$$\begin{aligned} P,Q &:= \overline{a}\langle b \rangle \,|\, M \,|\, P \mid Q \,|\, \boldsymbol{\nu} a P \,|\, !P \\ M,N &:= \mathbf{0} \,|\, a(b).P \,|\, \tau.P \,|\, M+N \,|\, [x=y]M \end{aligned}$$

Its semantics is described by a transition relation $\xrightarrow{\alpha}$ where α is an action. We use an early transition semantics, meaning that when an input is performed, the name received is used to instantiate the parameter of the input particle: $a(c).P \xrightarrow{a\langle b\rangle} P\{b/c\}$. The other rules of the transition relations are standard and left to Appendix A.3

The calculus considered also has the matching operator $[x=y]\,P$ that was not present in asynchronous CCS. With this operator, the process P can only perform actions when x and y are the same channels. This means that $[x=y]\,P \xrightarrow{\alpha} Q$ is only possible when $x=y$. This operator is useful when combined with the choice operator as in $Q+[x=y]\,P$ where the second branch can only be chosen if $x=y$.

As in asynchronous CCS, we use the internal choice $P \oplus Q$ as syntactic sugar for $\tau.P+\tau.Q$.

For simplicity, we assume that some names do not communicate any channels even though we use the same meta-variables as for other names, i.e. actions on these names are a and $\overline{a}$. These names are the ones we use to generate barbs. We sometimes also use these names when the additional communicated channel would be irrelevant.

Synchronous π-calculus The final calculus we consider is the *synchronous π-calculus* [8]. As in synchronous CCS with value-passing processes can use output as prefix and exchange natural numbers.

The processes of synchronous π-calculus with natural numbers follow this syntax:

$$\begin{aligned}\pi &::= \overline{a}\langle n,b\rangle \,\big|\, a(n,b) \,\big|\, \tau \,\big|\, [x=y]\,\pi \\ P,Q &::= M \,\big|\, P \mid Q \,\big|\, \boldsymbol{\nu} aP \,\big|\, !P \,\big|\, \texttt{if}\, n=m\, \texttt{then}\, P\, \texttt{else}\, Q \\ M,N &::= \mathbf{0} \,\big|\, \pi.P \,\big|\, M+N\end{aligned}$$

The actions now have the form $\overline{a}\langle n\rangle(b)$ for bound-outputs, $\overline{a}\langle n,b\rangle$ for free-outputs and $a\langle n,b\rangle$ for inputs. We also have the `if then else` operator to branch depending on natural numbers. Otherwise the transition semantics is similar to the one for the asynchronous π-calculus. We refer to Appendix A.4 for the complete transition relation rules.

Similarly as for asynchronous π-calculus, we assume that some names only communicate natural numbers even though we use the same meta-variables as for other names, i.e. actions on these names are $a\langle n\rangle$ and $\overline{a}\langle n\rangle$. These names are the ones we use to generate barbs. We sometimes also use these names when the additional communicated channel would be irrelevant.

Standard equivalences We define here the different equivalences for process calculi. These definitions are the same for all the calculi we consider but are instantiated with different processes and different actions. These equivalences are weak; this means that they abstract from internal process reductions. Weak transitions are defined as follow: We write $\Rightarrow$ for the reflexive and transitive closure of $\xrightarrow{\tau}$; and for an action α we write $P \overset{\alpha}{\Rightarrow} Q$ if $P \Rightarrow \xrightarrow{\alpha} \Rightarrow Q$.

We write $P \Downarrow_a$ to indicate the possibility for P to exhibit a barb at a, possibly after some reductions. This means that there exists P' such that $P \Rightarrow P'$ and P' has an unguarded occurence of $\overline{a}$.

Structural congruence, written $\equiv$, is a syntactic congruence on processes. It states for example that the parallel composition and the choice operator are associative and commutative. It also states that restrictions can be commuted and moved around as long as doing so does not create free names: $\boldsymbol{\nu} a(P \mid Q) \equiv (\boldsymbol{\nu} aP) \mid Q$ when $a \notin \texttt{fn}(Q)$. Processes are considered up to $\equiv$, which allows us to write n-ary parallel compositions, written $\prod_{i\in I} P_i$, and restriction of tuples of names, written $\boldsymbol{\nu}\tilde{a}P$. The rules

for structural congruence are described in Appendix B

In process calculi, the reference behavioural equivalence is usually barbed congruence:

Definition 1 ((weak) Barbed bisimilarity and congruence)**.**

- A symmetric relation $\mathscr{R}$ is a barbed bisimulation if for all processes P, Q such that $P\mathscr{R}Q$:
  - If $P \Rightarrow P'$ then there exists Q' such that $Q \Rightarrow Q'$ and $P'\mathscr{R}Q'$
  - For any name a, if $P \Downarrow_a$ then also $Q \Downarrow_a$
- Barbed bisimilarity, written $\dot{\approx}$, is the largest barbed bisimulation.
- Two processes P, Q are barbed congruent, written $P \simeq Q$, if for all contexts C, $C[P] \dot{\approx} C[Q]$.

Another variant of barbed congruence that is sometimes considered is open barbed bisimilarity [3]: the equivalence defined as the largest barbed bisimulation that is a congruence. When proving characterisation of open barbed bisimilarity with bisimilarity, there are no issues with image finiteness as barbed congruence is both a barbed bisimulation and a congruence.

The usual characterization of barbed congruence is given by bisimilarity:

Definition 2 ((weak) Bisimilarity)**.**

- A symmetric relation $\mathscr{R}$ is a bisimulation if for all processes P, Q such that $P\mathscr{R}Q$ for any action α, if $P \overset{\alpha}{\Rightarrow} P'$ then there exist Q' such that $Q \overset{\alpha}{\Rightarrow} Q'$ and $P'\mathscr{R}Q'$
- Bisimilarity, written $\approx$, is the largest bisimulation.

In synchronous π-calculus, bisimilarity does not directly coincide with barbed congruence, to obtain the characterization we need to take the closure under substitutions of bisimilarity. In fact, the equivalence that coincides with bisimilarity is barbed equivalence. It is a well known result that the closure under substitutions of bisimilarity is the largest congruence included in bisimilarity. When considering synchronous π-calculus, we will focus on the coincidence between bisimilarity and barbed equivalence.

Definition 3 (Barbed equivalence)**.** Two processes P, Q are barbed equivalent, written $P \cong Q$, if for all processes R and tuple of names $\tilde{a}$, $\boldsymbol{\nu}\tilde{a}(P \mid R) \dot{\approx} \boldsymbol{\nu}\tilde{a}(Q \mid R)$.

2 Weighted Bards

In this section we introduce two notions of weighted barbs, one for asynchronous calculi and one for synchronous calculi. We also show that the resulting barbed congruence coincides with the usual one and sketch a proof of characterisation of barbed congruence as bisimilarity using weighted barbs.

We start with the definition of weighted barbs for asynchronous calculi.

Definition 4 (Weighted barbs for asynchronous calculi)**.** We say that a process P has weighted barb at a^n, written $P \Downarrow_a^n$, if there exists Q such that $P \Rightarrow \equiv \underbrace{\overline{a} \mid \ldots \mid \overline{a}}_{n \text{ times}} \mid Q$.

This definition does not impose that n is unique: if $P \Downarrow_a^n$ and $m \leq n$, then $P \Downarrow_a^m$ also holds. For synchronous calculi this definition would not make sense as output particles can have continuations and it becomes difficult to observe syntactically what multiplicity to give to certain barbs. For example: Does the process $P = \overline{a}.\tau.\overline{a}$ have only have barb a^1 or also a^2? Its asynchronous counterpart could be written $\overline{a} \mid \tau.\overline{a}$ which can reduce to $\overline{a} \mid \overline{a}$ and therefore has barb at a^2. In contrast, P needs to perform some internal reduction between the two outputs at a and therefore one could argue that it should only have

barb at a^1. Another issue, is that if we wanted to count for example the number of unguarded barbs, the test on barbs would discriminate $\overline{a}.\overline{a}$ and $\overline{a} \mid \overline{a}$ which are bisimilar. These are the kind of problems that bring into the realm of true-concurrency semantics, where actions can occur at the same time instead of allowing the different interleavings of these actions.

To settle these issues, we require that the barb in itself provides explicitly its weight by communicating a natural number. Hence the following definition:

Definition 5 (Weighted barbs for synchronous calculi)**.** We say that a process P has weighted barb at a^n, written $P \Downarrow_a^n$, if there exists Q such that $P \Rightarrow Q$ and Q has an unguarded occurence of $\overline{a}\langle n \rangle$.

We can now instantiate the usual definition of barbed congruence with weighted barbs to replace barbs.

Definition 6 (Weighted barbed bisimilarity and congruence)**.**

- A symmetric relation $\mathscr{R}$ is a weighted barbed bisimulation if for all P, Q such that $P \mathscr{R} Q$:
  - If $P \Rightarrow P'$ then there exists Q' such that $Q \Rightarrow Q'$ and $P' \mathscr{R} Q'$
  - For any name a and natural number n, if $P \Downarrow_a^n$ then also $Q \Downarrow_a^n$
- Weighted barbed bisimilarity, written $\dot{\approx}_{\mathtt{w}}$, is the largest weighted barbed bisimulation.
- Two processes P, Q are weighted barbed congruent, written $P \simeq_{\mathtt{w}} Q$, if for all contexts C, $C[P] \dot{\approx}_{\mathtt{w}} C[Q]$.
- Two processes P, Q are weighted barbed equivalent, written $P \cong_{\mathtt{w}} Q$, if for all processes R and tuple of names $\tilde{a}$, $\boldsymbol{\nu}\tilde{a}(P \mid R) \dot{\approx}_{\mathtt{w}} \boldsymbol{\nu}\tilde{a}(Q \mid R)$.

In order to understand how these weighted barbs can help us with proving the characterisation of barbed congruence as bisimilarity, let us take a look at the usual method used to make this proof. The inclusion of bisimilarity in barbed congruence usually follows from bisimilarity being a congruence, this part of the proof is not impacted by the use of weighted barbs. We focus on the inclusion of barbed congruence in bisimilarity :

1. Bisimilarity is stratified in relations $\approx^n$ where bisimulation games are played up-to depth n, and the relation $\approx^\omega$ is defined as the intersection of all the $\approx^n$. Thus $P \approx^\omega Q$ means that for any fixed n, P and Q cannot be distinguished by a bisimulation game up-to depth n
2. $\approx^\omega$ is shown to coincide with $\approx$ for image-finite processes
3. For two image finite processes P, Q such that $P \not\approx Q$, there exists n such that $P \not\approx^n Q$ and therefore there exists a bisimulation game of length at most n that distinguishes P and Q
4. One defines inductively a context that follows this bisimulation game and can distinguish P and Q within barbed bisimulation. Hence $P \not\simeq Q$. In this inductive construction we need the image-finiteness hypothesis in order to have a finite number of induction hypotheses to apply and keep a finite context.

We can see that there are two places where image-finiteness is important, in points (2) and (4), so if we do not want to require image-finiteness we need to focus on these two points.

In point (4), one solution is to build a universal context. This means that instead of only testing the actions from the bisimulation game, the context will be able to test all possible actions. This set of actions is finite because the number of free names in the processes is finite. This way, we will only need one induction hypothesis for all the possibles images of the processes as their free names are bound by their previous free names and the names present in the action performed.

In point (2), this restriction to image-finite processes means that we need to consider infinite bisimulation games. This was handled by Sangiorgi [7] using transfinite induction, however this technique required the use of choices between infinitely many processes and we want to keep the processes finite. The solution we consider is to transform the previous induction steps when building the universal context into recursion steps of the context. Most of the operations performed in the induction proof can be performed at the process level without too many difficulties. However one operation that cannot be done is generating fresh names that can be used to perform barbs. This is where weighted barbs come into play: while keeping the number of available names finite, weighted barbs give access to an infinite number of different barbs.

The weighted barbs can, for example, be used to count the depth at which the bisimulation game is simulated by the context and thus distinguish two different iterations of the recursions on the context.

For each of the calculi considered, we show that we can build contexts that will relate weighted barbed bisimilarity with barbed bisimilarity.

Lemma 7. *For any processes P,Q, there exist a tuple of names $\tilde{a}$ and a context C such that if $\boldsymbol{\nu}\tilde{a}(P \mid C) \dot{\approx} \boldsymbol{\nu}\tilde{a}(Q \mid C)$ then $P \dot{\approx}_{\mathtt{w}} Q$.*

Proof. We start with the asynchronous case. We show it for processes that can only make weighted barbs on one name b, the general case can be deduced from this one through a simple induction. Let $x,y \notin \mathtt{fn}(P,Q)$

$$F := f_1.\mathbf{0} \mid f_2.\overline{x} \mid f_3.\overline{y} \mid !b.(f_1.G_1 \oplus f_2.G_2 \oplus f_3.G_3)$$

$$G_0 := \overline{f_1} \oplus \overline{f_2} \oplus \overline{f_3} \qquad G_1 := \overline{f_2} \oplus \overline{f_3} \qquad G_2 := \overline{f_1} \oplus \overline{f_3} \qquad G_3 := \overline{f_1} \oplus \overline{f_2}$$

$$C := \boldsymbol{\nu} f_1, f_2, f_3\,(F \mid G_0)$$

We show that for any n,m,P,Q,i,j such that $P \Downarrow_b^n, P \not\Downarrow_b^{n+1}, Q \Downarrow_b^m, Q \not\Downarrow_b^{m+1}$ if $(n,i) \neq (m,j)$ then $\boldsymbol{\nu}b(P \mid F \mid G_i) \dot{\not\approx} \boldsymbol{\nu}b(Q \mid F \mid G_j)$. This is shown by induction on n and m, the idea being that (considering $m \geq n$) $\boldsymbol{\nu}b(Q \mid F \mid G_j)$ can reduce to some $\boldsymbol{\nu}b(Q' \mid F \mid G_k)$ such that if $m \neq n$ then $Q' \Downarrow_b^n, Q' \not\Downarrow_b^{n+1}$ and $k \neq i$ and if $m = n$, without loss of generality assuming $i \neq 0$ then $Q \Downarrow_b^{n-1}, Q \not\Downarrow_b^n$ and $k = i$, this way for any reduction of $\boldsymbol{\nu}b(P \mid F \mid G_i)$ either the new n will be different from $m-1$ or the new i will be different from k.
We conclude by showing that $\{(P,Q) \mid x,y \notin \mathtt{fn}(P,Q) \text{ and } \mathtt{fn}(P,Q) = \{b\} \text{ and } \boldsymbol{\nu}b(P \mid C) \dot{\approx} \boldsymbol{\nu}b(Q \mid C)\}$ is a weighted barbed bisimulation.

For the synchronous case, all we need to do is to build a process that synchronizes with the output $\overline{a}\langle n \rangle$ and generates n outputs on b where b is taken fresh and combine it with the previous context C. This process is the following :

$$!a(n).\mathtt{if}\, n = 0\, \mathtt{then}\, \mathbf{0}\, \mathtt{else}\, (\overline{b} \mid \overline{a}\langle n-1 \rangle)$$

□

From this result we can deduce that for each of the calculi we consider we have that weighted barbed congruence coincides with barbed congruence and weighted barbed equivalence coincides with barbed equivalence.

Theorem 8 (Weighted/non-weighted)**.**

- $\simeq_{\mathtt{w}} = \simeq$
- $\cong_{\mathtt{w}} = \cong$

3 Asynchronous CCS

The goal of this section is to prove that weighted barbed congruence is included in bisimilarity in asynchronous CCS using universal distinguishing contexts that simulate bisimulation games.

We chose to start with asynchronous CCS as it is the simplest of the calculi we will consider. This is mainly because a process cannot introduce new free names; bound names can be introduced using restrictions but the actions of these names will not be observed by any bisimulation game. Therefore, we can build universal contexts to test only a fixed, finite set of actions A. The role of the context will be to choose one action $\alpha \in A$ and provide the dual of that action for the tested process to synchronize with (in practice we will assume that if $\alpha \in A$ then so is the dual of α, so we directly provide the action α). However, when comparing two processes, we need to ensure that each instance of the context chooses the same action to provide. To do so we use unique barbs for each action in order to distinguish the processes when two different actions were chosen; for this we take a finite set of fresh names $X_A = \{x_\alpha \mid \alpha \in A\}$ (with $\mid A \mid = \mid X_A \mid$). We do not need to use weighted barbs on these names as we only have a finite number of actions. This way we can know if both contexts chose the same action; however, this is not sufficient: sometimes the barbs of the other actions will be available by performing one more iteration of the context, i.e. comparing P and Q, if the context of P choses action α and Q choses action β, then Q could synchronize by performing action β and then the context choses action α in the next iteration. This is where weighted barbs come into play, we use them to count the number of iterations of the context that were performed, we take a fresh name a and on the n-th iteration, the context should provide n barbs at a.

We build a universal context E such that for any processes P, Q such that $\mathtt{fn}(P,Q) \cap (X_A \cup \{a\}) = \emptyset$ and $\{\overline{b}, b \mid b \in \mathtt{fn}(P,Q)\} \subseteq A$, we have that if $P \mid E \dot{\approx}_{\mathtt{w}} Q \mid E$ then $P \approx Q$.
We define E as follow :

$$\mathtt{Main} := \prod_{\alpha \in A} !f.(\mathtt{Test}_\alpha \oplus \alpha.(\overline{f} \mid p.(\overline{a} \mid \overline{p}))) \qquad \mathtt{Test}_\alpha := \overline{x_\alpha} \oplus \overline{p} \qquad N_k := \prod_{1 \le i \le k} p.(\overline{a} \mid \overline{p})$$

$$E_k := \boldsymbol{\nu} f, p(\mathtt{Main} \mid N_k \mid \overline{f}) \qquad E := E_0$$

- f is used for the recursive calls of the context, in `Main`, we have a replicated input on f for each action $\alpha \in A$. This means that at each recursive call, one of the actions will be chosen non deterministically. At the beginning there is one output $\overline{f}$ to trigger the first iteration and at the end of each execution a new output $\overline{f}$ is provided.
- p and N_k are used to know which is the current iteration, $\overline{p} \mid N_k \Rightarrow \overline{p} \mid \underbrace{\overline{a} \mid \ldots \mid \overline{a}}_{k \text{ times}}$, the important thing is that it has barb at a^k and does not have barb at a^{k+1}. We also have that $E_k \stackrel{\alpha}{\Rightarrow} E_{k+1}$ for any natural number k and action $\alpha \in A$.

In order to prove the expected result we need to remark a few properties:

- $T_{\alpha,k} = \boldsymbol{\nu} f, p(\mathtt{Main} \mid \mathtt{Test}_\alpha \mid N_k)$ is the only process such that $E_k \Rightarrow T_{\alpha,k}$, $T_{\alpha,k} \Downarrow_{x_\alpha}$, $T_{\alpha,k} \Downarrow_a^k$, $T_{\alpha,k} \not\Downarrow_a^{k+1}$ and for any action $\beta \neq \alpha$, $T_{\alpha,k} \not\Downarrow_{x_\beta}$.
- $C_{\alpha,k} = \boldsymbol{\nu} f, p((\mathtt{Test}_\alpha \oplus \alpha.(\overline{f} \mid p.(\overline{a} \mid \overline{p}))) \mid N_k)$ is the only process such that $E_k \Rightarrow C_{\alpha,k}$, $C_{\alpha,k} \Rightarrow T_{\alpha,k}$, $C_{\alpha,k} \stackrel{\alpha}{\Rightarrow} E_{k+1}$ and $C_{\alpha,k} \not\Rightarrow T_{\beta,k}$ for $\beta \neq \alpha$.

Lemma 9. *The following relation is a bisimulation.*

$$\mathscr{R}_{A,X_A,a} = \{(P,Q) \mid \exists k,\ \begin{array}{ll} & P \mid E_k \dot{\approx}_{\mathtt{w}} Q \mid E_k \\ \wedge & \mathtt{fn}(P,Q) \cap (X_A \cup \{a\}) = \emptyset \\ \wedge & \{\overline{b}, b \mid b \in \mathtt{fn}(P,Q)\} \subseteq A \end{array} \}$$

Proof. $\dot{\approx}_{\mathtt{w}}$ is symmetric therefore $\mathscr{R}_{A,X_A,a}$ is symmetric.
Let $(P,Q) \in \mathscr{R}_{A,X_A,a}$.
If $P \stackrel{\alpha}{\Rightarrow} P'$, then $\overline{\alpha} \in A$, we have that $E_k \Rightarrow C_{\overline{\alpha},k} \stackrel{\overline{\alpha}}{\Rightarrow} E_{k+1}$ therefore $P \mid E_k \Rightarrow P \mid C_{\overline{\alpha},k} \Rightarrow P' \mid E_{k+1}$. We have that $P \mid E_k \dot{\approx}_{\mathtt{w}} Q \mid E_k$ therefore there exists Q', E' such that $Q \mid E_k \Rightarrow Q' \mid E'$, $P \mid C_{\overline{\alpha},k} \dot{\approx}_{\mathtt{w}} Q' \mid E'$ and there exists a trace t such that $Q \stackrel{t}{\Rightarrow} Q'$ and $E_k \stackrel{\bar{t}}{\Rightarrow} E'$.
$P \mid C_{\overline{\alpha},k} \Rightarrow P \mid T_{\overline{\alpha},k}$ therefore there exists Q'', E'' such that $Q' \mid E' \Rightarrow Q'' \mid E''$ and $Q'' \mid E'' \Downarrow_{x_{\overline{\alpha}}}$, $Q'' \mid E'' \Downarrow_a^k$, $Q'' \mid E'' \not\Downarrow_a^{k+1}$ and for any action $\beta \neq \overline{\alpha}$, $Q'' \mid E'' \not\Downarrow_{x_\beta}$, which is only possible if $E'' = T_{\overline{\alpha},k}$.
Similarly we obtain that $E' \not\Rightarrow T_{\beta,k}$ for some $\beta \neq \overline{\alpha}$, hence $E' = C_{\overline{\alpha},k}$
$P \mid C_{\overline{\alpha},k} \Rightarrow P' \mid E_{k+1}$ therefore there exists Q'' and E'' such that $Q' \mid C_{\overline{\alpha},k} \Rightarrow Q'' \mid E''$ and $P' \mid E_{k+1} \dot{\approx}_{\mathtt{w}} Q'' \mid E''$. We have that for all $\beta \in A$, $P' \mid E_{k+1} \Rightarrow P' \mid T_{\beta,k+1}$, hence with a similar reasoning as earlier we obtain that $E'' = E_{k+1}$, hence $(P', Q'') \in \mathscr{R}_{A,X_A,a}$ □

It is now easy to show that $\simeq_{\mathtt{w}} \subseteq \bigcup_{A,X_A,a} \mathscr{R}_{A,X_A,a} \subseteq \approx$, hence the following theorem :

Theorem 10. $\simeq_{\mathtt{w}} \subseteq \approx$

4 Synchronous CCS

In synchronous calculi, the definition of weighted barbs changes; rather than counting the multiplicity of the barbs, as in Definition 4, we observe directly a natural number provided by the barb, as in Definition 5. Therefore the calculus considered needs to be able to exchange natural numbers. Hence, the synchronous variant of CCS we consider is *Synchronous CCS with value-passing*.

Compared to *asynchronous CCS* there are two main differences that we need to take into account in the proof showing inclusion of weighted barbed congruence in bisimilarity:

- We need to use the synchronous version of weighted barbs, therefore the context can directly use natural numbers in the barbs. This simplifies the context as we do not need to simulate a counter that we increment at each iteration, we simply increment a natural number.
- The calculus uses value-passing, meaning that actions are of the form $\overline{a}\langle n\rangle, a\langle n\rangle$. The number of possible actions is no longer finite. We need to add a way to generate natural numbers.

4.1 Number generator

In order to generate any natural number at will, we use a replicated process that will enumerate natural numbers and can stop at any time to return the current natural number. We will also need to check that both instances of the context chose the same natural number in the bisimulation game, to do so we use two fresh names a and c. As in asynchronous CCS, a is used to make a barb with the index of the current iteration as weight. The name c is used to make a barb with the selected natural number as weight. We use a return channel p to return the selected natural number.

The process used to generate natural numbers is the following one:

$$\texttt{Gen} := !\texttt{gen}(n).\,\boldsymbol{\nu} g \left(\begin{array}{ll} & !g(i).\,(\overline{g}\langle i+1\rangle \oplus (\overline{p}\langle i\rangle \oplus (\overline{a}\langle n\rangle + \overline{c}\langle i\rangle))) \\ \mid & \overline{g}\langle 0\rangle \end{array} \right)$$

After an execution of $\texttt{gen}(n)$ there is a residual $\boldsymbol{\nu} g(!g(i)\dots)$. This is not an issue because $\boldsymbol{\nu} g(!g(i)\dots) \approx \mathbf{0}$; and we can work up-to bisimilarity.

Remark that even though we are in a synchronous setting where the choice operator can be used with output prefixed processes, we still use the internal choice operator $\oplus$, this is to allow reductions that will discard some choices without committing to only one choice: with $\overline{p}\langle i\rangle + (\overline{a}\langle n\rangle + \overline{c}\langle i\rangle)$ we cannot reduce to $\overline{a}\langle n\rangle + \overline{c}\langle i\rangle$ which will be necessary in the proofs as it excludes all the barbs allowed by further reductions. Hence $\overline{p}\langle i\rangle$ is used in an internal choice.

4.2 Universal context

Another consequence of value-passing is that we must treat the cases of input and output differently. For outputs, we can directly send the natural number obtained by the number generator. However, for inputs we must first provide an input that receives any natural number, and then we compare it with the natural number obtained by the number generator.

In order to distinguish the different actions we use two fresh names b, d. Name b is used to make a barb weighted with an index corresponding to the name on which the action is performed. Name d is used to distinguish between inputs and outputs; it will have weight 0 for inputs and 1 for outputs.

We can now construct the universal context for a set of names A and fresh names a, b, c, d:

$$\texttt{Main} := \texttt{Inp} \mid \texttt{Out}$$

$$\texttt{Inp} := \prod_{x\in A} !f(n).\overline{\texttt{gen}}\langle n\rangle.\,p(i).x(j).\,\texttt{if}\, i = j \,\texttt{then}\,(\texttt{Test\_inp}_x \oplus (\overline{f}\langle n+1\rangle))\,\texttt{else}\,\mathbf{0}$$

$$\texttt{Out} := \prod_{x\in A} !f(n).\overline{\texttt{gen}}\langle n\rangle.\,p(i).\overline{x}\langle i\rangle.\,(\texttt{Test\_out}_x \oplus (\overline{f}\langle n+1\rangle)) \qquad \texttt{Test\_inp}_x := \overline{b}\langle i_x\rangle + \overline{a}\langle n\rangle + \overline{d}\langle 0\rangle$$

$$\texttt{Test\_out}_x := \overline{b}\langle i_x\rangle + \overline{a}\langle n\rangle + \overline{d}\langle 1\rangle$$

$$E_k := \boldsymbol{\nu} f, p(\texttt{Main} \mid \overline{f}\langle k\rangle) \qquad\qquad E := E_0$$

The crucial lemma is then:

Lemma 11. *The following relation is a bisimulation up-to bisimilarity.*

$$\mathscr{R}_A = \left\{ \begin{array}{lll} & & P \mid E_k \,\dot{\approx}_{\texttt{w}}\, Q \mid E_k \\ (P,Q) \mid \exists k, & \wedge & a,b,c,d \notin \texttt{fn}(P,Q) \\ & \wedge & \texttt{fn}(P,Q) \subseteq A \end{array} \right\}$$

Thus we deduce that weighted barbed congruence is included in bisimilarity:

Theorem 12. $\simeq_{\texttt{w}} \subseteq \approx$

The only synchronous calculi we consider, in this section and in Section 6, use value passing. However, similar ideas can be used to obtain the same results in synhronous calculi without value passing, this is discussed in the following remark.

Remark 13 (Synchronous calculi without value-passing)**.** The definition we use for synchronous weighted barbs requires that the calculus allows value-passing of natural numbers, which is not always desirable. With synchronous calculi without value-passing, we can still use the idea of weighted barbs by considering the asynchronous definition of weighted barbs. One tricky part when doing this is that, in a synchronous calculus, bisimilarity is not a (asynchronous) weighted barbed bisimulation: the processes $\overline{a} \mid \overline{a}$ and $\overline{a}.\overline{a}$ are equated by bisimilarity and distinguished by weighted barbed bisimilarity. This has two consequences, the first one is that the inclusion of bisimilarity in barbed congruence should be shown using the usual barbed bisimilarity. In asynchronous calculi the weight of the barbs coincide with the amount of consecutive (usual) barbs that can be performed, this means that $P \Downarrow_b^n, P \not\Downarrow_b^{n+1}$ if and only if there exist P' such that $P \xrightarrow{\overline{b}} P'$ and $P \Downarrow_b^{n-1}, P \not\Downarrow_b^n$. The proof of coincidence between (asynchronous) weighted barbed congruence and barbed congruence relies on this property. The second consequence is that this property is not true in synchronous calculi. However, as long as this property holds for the barbs that can be performed by the process, the proof holds even if the calculus is synchronous. Therefore, if we make sure that names used to perform weighted barbs are used in an asynchronous way, we can use (asynchronous) weighted barbed congruence to obtain a characterisation of barbed congruence with bisimilarity in a synchronous calculus.

5 Asynchronous π-calculus

In π-calculus, names can be communicated and therefore new free names can appear throughout the execution of a process. As a consequence, the set of possible actions a process can perform is not fixed but can evolve after some actions. To take this into account, we define name providers: a structure of pool of names that a process can interrogate to obtain any name of the pool. Name providers can also be updated to add new names to the pool of names. However, at each step, the number of possible actions stays finite; let us consider a process P, the potential actions of P are of the form $\overline{a}\langle b\rangle, a\langle b\rangle$ with $a, b \in \mathtt{fn}(P)$ or also $\overline{a}(c)$ and $a\langle c\rangle$ with $a \in \mathtt{fn}(P), c \notin \mathtt{fn}(P)$. In the first two cases, providing the action is similar to the case of asynchronous CCS as the number of possible actions is bound by the free names of the process and do not introduce new free names. In the last two cases, reasoning up to α-conversion, it suffices to consider one instance of such c.

Still, these new names introduced have to be considered for future actions, for example if $P \xrightarrow{\overline{a}(c)} P'$ we have $\mathtt{fn}(P') \subseteq \mathtt{fn}(P) \cup \{c\}$.

5.1 Name providers

A name provider for a set of names A is a process $\mathtt{Prov}_A^f$ that can be interrogated at f and will non-deterministically return any name of A. This means that for any name $a \in A$, $\mathtt{Prov}_A^f \xrightarrow{f\langle p\rangle} \xrightarrow{\overline{p}\langle a\rangle} \mathtt{Prov}_A^f$. We should also be able to update a name provider to add some name $a \notin A$, from $\mathtt{Prov}_A^f$ to $\mathtt{Prov}_{A\cup\{a\}}^f$.

The name provider will have the form :

$$\mathtt{Prov}_A^f := \prod_{a\in A} !f(p).(\overline{p}\langle a\rangle \oplus \mathtt{Test}_a)$$

Where $\mathtt{Test}_a$ is here so that we can distinguish which branch was selected. In order to add a name b, we just have to add $!f(p).(\overline{p}\langle b\rangle \oplus \mathtt{Test}_b)$ in parallel.

In the process $\mathtt{Test}_a$ we want to be able to test:

1. that the selected branch is indeed the one corresponding to a;
2. in which iteration of the whole context was this branch selected. This is to remove the cases where the context makes one more iteration to retrieve the barbs of the other tests;
3. a barb on an additional channel that will be used to distinguish multiple calls to the name provider within the same iteration of the context.

For point (1), we associate to each name $a \in A$ a specific natural number i_a in $1,\ldots,|A|$ ($i_a \neq i_b$ if $a \neq b$) and take $|A|+1$ distinct names which we write $\underline{0},\underline{1},\ldots,\underline{|A|}$. The idea is that an input $\underline{i_a}(g)$ will generate i_a barbs on g. The definitions will have the form $!\underline{i_a}(g).(\overline{g} \mid \overline{\underline{i_a - 1}}\langle g\rangle)$. And in order to build a new $\underline{i_a}$, we provide a channel $\mathtt{max}$ that will provide $\underline{i_b}$ such that i_b is maximal among the current names provided. Last, to use this construction we take a fresh name w_1 on which weighted barbs will be made.

For point (2), the idea will be the same as for asynchronous CCS; the name we will use to trigger the barbs depending on the iterations is r.

For point (3), we simply add an output on a channel name q that the other parts of the context can use to trigger other barbs.

We then have $\mathtt{Test}_a = \overline{\underline{i_a}}\langle w_1\rangle \oplus \overline{r} \oplus \overline{q}$.

Hence the complete name provider is :

$$\mathtt{Prov}_A^f := \boldsymbol{\nu}\tilde{\underline{k}} \left(\begin{array}{ll} & \prod_{a\in A} !f(p).(\overline{p}\langle a\rangle \oplus \mathtt{Test}_a) \\ \mid & \overline{\mathtt{max}}\langle \underline{k}\rangle \\ \mid & \prod_{i\leq k} !\underline{i}(a).(\overline{a} \mid \overline{\underline{i-1}}\langle a\rangle) \end{array} \right)$$

where $k = |A|$ and $\tilde{\underline{k}} = \{\underline{i} \mid i \leq k\}$.

And, in order to add a name to a provider we use the following process :

$$\mathtt{Add\_Prov}_a := \mathtt{max}(\underline{k}).\boldsymbol{\nu}\underline{i_a} \left(\begin{array}{ll} & !f(p).(\overline{p}\langle a\rangle \oplus \mathtt{Test}_a) \\ \mid & !\underline{i_a}(b).(\overline{b} \mid \overline{\underline{k}}\langle b\rangle) \\ \mid & \overline{\mathtt{max}}\langle \underline{i_a}\rangle \end{array} \right)$$

This way we have $\mathtt{Prov}_A^f \mid \mathtt{Add\_Prov}_a \Rightarrow \mathtt{Prov}_{A\cup\{a\}}^f$.

In order to choose which action will be performed, we need to choose two names, one for the channel used to communicate and one that will be communicated. To do so, we perform two requests to $\mathtt{Prov}_A^f$. We need to distinguish the two choices. This will be done using a fresh name w_3 and the name q; on the first request, an output on q will not generate any barb at w_3 and on the second request it will generate 1 barb at w_3, after the request we clean up the temporary inputs at q we added to perform these barbs:

$$\mathtt{Requ}(x,y) := \begin{array}{ll} & \overline{f}\langle p\rangle \\ \mid & p(x).\left(\begin{array}{ll} & \overline{f}\langle p\rangle \\ \mid & q.\overline{w_3} \\ \mid & p(y).(\overline{q} \mid w_3.\bullet) \end{array} \right) \end{array}$$

$\mathtt{Requ}(x,y)$ is a context where $\bullet$ is a hole and it binds the names x,y.

5.2 Main body

In the main body, we will have a channel g to trigger an iteration. It will start by requesting two names x,y and perform one of these four actions : $\overline{x}\langle y\rangle, x\langle y\rangle, \overline{x}(z), x\langle z\rangle$ where z is a new name. We add a fresh name

w_4 that will be used to distinguish among the four kinds of actions. When a new name is introduced, it is added to the pool of names in the name provider.

In the case of an input we cannot control which name will be used in the synchronization, therefore when receiving a name that was already free we use the match operator to create a deadlock when the received name do not match with the name we expected to receive.

We also need to know how many iterations have already been performed (to distinguish two different iterations). To do so we use the name r and a fresh name w_2 for which n barbs will be available at iteration n. Similarly to what is done in asynchronous CCS, when starting a new iteration we provide a new instance of $r.(\overline{w_2} \mid \overline{r})$.

$$\mathtt{Main}_n := !g.\mathtt{Requ}(x,y)[(\mathtt{Inp}_1 \oplus \mathtt{Inp}_2 \oplus \mathtt{Out}_1 \oplus \mathtt{Out}_2)] \mid N_n \qquad N_n := \prod_{1\leq i\leq n} r.(\overline{w_2} \mid \overline{r})$$

$$\mathtt{Inp}_1 := x(z).[z=y]\,((\overline{g} \mid r.(\overline{w_2} \mid \overline{r})) \oplus \overline{r})$$

$$\mathtt{Inp}_2 := x(z).((\overline{g} \mid r.(\overline{w_2} \mid \overline{r}) \mid \mathtt{Add\_Prov}_z) \oplus (\overline{w_4} \oplus \overline{r}))$$

$$\mathtt{Out}_1 := \overline{x}\langle y\rangle \mid ((\overline{g} \mid r.(\overline{w_2} \mid \overline{r})) \oplus ((\overline{w_4} \mid \overline{w_4}) \oplus \overline{r}))$$

$$\mathtt{Out}_2 := \boldsymbol{\nu} z\,\overline{x}\langle z\rangle \mid ((\overline{g} \mid r.(\overline{w_2} \mid \overline{r}) \mid \mathtt{Add\_Prov}_z) \oplus ((\overline{w_4} \mid \overline{w_4} \mid \overline{w_4}) \oplus \overline{r}))$$

5.3 Universal context

We now have all the pieces to build the universal contexts that simulate bisimulation games. Let A a set of names, names $w_1, w_2, w_3, w_4 \notin A$ and a natural number $k \in \mathbb{N}$. We define the process.

$$E(k,A,w_1,w_2,w_3,w_4) := \boldsymbol{\nu} g,f,\mathtt{max},p,q,r\left(\mathtt{Main}_k \mid \mathtt{Prov}_A^f \mid \overline{g}\right)$$

The idea to prove that bisimilarity contains barbed congruence is similar to the proofs in CCS :

Lemma 14. *The following relation is a bisimulation.*

$$\mathscr{R}_{A,w_1,w_2,w_3,w_4} := \left\{ (P,Q) \mid \exists k,\ \begin{array}{ll} & P \mid E(k,A,w_1,w_2,w_3,w_4) \simeq_{\mathtt{w}} Q \mid E(k,A,w_1,w_2,w_3,w_4) \\ \wedge & \mathtt{fn}(P,Q) \subseteq A \\ \wedge & w_1,w_2,w_3,w_4 \notin \mathtt{fn}(P,Q) \end{array} \right\}$$

And we conclude that

Theorem 15. $\simeq_{\mathtt{w}} \subseteq \approx$

6 Synchronous π-calculus

In this section, we show how both adaptations to synchronous calculi and to communication of names can be combined. In synchronous π-calculus, the equivalence that coincides with bisimilarity is not barbed congruence but barbed equivalence, therefore, instead of considering the former as in the previous sections, we consider the latter. Most of the construction ideas can be extrapolated from the previous sections.

In this construction, some communications do not need to communicate both a natural number and a channel when they are irrelevant we use the placeholder "_".

Number generator The number generator can be directly translated from synchronous CCS with value-passing to synchronous π-calculus. We just performed some renaming, w_5 is the fresh name used to make a barb the selected natural number as weight and w_2 is the fresh name used to check the depth of the bisimulation game.

$$\texttt{Gen} := !\texttt{gen}(n,p)\boldsymbol{\nu} h \left(\begin{array}{ll} & \overline{h}\langle 0,p\rangle \\ \mid & !h(i,p).\left(\overline{h}\langle i+1,p\rangle \oplus \left(\overline{p}\langle i,\_\rangle \oplus \left(\overline{w_5}\langle i\rangle + \overline{w_2}\langle n\rangle\right)\right)\right) \end{array} \right)$$

Name providers Name providers can be simplified as we do not need to keep a server for each name that provides the right number of barbs; the correct weighted barb is directly done in the test part. Moreover, the barb used to distinguish multiple calls to the name provider is also directly in the test, in order to do this we need to ask the caller to provide a number proper to the call with $p(j)$.

$$\begin{array}{rll} \texttt{Prov}_A^f := & \prod_{a\in A} !f(n,p).p(j,\_).\left(\overline{p}\langle \_,a\rangle \oplus \left(\overline{w_1}\langle i_a\rangle + \overline{w_2}\langle n\rangle + \overline{w_3}\langle j\rangle\right)\right) \\ \mid & \overline{\texttt{max}}\langle k\rangle \end{array}$$

$$\begin{array}{rll} \texttt{Add\_Prov}_a := \texttt{max}(k). & !f(n,p).p(j,\_).\left(\overline{p}\langle \_,a\rangle \oplus \left(\overline{w_1}\langle k+1\rangle + \overline{w_2}\langle n\rangle + \overline{w_3}\langle j\rangle\right)\right) \\ \mid & \overline{\texttt{max}}\langle k+1\rangle \end{array}$$

Requests For requesting the different variables of the action that will be performed, we add the call to the number generator and adapt to the changes made in the name provider. For simplicity, we also use that we are allowed output prefix processes to do all the calls in a sequence. This request context binds the names x, y and the natural number i

$$\texttt{Requ}(x,y,i) := \overline{f}\langle p\rangle.\overline{p}\langle 0,\_\rangle.p(\_,x).\overline{f}\langle p\rangle.\overline{p}\langle 1,\_\rangle.p(\_,y).\overline{\texttt{gen}}\langle n,p\rangle.p(i,\_).\bullet$$

Main body In the main body, there are two main differences with the asynchronous π-calculus:

- the number of the iteration is received as a parameter, thus the main body is the same for any iteration,
- the actions performed need to either check that the received natural number corresponds to the generated one in the case of inputs, or send the generated natural number in the case of outputs.

$$\texttt{Main} := !g(n).\texttt{Requ}(x,y,i)[(\texttt{Inp}_1 \oplus \texttt{Inp}_2 \oplus \texttt{Out}_1 \oplus \texttt{Out}_2)]$$

$$\texttt{Inp}_1 := x(j,z).[z=y]\,\texttt{if}\, i=j\,\texttt{then}\left(\overline{g}\langle n+1\rangle \oplus \left(\overline{w_2}\langle n\rangle + \overline{w_4}\langle 0\rangle\right)\right)\texttt{else}\,\mathbf{0}$$

$$\texttt{Inp}_2 := x(j,z).\texttt{if}\, i=j\,\texttt{then}\left(\overline{g}\langle n+1\rangle \mid \texttt{Add\_Prov}_z\right) \oplus \left(\overline{w_2}\langle n\rangle + \overline{w_4}\langle 1\rangle\right)\texttt{else}\,\mathbf{0}$$

$$\texttt{Out}_1 := \overline{x}\langle i,y\rangle.\left(\overline{g}\langle n+1\rangle \oplus \left(\overline{w_2}\langle n\rangle + \overline{w_4}\langle 2\rangle\right)\right)$$

$$\texttt{Out}_2 := \boldsymbol{\nu} z\,\overline{x}\langle i,z\rangle.\left(\left(\overline{g}\langle n+1\rangle \mid \texttt{Add\_Prov}_z\right) \oplus \left(\overline{w_2}\langle n\rangle + \overline{w_4}\langle 3\rangle\right)\right)$$

Universal context We now have all the pieces to construct the universal context that simulates bisimulation games. Let A a set of names, names $w_1, w_2, w_3, w_4, w_5 \notin A$ and a natural number $k \in \mathbb{N}$. We define the process.

$$E(k, A, w_1, w_2, w_3, w_4, w_5) := \boldsymbol{\nu} g, f, \mathtt{max}, \mathtt{gen}, p \left(\mathtt{Main} \mid \mathtt{Prov}_A^f \mid \mathtt{Gen} \mid \overline{g}\langle k \rangle \right)$$

The idea to prove that bisimilarity contains barbed congruence is similar to the previous proofs :

Lemma 16. *The following relation is a bisimulation up-to bisimilarity.*

$$\mathscr{R}_{A,w_1,w_2,w_3,w_4,w_5} := \left\{ \begin{array}{lll} & & P \mid E(k, A, w_1, w_2, w_3, w_4, w_5) \simeq_{\mathtt{w}} Q \mid E(k, A, w_1, w_2, w_3, w_4, w_5) \\ (P, Q) \mid \exists k, & \wedge & \mathtt{fn}(P, Q) \subseteq A \\ & \wedge & w_1, w_2, w_3, w_4, w_5 \notin \mathtt{fn}(P, Q) \end{array} \right\}$$

And we conclude that

Theorem 17. $\cong_{\mathtt{w}} \subseteq \approx$

7 Summary of the constructions

The different constructions above include some design choices that are not always relevant. In this section, we summarize informally the important ideas of the constructions so that one can replicate more easily these constructions for different process calculi.

Testing An important idea of distinguishing contexts (not specific to our method) is to use barbs in order to track how the context behaves, i.e., which steps were performed and which choices were made. In our constructions, we have a test immediately after each choice; this test consists of a unique combination of barbs :

- Each outcome of the choice should have a different barb: if the choice is between n actions, then choosing action i could trigger a barb at a^i (for some name a),
- If the same choice can occur multiple times, then they should trigger different barbs each time: for example, each choice can trigger a barb weighted by the number of previous iterations of the context combined with a barb weighted by the number of previous occurrences of this choice in the current iteration.

In asynchronous calculi, in order to track the number of previous iterations of the context, one can encode a counter that will be incremented at each iteration and such that when called it triggers a barb with as much multiplicity as the number of previous iterations. This counter does not need to be replicated as it will only be used once; and we do not need it to come from a complete encoding of the natural numbers and of operations on natural numbers.

Number generators In process-calculi with value-passing, for example for natural numbers when considering synchronous calculi, one usually needs to provide an infinite amount of different actions. In this case and when it is possible for the processes to iterate over these values, the idea is to have a process that iterates over values and returns one at random. Then, when choosing the action to consider, one can request a value to that process and use it in the action. In our constructions, this process is an *number generator*.

Name providers In process-calculi that can exchange names, one needs a process to which names can be requested at will. In our constructions, this process is a *name provider*. It is a structure of pool of names such that at each request it returns a name from the pool at random. When names can be created dynamically, one needs to be able to update name providers by adding a name to the pool.

8 Conclusion

We presented notions of *weighted barbs* and the behavioural equivalence *weighted barbed congruence* for both asynchronous and synchronous calculi. Weighted barbs allow more distinguishing power compared to regular barbs and therefore weighted barbed bisimilarity is strictly stronger than barbed bisimilarity. However, we show that this distinction does not lift to the contextual equivalences, and weighted barbed congruence coincides with barbed congruence in the calculi we considered.

We used these notions to showcase a method to construct universal contexts that are able to simulate bisimulation games and allow for proofs that weighted barbed congruence is included in bisimilarity without needing either contexts of infinite size or restriction to image-finite processes. This method is general enough to cover *asynchronous* and *synchronous* calculi, as well as variants of *CCS* and variants of the π*-calculus*.

References

[1] Yuxin Deng and Wenjie Du. Probabilistic barbed congruence. *Electronic Notes in Theoretical Computer Science*, 190(3):185–203, September 2007. doi:10.1016/j.entcs.2007.07.011.

[2] Cédric Fournet and Georges Gonthier. A hierarchy of equivalences for asynchronous calculi. *The Journal of Logic and Algebraic Programming*, 63(1):131–173, April 2005. doi:10.1016/j.jlap.2004.01.006.

[3] Kohei Honda and Nobuko Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, November 1995. doi:10.1016/0304-3975(95)00074-7.

[4] Robin Milner. *A Calculus of Communicating Systems*. Springer Berlin Heidelberg, 1980. doi:10.1007/3-540-10235-3.

[5] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[6] Robin Milner and Davide Sangiorgi. *Barbed bisimulation*, page 685–695. Springer Berlin Heidelberg, 1992. doi:10.1007/3-540-55719-9_114.

[7] Davide Sangiorgi and David Walker. *On Barbed Equivalences in π-Calculus*, page 292–304. Springer Berlin Heidelberg, 2001. doi:10.1007/3-540-44685-0_20.

[8] Davide Sangiorgi and David Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, July 2001. doi:10.1017/9781316134924.

A Labelled Transition Systems

A.1 Asynchronous CCS

In asynchronous CCS, the actions are :

$$\alpha := \overline{a} \,\big|\, a \,\big|\, \tau$$

Where $\overline{a}$ is an output action, a an input action, and τ a silent action. The labelled transition system for asynchronous CCS processes is described in 1. $\overline{\alpha}$ designate the dual of an action: if $\alpha = a$ then $\overline{\alpha} = \overline{a}$ and if $\alpha = \overline{a}$ then $\overline{\alpha} = a$.

$$\frac{}{\overline{a} \xrightarrow{\overline{a}} \mathbf{0}} \qquad \frac{}{a.P \xrightarrow{a} P} \qquad \frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} aP \xrightarrow{\alpha} \boldsymbol{\nu} aP'}\ a \notin \mathtt{fn}(\alpha)$$

Figure 1: LTS of Asynchronous CCS

A.2 Synchronous CCS with value-passing

In synchronous CCS with value-passing, the actions are :

$$\alpha := \overline{a}\langle n \rangle \,\big|\, a\langle n \rangle \,\big|\, \tau$$

Where $\overline{a}\langle n \rangle$ is an output action, $a\langle n \rangle$ an input action, and τ a silent action. The labelled transition system for synchronous CCS with value-passing processes is described in 2. $\overline{\alpha}$ designate the dual of an action: if $\alpha = a\langle n \rangle$ then $\overline{\alpha} = \overline{a}\langle n \rangle$ and if $\alpha = \overline{a}\langle n \rangle$ then $\overline{\alpha} = a\langle n \rangle$.

$$\frac{}{\overline{a}\langle n \rangle.P \xrightarrow{\overline{a}\langle n \rangle} P} \qquad \frac{}{a(m).P \xrightarrow{a\langle n \rangle} P\{n/m\}} \qquad \frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\overline{\alpha}} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} aP \xrightarrow{\alpha} \boldsymbol{\nu} aP'}\ a \notin \mathtt{fn}(\alpha)$$

$$\frac{}{\mathtt{if}\, n = n \,\mathtt{then}\, P \,\mathtt{else}\, Q \xrightarrow{\tau} P} \qquad \frac{}{\mathtt{if}\, n = m \,\mathtt{then}\, P \,\mathtt{else}\, Q \xrightarrow{\tau} Q}\ n \neq m$$

Figure 2: LTS of Synchronous CCS with value-passing

A.3 Asynchronous π-calculus

In asynchronous-π, the actions are :

$$\alpha := \overline{a}\langle b\rangle \,|\, \overline{a}(b) \,|\, a\langle b\rangle \,|\, \tau$$

Where $\overline{a}\langle b\rangle$ is an free output action, $\overline{a}(b)$ is a bound output action, $a\langle b\rangle$ an input action, and τ a silent action. The labelled transition system for synchronous CCS with value-passing processes is described in 3.

$$\frac{}{\overline{a}\langle b\rangle \xrightarrow{\overline{a}\langle b\rangle} \mathbf{0}} \qquad \frac{}{a(b).P \xrightarrow{a\langle c\rangle} P\{c/b\}} \qquad \frac{P \xrightarrow{a\langle b\rangle} P' \qquad Q \xrightarrow{\overline{a}\langle b\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{\overline{a}\langle b\rangle} P' \qquad Q \xrightarrow{a\langle b\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{a\langle b\rangle} P' \qquad Q \xrightarrow{\overline{a}(b)} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} b(P' \mid Q')} \qquad \frac{P \xrightarrow{\overline{a}(b)} P' \qquad Q \xrightarrow{a\langle b\rangle} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} b(P' \mid Q')} \qquad \frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{P + Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P + Q \xrightarrow{\alpha} Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} aP \xrightarrow{\alpha} \boldsymbol{\nu} aP'}\ a \notin \mathtt{fn}(\alpha)$$

$$\frac{P \xrightarrow{\overline{a}\langle b\rangle} P'}{\boldsymbol{\nu} bP \xrightarrow{\overline{a}(b)} P'} \qquad \frac{P \xrightarrow{\alpha} P'}{[a = a]\, P \xrightarrow{\alpha} P'}$$

Figure 3: LTS of Asynchronous π-calculus

A.4 Synchronous π-calculus

In asynchronous-π, the actions are :

$$\alpha := \overline{a}\langle n, b\rangle \,|\, \overline{a}\langle n\rangle(b) \,|\, a\langle n, b\rangle \,|\, \tau$$

Where $\overline{a}\langle n, b\rangle$ is an free output action, $\overline{a}\langle n\rangle(b)$ is a bound output action, $a\langle b\rangle$ an input action, and τ a silent action. The labelled transition system for synchronous CCS with value-passing processes is described in 4.

B Structural congruence

Structural congruence is defined similarly for all the calculi we consider; it will only differ on the prefixes available in the calculus.

Structural congruence, written $\equiv$ is the smallest equivalence relation that contains α-conversion, is preserved by all operators of the calculus and satisfies the axioms in Figure 5.

$$\frac{}{\overline{a}\langle n,b\rangle.P \xrightarrow{\overline{a}\langle n,b\rangle} P} \qquad \frac{}{a(m,b).P \xrightarrow{a\langle n,c\rangle} P\{n/m,c/b\}} \qquad \frac{P \xrightarrow{a\langle n,b\rangle} P' \qquad Q \xrightarrow{\overline{a}\langle n,b\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\frac{P \xrightarrow{\overline{a}\langle n,b\rangle} P' \qquad Q \xrightarrow{a\langle n,b\rangle} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \frac{P \xrightarrow{a\langle n,b\rangle} P' \qquad Q \xrightarrow{\overline{a}\langle n\rangle(b)} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} b(P' \mid Q')} \qquad \frac{P \xrightarrow{\overline{a}\langle n\rangle(b)} P' \qquad Q \xrightarrow{a\langle n,b\rangle} Q'}{P \mid Q \xrightarrow{\tau} \boldsymbol{\nu} b(P' \mid Q')}$$

$$\frac{}{\tau.P \xrightarrow{\tau} P} \qquad \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \frac{P \xrightarrow{\alpha} P'}{P+Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P+Q \xrightarrow{\alpha} Q'}$$

$$\frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P} \qquad \frac{P \xrightarrow{\alpha} P'}{\boldsymbol{\nu} aP \xrightarrow{\alpha} \boldsymbol{\nu} aP'}\ a \notin \mathtt{fn}(\alpha) \qquad \frac{P \xrightarrow{\overline{a}\langle n,b\rangle} P'}{\boldsymbol{\nu} bP \xrightarrow{\overline{a}\langle n\rangle(b)} P'} \qquad \frac{P \xrightarrow{\alpha} P'}{[a=a]P \xrightarrow{\alpha} P'}$$

$$\frac{}{\mathtt{if}\, n=n \,\mathtt{then}\, P \,\mathtt{else}\, Q \xrightarrow{\tau} P} \qquad \frac{}{\mathtt{if}\, n=m \,\mathtt{then}\, P \,\mathtt{else}\, Q \xrightarrow{\tau} Q}\ n \neq m$$

Figure 4: LTS of Synchronous π-calculus

$$\overline{P \mid Q \equiv Q \mid P} \qquad \overline{P \mid \mathbf{0} \equiv P} \qquad \overline{P \mid (Q \mid procC) \equiv (P \mid Q) \mid R} \qquad \overline{P+Q \equiv Q+P} \qquad \overline{P+\mathbf{0} \equiv P}$$

$$\overline{P+(Q+R) \equiv (P+Q)+R} \qquad \frac{a \notin \mathtt{fn}(Q)}{\boldsymbol{\nu} a(P \mid Q) \equiv \boldsymbol{\nu} aP \mid Q} \qquad \overline{\boldsymbol{\nu} a \boldsymbol{\nu} bP \equiv \boldsymbol{\nu} b \boldsymbol{\nu} aP} \qquad \overline{!P \equiv !P \mid P}$$

Figure 5: Structural congruence