The Concurrent Calculi Formalisation Benchmark

https://concurrentbenchmark.github.io/

1 Challenge Description

1.1 Preliminaries

First, we list some common notions and conventions that we use in the challenges. Since the calculi under study are somewhat different, each section lists the changes that apply.

We assume the existence of some set of *base values*, represented by the symbols $a, b, \ldots$, of some set of *variables*, represented by the symbols $l, m, \ldots$, and of some set of *names*, represented by the symbols $x, y, \ldots$.[1] We assume that all of these sets are infinite and that their elements can be compared for equality.

The syntax of processes includes: the process $\mathbf{0}$ or *inaction*, a process which can do nothing. The process $P \mid Q$ is the *parallel composition* of process P and process Q. The two components can proceed independently of each other, or they can interact via shared names.

For communication, processes include *input* and *output*, whose signature signature depends on the calculus being value-passing or name-passing. We use here the metavariables c, k to abstract over this choice — i.e. c may be either a value or a variable or a name, whereas k may be a variable or a name. The process $x!c.P$ is an *output*, which can send c via x, then continue as P. The process $x?(k).P$ is an *input*, which can receive a c via x, then continue as P with the received element substituted for k. The input operator thus binds k in P.

The process $(\boldsymbol{\nu} x)\ P$ is the *restriction* of the name x to P, binding x in P.

The process $!P$ is the *replication* of the process P. It can be thought of as the infinite composition $P \mid P \mid \cdots$. Replication makes it possible to express infinite behaviours.

We use the notation $\mathrm{fn}(P)$ to denote the set of names that occur free, $\mathrm{bn}(P)$ to denote the set of names that occur bound in P and $\mathrm{fv}(P)$ to denote the set of variables that occur free in P. We use the notation $\mathrm{bv}(P)$ for the set of variables that occur bound in P. We use the notation $P\{a/l\}$ to denote the process P with base value a substituted for variable l. Similarly, $P\{x/y\}$ denotes the process P with name x substituted for name y. We use the notation $P\sigma$ to denote the process P with a finite number of arbitrary substitutions applied to it.

[1] Unlike the standard π-calculus, we distinguish variables from names to better control the expressiveness of the calculi under study, and the scope of the corresponding challenges: the key distinction is that names are used as communication channels (and can be sent and received in the scope extrusion challenge), whereas variables are only bound by inputs and cannot be restricted, sent nor received, in the style of value-passing CCS [3].

Two processes P and Q are α-convertible, written $P =_\alpha Q$, if Q can be obtained from P by a finite number of substitutions of bound variables. As a convention, we identify α-convertible processes and we assume that bound names and bound variables of any processes are chosen to be different from the names and variables that occur free in any other entities under consideration, such as processes, substitutions, and sets of names or variables. This is justified because any overlapping names and variables may be α-converted such that the assumption is satisfied.

A *context* is obtained by taking a process and replacing a single occurrence of $\mathbf{0}$ in it with the special *hole* symbol $[\cdot]$. As a convention, we do *not* identify α-convertible contexts. A context acts as a function between processes: a context C can be *applied* to a process P, written $C[P]$, by replacing the hole in C by P, thus obtaining another process. The replacement is literal, so names and variables that are free in P can become bound in $C[P]$.

We say that an equivalence relation $\mathcal{S}$ is *compatible* if $(P, Q) \in \mathcal{S}$ implies that for any context C, $(C[P], C[Q]) \in \mathcal{S}$.

1.2 Challenge: Linearity and Behavioural Type Systems

This challenge formalises a proof that requires reasoning about the linearity of channels. Linearity is the notion that a channel must be used exactly once by a process. This is necessary to prove properties about session type systems, and the key issue of this challenge is reasoning about the linearity of context splitting operations. Linear reasoning is also necessary to formalise, *e.g.*, linear and affine types for the π-calculus and cut elimination in linear logics.

The setting for this challenge is a small calculus with a session type system, the syntax and semantics of which are given below. The calculus is a fragment of the one presented in [5], formulated in the dual style of [1].

The main objective of this challenge is to prove type preservation (also known as subject reduction), *i.e.*, that well-typed processes can only transition to processes which are also well-typed in the same context. The second objective is to prove type safety, *i.e.*, that well-typed processes are also well-formed in the sense that they do not use endpoints in a non-dual way.

Syntax. The syntax is given by the grammar

$$\begin{array}{lcl} v, w & ::= & a \mid l \\ P, Q & ::= & \mathbf{0} \mid x!v.P \mid x?(l).P \mid (P \mid Q) \mid (\boldsymbol{\nu} xy)\ P \end{array}$$

where a *value* $v, w, \ldots$ is either a base value a or a variable l.

The output process $x!v.P$ sends the value v via x and then continues as P. The intention is that the value v must be a base value when it is actually sent, and this is enforced in the semantics later on. The input process $x?(l).P$ waits for a base value from x and then continues as P with the received value substituted for the variable l. The process $(\boldsymbol{\nu} xy)\ P$ represents a *session* with endpoints named x and y which are bound in P. In P, the names x and y can be used to exchange

messages over the session (sending on x and receiving on y or vice versa). Note that in this calculus channels cannot be sent in messages, therefore the topology of the communication network described by a process cannot change. Also, there is no recursion or replication in the syntax, hence no infinite behaviours can be expressed. In particular, we only model linear (as opposed to shared) channels.

Semantics. We describe the actions that the system can perform through a small step operational semantics. As usual, we use a *structural congruence* relation that equates processes that we deem to be indistinguishable. Structural congruence is the smallest compatible equivalence relation that satisfies the following axioms:

$$\textsc{Sc-Par-Comm}\ \frac{}{P \mid Q \equiv Q \mid P} \qquad \textsc{Sc-Par-Assoc}\ \frac{}{(P \mid Q) \mid R \equiv P \mid (Q \mid R)} \qquad \textsc{Sc-Par-Inact}\ \frac{}{P \mid \mathbf{0} \equiv P}$$

$$\textsc{Sc-Res-Par}\ \frac{}{(\boldsymbol{\nu} xy)\ P \mid Q \equiv (\boldsymbol{\nu} xy)\ (P \mid Q)} \qquad \textsc{Sc-Res-Inact}\ \frac{}{(\boldsymbol{\nu} xy)\ \mathbf{0} \equiv \mathbf{0}}$$

$$\textsc{Sc-Res}\ \frac{}{(\boldsymbol{\nu} x_1 y_1)\ (\boldsymbol{\nu} x_2 y_2)\ P \equiv (\boldsymbol{\nu} x_2 y_2)\ (\boldsymbol{\nu} x_1 y_1)\ P}$$

The operational semantics are defined as the following relation on processes:

$$\textsc{R-Com}\ \frac{}{(\boldsymbol{\nu} xy)\ (x!a.P \mid y?(l).Q \mid R) \to (\boldsymbol{\nu} xy)\ (P \mid Q\{a/l\} \mid R)} \qquad \textsc{R-Res}\ \frac{P \to Q}{(\boldsymbol{\nu} xy)\ P \to (\boldsymbol{\nu} xy)\ Q}$$

$$\textsc{R-Par}\ \frac{P \to Q}{P \mid R \to Q \mid R} \qquad \textsc{R-Struct}\ \frac{P \equiv P' \qquad P' \to Q' \qquad Q \equiv Q'}{P \to Q}$$

Note that reductions are allowed only for restricted pairs of session endpoints. This makes it possible to formulate subject reduction so that the typing context is exactly the same before and after each reduction. Note also that due to rule R-Com, the process $y?(l).P$ can receive *any* base value. Since the rule R-Com only applies to sending base values, there is no way to send a variable or a name.

Session Types. Our process syntax allows us to write processes that are ill formed in the sense that they either use the endpoints bound by a restriction to communicate in a way that does not follow the intended duality, or attempt to send something which is not a base value. As an example, the process $(\boldsymbol{\nu} xy)\ (x!a.\mathbf{0} \mid y!a.\mathbf{0})$ attempts to send a base value on both x and y, whereas one of the names should be used for receiving in order to guarantee progress. Another example is the process $(\boldsymbol{\nu} xy)\ (x!l.\mathbf{0} \mid y?(l).\mathbf{0})$, which attempts to send a variable that is not instantiated at the time of sending. To prevent these issues, we introduce a *session type system* which rules out ill-formed processes.

Syntax. Our type system does not type processes directly, but instead focuses on the channels used in the process. The syntax of *session types* S, T, unrestricted typing contexts Γ and linear typing contexts Δ is as follows:

$$\begin{array}{rcl} S, T & ::= & \mathbf{end} \mid \mathbf{base} \mid ?.S \mid !.S \\ \Gamma & ::= & \cdot \mid \Gamma, l \\ \Delta & ::= & \cdot \mid \Delta, x : S \end{array}$$

The *end type* **end** describes an endpoint over which no further interaction is possible. The *base type* **base** describes base values. The *input type* $?.S$ describes endpoints used for receiving a value and then according to S. The *output type* $!.S$ describes endpoints used for sending a value and then according to S.

Typing contexts gather type information about names and variables. *Unrestricted* contexts are simply sets of names since we only have one base type. *Linear contexts* associate a type to endpoints. We use the comma as split/union, overloaded to singletons, and $\cdot$ as the empty context. We extend the Barendregt convention [2] to contexts, so that all entries are distinct. Note that the order in which information is added to a type context does not matter. We denote with $\mathbf{end}(\Delta)$ a (linear) context whose names all have type **end**.

Since we need to determine whether endpoints are used in complementary ways to determine whether processes are well formed, we need to formally define the dual of a type as follows:

$$\overline{?.S} = !.\overline{S} \qquad \overline{!.S} = ?.\overline{S} \qquad \overline{\mathbf{end}} = \mathbf{end}$$

Note that the dual function is partial since it is undefined for the base type.

Typing Rules. Our type system is aimed at maintaining two invariants:

1. No endpoint is used simultaneously by parallel processes;
2. The two endpoints of the same session have dual types.

The first invariant is maintained by linearly splitting type contexts when typing compositions of processes, the second by requiring duality when typing restrictions.

We have two typing judgments: one for values, and one for processes. The typing rules for values are:

$$\text{T-Base}\ \frac{}{\Gamma \vdash_v a : \mathbf{base}} \qquad \text{T-Var}\ \frac{}{\Gamma, l \vdash_v l : \mathbf{base}}$$

The typing rules for processes are as follows:

$$\text{T-Inact}\ \frac{\mathbf{end}(\Delta)}{\Gamma; \Delta \vdash \mathbf{0}} \qquad \text{T-Par}\ \frac{\Gamma; \Delta_1 \vdash P \qquad \Gamma; \Delta_2 \vdash Q}{\Gamma; \Delta_1, \Delta_2 \vdash P \mid Q} \qquad \text{T-Res}\ \frac{\Gamma; (\Delta, x : T, y : \overline{T}) \vdash P}{\Gamma; \Delta \vdash (\boldsymbol{\nu} xy)\ P}$$

$$\text{T-Out}\ \frac{\Gamma \vdash_v v : \mathbf{base} \qquad \Gamma; \Delta, x : T \vdash P}{\Gamma; (\Delta, x : !.T) \vdash x!v.P} \qquad \text{T-IN}\ \frac{(\Gamma, l); (\Delta, x : T) \vdash P}{\Gamma; (\Delta, x : ?.T) \vdash x?(l).P}$$

Note that we do not need a judgment for typing channels, since it is already folded into the T-In and T-Out rules.

Challenge. The objective of this challenge is to prove subject reduction and type safety for our calculus with session types. We start with some lemmata:

Lemma 1 (Weakening).

1. *If* $\Gamma; \Delta \vdash P$, *then* $(\Gamma, l); \Delta \vdash P$.
2. *If* $\Gamma; \Delta \vdash P$, *then* $\Gamma; (\Delta, x : \mathbf{end}) \vdash P$.

Proof. By induction on the given derivations.

Lemma 2 (Strengthening).

1. *If* $(\Gamma, l) \vdash_v v : \mathbf{base}$, *then* $\Gamma \vdash_v v : \mathbf{base}$.
2. *If* $\Gamma; (\Delta, x : T) \vdash P$ *and* $x \notin fn(P)$, *then* $\Gamma; \Delta \vdash P$

Proof.

1. By immediate case analysis on the given derivation.
2. By induction on the derivation of $\Gamma; (\Delta, x : T) \vdash P$.
   T-Inact Since $\mathbf{end}(\Delta)$, we can just reapply the rule without $x : T$.
   T-Par In this case, we have that $\Delta, x : T = \Delta_0, \Delta_1$. By cases on which context x is in, we just apply the induction hypothesis on that context.
   T-Res Without loss of generality, we assume that $x \notin \{y, z\}$, for $P = (\boldsymbol{\nu} yz)\ P'$. Since $\Gamma; (\Delta, y : T_0, z : \overline{T_0}, x : T) \vdash P'$, by induction hypothesis, we have $\Gamma; (\Delta, y : T_0, z : \overline{T_0}) \vdash P'$. Applying again T-Res, we have $\Gamma; \Delta \vdash (\boldsymbol{\nu} yz)\ P'$.
   The remaining cases are analogous.

Lemma 3 (Substitution). *If* $(\Gamma, l); \Delta \vdash P$ *and* $\Gamma \vdash_v a : \mathbf{base}$ *then* $\Gamma; \Delta \vdash P\{a/l\}$.

Proof. By induction on the derivation of $(\Gamma, l); \Delta \vdash P$.

T-Inact Immediate since $\mathbf{end}(\Delta)$.
T-Par For $P = P_0 \mid P_1$, we apply the induction hypothesis on the derivations for P_0 and P_1.
T-Res Immediate by the induction hypothesis.
T-Out Let $P = x!v.P'$. We have that $(\Gamma, l) \vdash_v v : \mathbf{base}$. We have $\Gamma \vdash_v v : \mathbf{base}$ by strengthening, so we build the conclusion with the induction hypothesis and T-Out.

The remaining cases are analogous.

To prove that process equivalence preserves typing, it is convenient to first consider the smallest relation closed under the six axioms of structural congruence, denoted by $\cdot \overset{a}{\equiv} \cdot$:

Lemma 4 (Preservation for $\overset{a}{\equiv}$). *If $P \overset{a}{\equiv} Q$, then $\Gamma; \Delta \vdash P$ iff $\Gamma; \Delta \vdash Q$.*

Proof. By case analysis on the Sc rule applied:

Par-Comm/Assoc By rearranging sub-derivations noting that order does not matter for linear contexts.
Par-Inact Right-to-left by Lemma 2. Vice-versa, by picking T to be **end** and applying Lemma 1, part 2.
Res-Par By case analysis on $x : T$ being linear or **end** and applying weakening and strengthening accordingly.
Res-Inact: By Lemma 1.
Res Noting that order does not matter.

Now, following Sangiorgi and Walker [4], we formalize the compatible equivalence relation induced by $\cdot \overset{a}{\equiv} \cdot$, which we still write as $\cdot \equiv \cdot$ as the smallest relation closed under reflexivity, symmetry, transitivity and the following condition:

$$\textsc{Cong}\ \frac{P \overset{a}{\equiv} Q}{C[P] \equiv C[Q]}$$

Lemma 5 (Preservation for $\equiv$). *If $P \equiv Q$, then $\Gamma; \Delta \vdash P$ iff $\Gamma; \Delta \vdash Q$.*

Proof. By induction on the structure of the derivation of $P \equiv Q$, with an inner induction of the structure of a process context.

Refl Immediate.
Sym By IH.
Trans By two appeals to the IH.
Cong By induction on the structure of C. If the context is a hole, apply Lemma 4. In the step case, apply the IH: for example if C has the form $C' \mid R$, noting that $(C' \mid R)[P]$ is equal to $C'[P] \mid R$ we have $\Gamma; \Delta \vdash (C' \mid R)[P]$ iff $\Gamma; \Delta \vdash (C' \mid R)[Q]$ by rule T-Par and the IH.

Theorem 1 (Subject reduction). *If $\Gamma; \Delta \vdash P$ and $P \to Q$, then $\Gamma; \Delta \vdash Q$.*

Proof. By induction on the derivation of $P \to Q$. The cases R-Par and R-Res follow immediately by IH. Case R-Struct appeals twice to preservation of $\equiv$ (Lemma 5) and to the IH. For R-Com, suppose that T-Res introduces in Δ the assumptions $x : !.U, y : ?.\overline{U}$. Building the only derivation for the hypothesis, we know that $\Delta = \Delta_1, \Delta_2, \Delta_3$ where $\Gamma; \Delta_3 \vdash R$. We also have $\Gamma; (\Delta_1, x : U) \vdash P$, $\mathcal{D}_2$ a proof of $\Gamma, l; (\Delta_2, y : \overline{U}) \vdash Q$ and $\mathcal{V}$ a proof of $\Gamma \vdash_v a :$ **base**. From $\mathcal{D}_2$ and $\mathcal{V}$ we use the substitution lemma 3 to obtain $\Gamma; \Delta_2, y : \overline{U} \vdash Q\{a/l\}$. We then conclude the proof with rules T-Par (twice) and T-Res.

To formulate safety, we need to formally define what we mean by well-formed process. We say that a process P is *prefixed at variable* x if $P \equiv x!v.P'$ or $P \equiv x?(l).P'$ for some P'. A process P is then *well formed* if, for every P_1,

P_2, and R such that $P \equiv (\boldsymbol{\nu} x_1 y_1)\ \ldots (\boldsymbol{\nu} x_n y_n)\ (P_1 \mid P_2 \mid R)$, with $n \geq 0$, it holds that, if P_1 is prefixed at x_1 and P_2 is prefixed at y_1 (or vice versa), then $P_1 \mid P_2 \equiv x_1!a.P_1' \mid y_1?(l).P_2'$, for some P_1' and P_2'.

Note that well-formed processes do not necessarily reduce. For example, the process

$$(\boldsymbol{\nu} x_1 y_1)\ (\boldsymbol{\nu} x_2 y_2)\ (x_1!a.y_2?(l).\mathbf{0} \mid y_2!x_2.y_1?(l).\mathbf{0})$$

is well formed but also irreducible.

Theorem 2 (Type safety). *If $\Gamma; \cdot \vdash P$, then P is well-formed.*

Proof. In order to prove that P is well-formed, let us consider any process of the form $(\boldsymbol{\nu} x_1 y_1)\ \ldots (\boldsymbol{\nu} x_n y_n)\ (P_1 \mid P_2 \mid R)$ that is structurally congruent to P. Clearly, by Lemma 5, well-typedness must be preserved by structural congruence. Moreover, assume that P_1 is prefixed at x_1 and P_2 is prefixed at y_1 such that $P_1 \equiv x_1!v.P_1'$ (the opposite case proceeds similarly). We need to show that $P_2 \equiv y_1?(l).P_2'$. This can be easily done by contradiction. In fact, if $P_2 \equiv y_1!v.P_2'$ then the typing rule for restriction would be violated since the type of x_1 and y_1 cannot be dual.

Corollary 1. *If $\Gamma; \cdot \vdash P$ and $P \to Q$, then Q is well formed.*

1.3 Challenge: Name Passing and Scope Extrusion

This challenge formalises a proof that requires explicit scope extrusion. Scope extrusion is the notion that a process can send restricted names to another process, as long as the restriction can safely be "extruded" (*i.e.*, expanded) to include the receiving process. This, for instance, allows a process to set up a private connection by sending a restricted name to another process, then using this name for further communication. The key issue of this challenge is reasoning about names that are "in the process" of being scope-extruded, which often presents difficulties for the mechanisation of binders.

Reasoning about scope extrusion explicitly can sometimes be avoided by introducing a structural congruence rule into the semantics, but doing this means we lose information about the scope when reasoning about the semantics. Explicitly reasoning about scope extrusion is necessary to describe, *e.g.*, runtime monitors and compositions of systems.

The setting for this challenge is a "classic" untyped π-calculus, where (unlike the calculi in the other challenges) names can be sent and received, and bound by input constructs (similarly to variables in the other calculi). We define two different semantics for our system: one that avoids explicit reasoning about scope extrusion, and one that does not. The objective of this challenge is to prove that the two semantics are equivalent up to structural congruence.

Syntax. The syntax of processes is given by:

$$P, Q \quad ::= \quad \mathbf{0} \quad \mid \quad (P \mid Q) \quad \mid \quad x!y.P \quad \mid \quad x?(y).P \quad \mid \quad (\boldsymbol{\nu} x)\ P$$

The process $x!y.P$ is an *output*, which can send the name y via x, then continue as P. The process $x?(y).P$ is an *input*, which can receive a name via x, then continue as P with the received name substituted for y. The input operator thus binds the name y in P. Note that the scope of a restriction may change when processes interact. Namely, a restricted name may be sent *outside* of its scope. Note that there is no recursion or replication in the syntax, and thus no infinite behaviours can be expressed. This simplifies the theory and is orthogonal to the concept of scope extrusion.

Reduction Semantics. The first semantics is an operational reduction semantics, which avoids reasoning explicitly about scope extrusion by way of a structural congruence rule. *Structural congruence* is the smallest congruence relation that satisfies the following axioms:

$$\textsc{Sc-Par-Assoc}\ \frac{}{P \mid (Q \mid R) \equiv (P \mid Q) \mid R} \qquad \textsc{Sc-Par-Comm}\ \frac{}{P \mid Q \equiv Q \mid P} \qquad \textsc{Sc-Par-Inact}\ \frac{}{P \mid \mathbf{0} \equiv P}$$

$$\textsc{Sc-Res-Par}\ \frac{}{(\boldsymbol{\nu} x)\ P \mid Q \equiv (\boldsymbol{\nu} x)\ (P \mid Q)} \qquad \textsc{Sc-Res-Inact}\ \frac{}{(\boldsymbol{\nu} x)\ \mathbf{0} \equiv \mathbf{0}} \qquad \textsc{Sc-Res}\ \frac{}{(\boldsymbol{\nu} x)\ (\boldsymbol{\nu} y)\ P \equiv (\boldsymbol{\nu} y)\ (\boldsymbol{\nu} x)\ P}$$

The operational semantics is defined as the following relation on processes:

$$\textsc{R-Com}\ \frac{}{x!y.P \mid x?(z).Q \to P \mid Q\{y/z\}} \qquad \textsc{R-Res}\ \frac{P \to Q}{(\boldsymbol{\nu} x)\ P \to (\boldsymbol{\nu} x)\ Q} \qquad \textsc{R-Par}\ \frac{P \to Q}{P \mid R \to Q \mid R}$$

$$\textsc{R-Struct}\ \frac{P \equiv P' \qquad P' \to Q' \qquad Q \equiv Q'}{P \to Q}$$

Note that there is no rule for inferring an action of an input or output process except those that match the input/output capability. Note also that due to rule R-Com, the process $x?(z).P$ can receive *any* name. Finally, note that rule R-Struct allows for applying the structural congruence both before and after the reduction: this makes the reduction relation closed under structural congruence.

Transition System Semantics. The second semantics of the system describe the actions that the system can perform by defining a labelled transition relation on processes. The transitions are labelled by *actions*, the syntax of which is:

$$\alpha \quad ::= \quad x!y \quad \mid \quad x?y \quad \mid \quad x!(y) \quad \mid \quad \tau$$

The *free output action* $x!y$ is sending the name y via x. The *input action* $x?y$ is receiving the name y via x. The *bound output action* $x!(y)$ is sending a fresh name y via x. The *internal action* τ is performing internal communication.

We extend the notion of free and bound occurrences with $\mathrm{fn}(\alpha)$ to denote the set of names that occur free in the action α and $\mathrm{bn}(\alpha)$ to denote the set of names that occur bound in the action α. In the free output action $x!y$ and the input action $x?y$, both x and y are free names. In the bound output action $x!(y)$, x is a free name, while y is a bound name. We also use the notation $\mathrm{n}(\alpha)$ to denote the union of $\mathrm{fn}(\alpha)$ and $\mathrm{bn}(\alpha)$, i.e. the set of all names that occur in the action α.

The transition relation is then defined by the following rules:

$$\textsc{Out}\ \frac{}{x!y.P \xrightarrow{x!y} P} \qquad \textsc{In}\ \frac{}{x?(z).P \xrightarrow{x?y} P\{y/z\}} \qquad \textsc{Par-L}\ \frac{P \xrightarrow{\alpha} P' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(Q) = \emptyset}{P \mid Q \xrightarrow{\alpha} P' \mid Q}$$

$$\textsc{Par-R}\ \frac{Q \xrightarrow{\alpha} Q' \qquad \mathrm{bn}(\alpha) \cap \mathrm{fn}(P) = \emptyset}{P \mid Q \xrightarrow{\alpha} P \mid Q'} \qquad \textsc{Comm-L}\ \frac{P \xrightarrow{x!y} P' \qquad Q \xrightarrow{x?y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\textsc{Comm-R}\ \frac{P \xrightarrow{x?y} P' \qquad Q \xrightarrow{x!y} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \textsc{Close-L}\ \frac{P \xrightarrow{x!(z)} P' \qquad Q \xrightarrow{x?z} Q' \qquad z \notin \mathrm{fn}(Q)}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} z)\ P' \mid Q'}$$

$$\textsc{Open}\ \frac{P \xrightarrow{x!z} P' \qquad z \neq x}{(\boldsymbol{\nu} z)\ P \xrightarrow{x!(z)} P'} \qquad \textsc{Close-R}\ \frac{P \xrightarrow{x?z} P' \qquad Q \xrightarrow{x!(z)} Q' \qquad z \notin \mathrm{fn}(P)}{P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu} z)\ P' \mid Q'}$$

$$\textsc{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad z \notin \mathrm{n}(\alpha)}{(\boldsymbol{\nu} z)\ P \xrightarrow{\alpha} (\boldsymbol{\nu} z)\ P'}$$

Note that there is no rule for inferring transitions from $\mathbf{0}$, and that there is no rule for inferring an action of an input or output process except those that match the input/output capability. Note also that due to rule In, the process $x?(z).P$ can receive *any* name.

We keep the convention that bound names of any processes or actions are chosen to be different from the names that occur free in any other entities under consideration, such as processes, actions, substitutions, and sets of names. The convention has one exception, namely that in the transition $P \xrightarrow{x!(z)} Q$, the name z (which occurs bound in P and the action $x!(z)$) may occur free in Q. Without this exception it would be impossible to express scope extrusion.

Challenge. As in the linearity challence, we first consider the smallest relation closed under the six axioms of structural congruence, denoted by $\cdot \overset{a}{\equiv} \cdot$:

Lemma 6. *If $P \overset{a}{\equiv} Q$ and $P \xrightarrow{\alpha} P'$, then for some Q' we have $Q \xrightarrow{\alpha} Q'$ and $P' \overset{a}{\equiv} Q'$.*

Proof (Sketch). By case analysis on the first derivation.

Lemma 7. *If $P \equiv Q$ and $P \xrightarrow{\alpha} P'$, then for some Q' we have $Q \xrightarrow{\alpha} Q'$ and $P' \equiv Q'$.*

Proof (Sketch). By induction on the structure of the derivation of $P \equiv Q$, with an inner induction on the structure of a process context and an appeal to Lemma 6.

For the proof of the challenge theorem, we introduce the notion of a *normalized derivation* of a reduction $P \to Q$, which is of the following form. The first rule applied is R-Com. The derivation continues with an application of R-Par, followed by zero or more applications of R-Res. The last rule is an application of R-Struct.

Lemma 8. *Every reduction has a normalized derivation.*

Proof (Sketch). To obtain a normalized derivation from an arbitrary derivation we will need to check that rules R-Com, R-Par and R-Res commute with R-Struct, and that two applications of R-Struct can be combined into one.

Lemma 9. *If $P \to Q$, then there are $x, y, z, z_1, \ldots, z_n, R_1, R_2$, and S such that*

$$P \equiv (\boldsymbol{\nu} z_1)\ \ldots (\boldsymbol{\nu} z_n)\ ((x!y.R_1 \mid x?(z).R_2) \mid S)$$
$$Q \equiv (\boldsymbol{\nu} z_1)\ \ldots (\boldsymbol{\nu} z_n)\ ((R_1 \mid R_2\{y/z\}) \mid S)$$

Proof. Follows immediately from lemma 8 and the shape of a normalized derivation.

The objective of this challenge is to prove the following theorems, which together show the equivalence between the reduction semantics and the transition system semantics up to structural congruence. The first of the theorems involves reasoning about scope extrusion more directly than the other, and if time does not permit proving both of the theorems, theorem 3 should be proven first.

Theorem 3. *$P \xrightarrow{\tau} Q$ implies $P \to Q$.*

Proof (Sketch). The proof is by induction on the inference of $P \xrightarrow{\tau} Q$ using the following lemmata:

1. if $Q \xrightarrow{x!y} Q'$ then $Q \equiv (\boldsymbol{\nu} w_1) \ldots (\boldsymbol{\nu} w_n)\ (x!y.R \mid S)$ and $Q' \equiv (\boldsymbol{\nu} w_1) \ldots (\boldsymbol{\nu} w_n)\ (R \mid S)$ where $x, y \notin \{w_1, \ldots, w_n\}$.
2. if $Q \xrightarrow{x!(z)} Q'$ then $Q \equiv (\boldsymbol{\nu} z)\ (\boldsymbol{\nu} w_1)\ \ldots (\boldsymbol{\nu} w_n)\ (x!z.R \mid S)$ and $Q' \equiv (\boldsymbol{\nu} w_1) \ldots (\boldsymbol{\nu} w_n)\ (R \mid S)$ where $x \notin \{z, w_1, \ldots, w_n\}$.
3. if $Q \xrightarrow{x?y} Q'$ then $Q \equiv (\boldsymbol{\nu} w_1) \ldots (\boldsymbol{\nu} w_n)\ (x?(z).R \mid S)$ and $Q' \equiv (\boldsymbol{\nu} w_1) \ldots (\boldsymbol{\nu} w_n)\ (R\{y/z\} \mid S)$ where $x \notin \{w_1, \ldots, w_n\}$.

Theorem 4. *$P \to Q$ implies the existence of a Q' such that $P \xrightarrow{\tau} Q'$ and $Q \equiv Q'$.*

Proof. If $P \to Q$, then by lemma 9, $P \equiv P'$ with

$$P' = (\boldsymbol{\nu} z_1) \ldots (\boldsymbol{\nu} z_n)\ ((x!y.R_1 \mid x?(z).R_2) \mid S)$$

and $Q \equiv Q'$ with

$$Q' = (\boldsymbol{\nu} z_1) \ldots (\boldsymbol{\nu} z_n)\ ((R_1 \mid R_2\{y/z\}) \mid S)\ .$$

We can easily check that $P' \xrightarrow{\tau} Q'$ and so by lemma 7, $P \xrightarrow{\tau} Q'$.

1.4 Challenge: Coinduction and Infinite Processes

This challenge is about the mechanisation of proofs concerning processes with infinite behaviours. This is usually connected to *coinductive* definitions where an infinite structure is defined as the greatest fixed point of a recursive definition. Coinduction is a technique for defining and proving properties of such infinite structures.

For this challenge, we adopt a fragment of the untyped π-calculus that includes process replication. The objective of this challenge is to draw a formal connection between strong barbed congruence and strong barbed bisimilarity. The result establishes that two processes are strong barbed congruent if the processes obtained by applying a finite number of substitutions to them and composing them in parallel with an arbitrary process are strongly barbed bisimilar. The key issue of this challenge is the coinductive reasoning about the infinite behaviours of the replication operator.

Syntax. The syntax of values and processes is given by:

$$\begin{array}{lcl} v, w & ::= & a \mid l \\ P, Q & ::= & \mathbf{0} \mid x!v.P \mid x?(l).P \mid (P \mid Q) \mid (\boldsymbol{\nu} x)\ P \mid\ !P \end{array}$$

The output process $x!v.P$ sends the value v on channel x and continues as P. The intention is that v must be a base value when it is actually sent, and this is enforced in the semantics later on. The input process $x?(l).P$ waits for a base value from channel x and then continues as P with the received value substituted for the variable l. Since replication allows for infinite copies of the process P, processes can dynamically create an infinite number of names during execution.

Semantics. We choose to give a labelled transition system semantics for this challenge.

The transitions are labelled by *actions*, the syntax of which is as follows:

$$\alpha\ ::=\ x!a \mid x?a \mid \tau$$

The *output action* $x!y$ is sending the base value a via x. The *input action* $x?y$ is receiving the base value y via x. The *internal action* τ is performing internal

communication. We use the notation $\mathrm{n}(\alpha)$ to denote the set of names that occur in the action α.

The transition relation is defined by the following rules:

$$\textsc{Out}\ \frac{}{x!a.P \xrightarrow{x!a} P} \qquad \textsc{In}\ \frac{}{x?(l).P \xrightarrow{x?a} P\{a/l\}} \qquad \textsc{Par-L}\ \frac{P \xrightarrow{\alpha} P'}{P \mid Q \xrightarrow{\alpha} P' \mid Q} \qquad \textsc{Par-R}\ \frac{Q \xrightarrow{\alpha} Q'}{P \mid Q \xrightarrow{\alpha} P \mid Q'}$$

$$\textsc{Comm-L}\ \frac{P \xrightarrow{x!a} P' \qquad Q \xrightarrow{x?a} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'} \qquad \textsc{Comm-R}\ \frac{P \xrightarrow{x?a} P' \qquad Q \xrightarrow{x!a} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$$

$$\textsc{Res}\ \frac{P \xrightarrow{\alpha} P' \qquad x \notin \mathrm{n}(\alpha)}{(\boldsymbol{\nu} x)\ P \xrightarrow{\alpha} (\boldsymbol{\nu} x)\ P'} \qquad \textsc{Rep}\ \frac{P \xrightarrow{\alpha} P'}{!P \xrightarrow{\alpha} P' \mid !P}$$

Note that there is no rule for inferring transitions from $\mathbf{0}$, and that there is no rule for inferring an action of an input or output process except those that match the input/output capability. Note also that due to rule In, the process $x?(l).P$ can receive *any* base value. Since the rule Out only applies to base values, there is no way to send a variable.

Strong Barbed Bisimilarity. Bisimilarity is a notion of equivalence for processes and builds on a notion of *observables*, i.e., what we can externally observe from the semantics of a process. If we allowed ourselves only to observe internal transitions (i.e., observe that a process is internally performing a step of computation) we would relate either too few processes (in the strong case where we relate only processes with exactly the same number of internal transitions) or every process (in the weak case where we relate processes with any amount of internal transitions). As a result, we must allow ourselves to observe more than just internal transitions, and we choose to describe a process's observables as the names it might use for sending and receiving.

To this end, we define the *observability predicate* $P \downarrow_\mu$ as follows:

$$P \downarrow_{x?} \quad \text{if } P \text{ can perform an input action via } x.$$
$$P \downarrow_{x!} \quad \text{if } P \text{ can perform an output action via } x.$$

A symmetric relation $\mathcal{R}$ is a *strong barbed bisimulation* if $(P, Q) \in \mathcal{R}$ implies

$$P \downarrow_\mu \text{ implies } Q \downarrow_\mu \tag{1}$$
$$P \xrightarrow{\tau} P' \text{ implies } Q \xrightarrow{\tau} Q' \text{ and } (P', Q') \in \mathcal{R} \tag{2}$$

Two processes are said to be *strong barbed bisimilar*, written $P \mathrel{\dot{\sim}} Q$, if there exists a strong barbed bisimulation $\mathcal{R}$ such that $(P, Q) \in \mathcal{R}$. Note that *strong barbed bisimilarity* $\dot{\sim}$ is the largest strong barbed bisimulation. Also, since our processes have potentially infinite behaviours, bisimilarity cannot be defined inductively since it is the largest strong barbed bisimulation.

Theorem 5. $\dot{\sim}$ *is an equivalence relation.*

Proof. We prove the three properties separately:

- Reflexivity is straightforward: for any P, we need to show that $P \dot{\sim} P$. In order to do so, we choose the identity relation and prove that it is a strong barbed bisimulation. Condition 1 follows trivially by definition. Condition 2 follows coinductively since we must always reach identical pairs.
- Symmetry follows immediately by definition.
- For transitivity, we need to prove that if $P \dot{\sim} Q$ and $Q \dot{\sim} R$ then $P \dot{\sim} R$. In order to do so, we prove that the relation $\mathcal{R} = \{(P, R) \mid \exists Q \text{ such that } P \dot{\sim} Q \wedge Q \dot{\sim} R\}$ is a strong barbed bisimulation. Let us assume that $(P, R) \in \mathcal{R}$. Hence, there exists a Q such that $P \dot{\sim} Q$ and $Q \dot{\sim} R$. Clearly, if $P \downarrow_\mu$ then, by $P \dot{\sim} Q$, $Q \downarrow_\mu$. And, by $Q \dot{\sim} R$, $R \downarrow_\mu$. Moreover, if $P \xrightarrow{\tau} P'$ there exists Q' such that $Q \xrightarrow{\tau} Q'$ and $P' \dot{\sim} Q'$. And also, $R \xrightarrow{\tau} R'$ with $Q' \dot{\sim} R'$. Finally, by definition of $\mathcal{R}$, $(P', R') \in \mathcal{R}$.

Unfortunately, strong barbed bisimilarity is not a good process equivalence since it is not a congruence, hence it does not allow for substituting a process with an equivalent one in any context. For instance, the processes $x!a.y!b.\mathbf{0}$ and $x!a.\mathbf{0}$ are strong barbed bisimilar, i.e., $x!a.y!b.\mathbf{0} \dot{\sim} x!a.\mathbf{0}$. This is because $x!$ is the only observable in both processes and they cannot perform a τ-action. However, in the context $C = [\cdot] \mid x?(l).\mathbf{0}$, the relation no longer holds: in fact, $x!a.y!b.\mathbf{0} \mid x?(l).\mathbf{0} \not\dot{\sim} x!a.\mathbf{0} \mid x?(l).\mathbf{0}$ because the left process can perform a τ-action such that $y!$ becomes observable, whereas the right process cannot.

Strong Barbed Congruence. In order to detect cases like the one above, we need to restrict strong barbed bisimilarity so that it becomes a congruence, i.e., we have to consider the environment in which processes may be placed.

We say that two processes P and Q are *strong barbed congruent*, written $P \simeq^c Q$, if $C[P] \dot{\sim} C[Q]$ for every context C.

Lemma 10. $\simeq^c$ *is the largest congruence included in* $\dot{\sim}$.

Proof. We first prove that $\simeq^c$ is indeed a congruence, i.e. it is an equivalence relation that is preserved by all contexts. Proving that $\simeq^c$ is an equivalence is easy; to prove that $\simeq^c$ is preserved by all contexts, we show that $\forall C : P \simeq^c Q$ implies $C[P] \simeq^c C[Q]$, by structural induction on the context C.

To prove that $\simeq^c$ is the largest congruence included in $\dot{\sim}$, we show that for any congruence $\mathcal{S} \subseteq \dot{\sim}$ we have $\mathcal{S} \subseteq \simeq^c$. Take any P, Q such that $P \mathcal{S} Q$ (hence, $P \dot{\sim} Q$): since S is a congruence by hypothesis, this implies $\forall C : C[P] \mathcal{S} C[Q]$ (hence, $C[P] \dot{\sim} C[Q]$). Therefore, by the definition of $\simeq^c$, we have $P \simeq^c Q$, from which we conclude $\mathcal{S} \subseteq \simeq^c$.

Challenge. The objective of this challenge is to prove a theorem that shows that making strong barbed bisimilarity sensitive to substitution and parallel composition is enough to show strong barbed congruence. To prove the theorem, we will use an *up-to technique*, utilizing the following definition and lemma. A relation $\mathcal{S}$ is called a *strong barbed bisimulation up to* $\dot{\sim}$ if, whenever $(P, Q) \in \mathcal{S}$, the following conditions hold:

1. $P \downarrow_\mu$ if and only if $Q \downarrow_\mu$.
2. if $P \xrightarrow{\tau} P'$ then $Q \xrightarrow{\tau} Q'$ for some Q' with $P' \mathrel{\dot{\sim}\mathcal{S}\dot{\sim}} Q'$.
3. if $Q \xrightarrow{\tau} Q'$ then $P \xrightarrow{\tau} P'$ for some P' with $P' \mathrel{\dot{\sim}\mathcal{S}\dot{\sim}} Q'$.

Lemma 11. *If $\mathcal{S}$ is a strong barbed bisimulation up to $\dot{\sim}$, $(P, Q) \in \mathcal{S}$ implies $P \mathrel{\dot{\sim}} Q$.*

Proof. We check that $\mathrel{\dot{\sim}\mathcal{S}\dot{\sim}}$ is a strong barbed bisimulation and is thus included in $\dot{\sim}$.

Theorem 6. *$P \simeq^c Q$ if, for any process R and substitution σ, $P\sigma \mid R \mathrel{\dot{\sim}} Q\sigma \mid R$.*

Proof. Since $\simeq^c$ is the largest congruence included in $\dot{\sim}$, it suffices to show that if $P\sigma \mid R \mathrel{\dot{\sim}} Q\sigma \mid R$ for any R and σ, then $C[P]\sigma \mid R \mathrel{\dot{\sim}} C[Q]\sigma \mid R$ for any C, R and σ. We proceed by induction on C.

$C = x?(z).C'$ Let $\mathcal{S} = \{(C[P]\sigma \mid R, C[Q]\sigma \mid R) \mid R \text{ and } \sigma \text{ arbitrary}\} \cup \dot{\sim}$. We can easily check that $\mathcal{S}$ is a strong barbed bisimulation, noting that $\dot{\sim}$ is preserved by restriction and is contained in $\mathcal{S}$.

$C = C' \mid S$ Then by the induction hypothesis,

$$C[P]\sigma \mid R \mathrel{\dot{\sim}} C'[P]\sigma \mid (S\sigma \mid R) \mathrel{\dot{\sim}} C'[Q]\sigma \mid (S\sigma \mid R) \mathrel{\dot{\sim}} C[Q]\sigma \mid R$$

for any R and σ.

$C = (\boldsymbol{\nu} z)\ C'$ Then by the induction hypothesis we have $C'[P]\sigma \mid R \mathrel{\dot{\sim}} C'[Q]\sigma \mid R$ for any R and σ. Without loss of generality, we assume that $z \notin \mathrm{fn}(R) \cup \mathrm{n}(\sigma)$. Then, using that $\dot{\sim}$ is preserved by restriction, we have

$$C[P]\sigma \mid R \mathrel{\dot{\sim}} (\boldsymbol{\nu} z)\ (C'[P]\sigma \mid R) \mathrel{\dot{\sim}} (\boldsymbol{\nu} z)\ (C'[Q]\sigma \mid R) \mathrel{\dot{\sim}} C[Q]\sigma \mid R$$

$C = !C'$ Let $\mathcal{S} = \{(C[P]\sigma \mid R, C[Q]\sigma \mid R) \mid R \text{ and } \sigma \text{ arbitrary}\}$. Using lemma 11, it suffices to show that $\mathcal{S}$ is a strong barbed bisimulation up to $\dot{\sim}$. To this end, let

$$A = C'[P]\sigma, \qquad A' = C[P]\sigma$$
$$B = C'[Q]\sigma, \qquad B' = C[Q]\sigma,$$

noting that $A' = !A$ and $B' = !B$.
Suppose $R \mid A' \xrightarrow{\tau} S$ for some S. Then we can show by a case analysis on the derivation of this transition that there exists a T such that $R \mid (A \mid A) \xrightarrow{\tau} T$ and $S \mathrel{\dot{\sim}} T \mid A'$. Using the induction hypothesis twice, we note that $R \mid B' \mathrel{\dot{\sim}} R \mid (A \mid A) \mid B'$. Since by rule Par-L, $R \mid (A \mid A) \mid B' \xrightarrow{\tau} T \mid B'$, there must thus exist a U such that $R \mid B' \xrightarrow{\tau} U$, $U \mathrel{\dot{\sim}} T \mid B'$ and $S \mathrel{\dot{\sim}\mathcal{S}\dot{\sim}} U$ as required. The proof for $R \mid B' \xrightarrow{\tau} S$ is analogous.

The remaining cases are similar.

References

1. Andrew Barber. Dual intuitionistic linear logic. Technical Report ECS-LFCS-96-347, University of Edinburgh, 1996.
2. Hendrik Pieter Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundation of Mathematics*. North-Holland, 2 edition, 1984.
3. Robin Milner. *Communication and Concurrency*. Prentice-Hall, USA, 1989.
4. Davide Sangiorgi and David Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, USA, 2001.
5. Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.