Acta Informatica manuscript No.
(will be inserted by the editor)

Session-Based Concurrency, Declaratively

Mauricio Cano · Hugo A. López · Jorge A. Pérez · Camilo Rueda

Received: date / Accepted: date

Abstract *Session-based concurrency* is a type-based approach to the analysis of message-passing programs. These programs may be specified in an *operational* or *declarative* style: the former defines how interactions are properly structured; the latter defines governing conditions for correct interactions. In this paper, we study rigorous relationships between operational and declarative models of session-based concurrency. We develop a correct encoding of session π-calculus processes into the linear concurrent constraint calculus (`lcc`), a declarative model of concurrency based on partial information (constraints). We exploit session types to ensure that our encoding satisfies precise correctness properties and that it offers a sound basis on which operational and declarative requirements can be jointly specified and reasoned about. We demonstrate the applicability of our results by using our encoding in the specification of realistic communication patterns with time and contextual information.

Cano and Pérez have been partially supported by the Netherlands Organization for Scientific Research (NWO) under the VIDI Project No. 016.Vidi.189.046 (Unifying Correctness for Communicating Software).
López is partially supported by the Innovation Fund Denmark project Ecoknow.org (705000034A) and the European Union Marie Sklodowska Curie grant agreement BehAPI No.778233.
Rueda is partially supported by Colciencias, ECOS-NORD project FACTS (C19M03).

M. Cano
University of Groningen
E-mail: m.a.cano.grijalba@gmail.com

H. A. López
University of Copenhagen and DCR Solutions A/S
E-mail: lopez@di.ku.dk

J. A. Pérez
University of Groningen and CWI, Amsterdam
E-mail: j.a.perez@rug.nl

C. Rueda
Pontificia Universidad Javeriana-Cali
E-mail: crueda@javerianacali.edu.co

1 Introduction

This paper addresses the problem of relating two distinct models of concurrent processes: one of them, the session π-calculus [45] (π, in the following), is *operational*; the other one, the linear concurrent constraint calculus [22,27] (`lcc`, in the following), is *declarative*. Our interest in these two models stems from the analysis of *message-passing software systems*, which are best specified by combining operational features (present in models such as π) and declarative features (present in models such as `lcc`). In this work, we aim at results of *relative expressiveness*, which explain how to faithfully encode programs in one model into programs in some other model [39,40]. Concretely, we are interested in *translations* in which π and `lcc` are, respectively, *source* and *target languages*; a key common trait supporting expressiveness results between π and `lcc` is *linearity*, in the sense of Girard's linear logic, the logic of consumable resources [24].

The process language π falls within *session-based concurrency*, a type-based approach to the analysis of message-passing programs. In this approach, protocols are organized into basic units called *sessions*; interaction patterns are abstracted as *session types* [29], against which specifications may be checked. A session connects exactly two partners; session types ensure that interactions always occur in matching pairs: when one partner sends, the other receives; when one partner offers a selection, the other chooses; when a partner closes the session, the other acknowledges. When specifications are given in the π-calculus [33,34], we obtain processes interacting along channels to implement session protocols. Sessions thus involve concurrency, mobility, and resource-awareness: a session is a sequence of deterministic interactions on *linear channels*, to be used exactly once.

In specifying message-passing programs, operational and declarative features are complementary: while the former describe *how* a message-passing program is implemented, the latter describe *what* are the (least) conditions that govern a program's correct behavior. Although languages based on the π-calculus can conveniently specify mobile, point-to-point communications, they do not satisfactorily express other kinds of requirements that influence protocol interactions and/or communicating partners—in particular, *partial and contextual information* can be unnatural or difficult to express in them.

To address this shortcoming, extensions of name-passing calculi such as, e.g., [21,11,20, 5,14], have been developed: they typically add declarative features based on *constraints* (or *assertions*), i.e., logical conditions that specify and influence process behavior. Interestingly, several of these extensions are inspired by *concurrent constraint programming* [43] (`ccp`, in the following), a model of concurrency in which constraints (but also other forms of partial information) are a primitive concept. Process languages based on `ccp` are appealing because they are simple, rest upon solid foundations, and are very expressive. Indeed, `ccp` languages such as `lcc` and `utcc` [37] can represent mobility as in the π-calculus; such representations, however, tend to be unpractical for reasoning about message-passing programs.

In our view, this current state of affairs begs for a *unifying* account of operational and declarative approaches to session-based concurrency. We envision a *declarative basis* for session-based concurrency in which constructs from operational models (such as π) are given correct, low-level implementations in declarative models (such as `lcc`). Such implementations can then be freely used as "macros" in larger declarative specifications, in which requirements related to partial and contextual information can be cleanly expressed. In this way, existing operational and declarative languages (and their analysis techniques) can be articulated at appropriate abstraction levels. An indispensable step towards this vision is developing rigorous ways of compiling operational languages into declarative ones. This is the main technical challenge in this paper.

In line with this challenge, our previous work [31] formally related the session π-calculus in [29] and `utcc` using an *encoding*, i.e., a language translation that satisfies certain *encodability criteria* [26]. Although this encoding already enables us to reason about message-passing specifications from a declarative standpoint, it presents some important limitations. First, the key rôle of *linearity* and *type-based correctness* in session-based concurrency is not explicit when encoding session π-calculus processes in `utcc`. Also, because `utcc` is a *deterministic* language, the encoding in [31] can only translate deterministic session processes, and so it rules out useful forms of non-determinism that naturally arise in session-based concurrency.

To address these limitations within a unified account for session-based concurrency, here we develop an encoding of π into `lcc`. Unlike `utcc`, `lcc` treats *constraints as linear resources* that can be used exactly once. Our main discovery is that `lcc` with its explicit treatment of linearity is a much better match for interactions in session-based concurrency than `utcc`. This is made formal by the tight correspondences between source processes in π and target processes in `lcc`. Unlike `utcc`, `lcc` is a non-deterministic language. Hence, by using `lcc` as target language, our encoding can translate π processes that cannot be translated by the encoding in [31], such as, e.g., a process specifying a session protocol in which a client can non-deterministically interact with multiple servers (cf. Ex. 3).

Summarizing, this paper develops the following contributions:

- A translation from π into `lcc` (§ 4). By using `lcc` as target language, our translation supports linearity and non-determinism, as essential in session-based concurrency.
- A study of the conditions under which the session types by Vasconcelos [45] enable us to correctly translate π processes into `lcc` (§ 3.1). We use these conditions to prove that our translation is a *valid encoding*: it satisfies Gorla's encodability criteria [26], in particular operational correspondence.
- Extended examples that showcase how processes resulting from our encoding can be used as macros in declarative specifications (§ 5). By exploiting a general strategy that uses encoded processes as code snippets, we specify in `lcc` two of the communication patterns with time in [36].

The rest of this paper is structured as follows. § 2 describes session-based concurrency and introduces the key ideas in our approach. § 3 presents required background on relative expressiveness, π, and `lcc`. In particular, § 3.1 presents a variant of the session types in [45] that is crucial to establish correctness for our translation. § 4 presents the translation of π into `lcc` and establishes its correctness. § 5 develops the extended examples. We close by discussing related work (§ 6) and giving some concluding remarks (§ 7). The appendices contain additional examples and omitted proofs.

This paper builds upon results first reported in the conference paper [17]. Such results include (i) an encoding of π into `lcc`, as well as (ii) an encoding of an extension of π with *session establishment* into a variant of `lcc`. Here we offer a revised, extended presentation of the results on (i), for which we present stronger correspondences, full technical details, and additional examples. This focus allows us to keep presentation compact; a detailed description of the results related to (ii) can be found in Cano's PhD thesis [15].

2 Overview of Key Ideas

We informally illustrate our approach and main results. We use π and `lcc` processes, whose precise syntax and semantics will be introduced in the following section.

Session-Based Concurrency. Consider a simple protocol between a client and a shop:

1. The client sends a description of an item that she wishes to buy to the shop.
2. The shop replies with the price of the item and offers two options to the client: to buy the item or to close the transaction.
3. Depending on the price, the client may choose to purchase the item or to end the transaction.

Here is a session type that specifies this protocol from the client's perspective:

$$S = !\mathsf{item}.\, ?\mathsf{price}.\, \oplus \{\mathit{buy} : !\mathsf{ccard}.\, ?\mathsf{invoice}.\, \mathtt{end}\,,\ \mathit{quit} : !\mathsf{bye}.\, \mathtt{end}\}$$

Type S says that the output of a value of type item (denoted !item) should be followed by the input of a value of type price (denoted ?price). These two actions should precede the selection (internal choice, denoted $\oplus$) between two different behaviors distinguished by *labels buy* and *quit*: in the first behavior, the client sends a value of type ccard, then receives a value of type invoice, and then closes the protocol (end denotes the concluded protocol); in the second behavior, the client emits a value of type bye and closes the session.

From the shop's perspective, we would expect a protocol that is *complementary* to S:

$$T = ?\,\mathsf{item}.\, !\mathsf{price}.\, \&\, \{\mathit{buy} : ?\mathsf{ccard}.\, !\mathsf{invoice}.\, \mathtt{end}\,,\ \mathit{quit} : ?\mathsf{bye}.\, \mathtt{end}\}$$

After receiving a value of type item, the shop sends a value of type price back to the client. Using external choice (denoted &), the shop then offers two behaviors to the client, identified by labels *buy* and *quit*. The complementarity between types such as S and T is formalized by *session type duality* (see, e.g., [23]). The intent is that implementations derived from dual session types will respect their (complementary) protocols at run-time, avoiding communication mismatches and other insidious errors.

We illustrate the way in which session types relate to π processes. We write $x\langle v\rangle.P$ and $x(y).P$ to denote output and input along name x, with continuation P. Also, given a finite set I, we write $x \triangleright \{\mathit{lab}_i : P_i\}_{i\in I}$ to denote the offer of labeled processes $P_1, P_2, \ldots$ along name x; dually, $x \triangleleft \mathit{lab}.\, P$ denotes the selection of a label *lab* along x. Moreover, process $b?\, P : Q$ denotes a conditional expression which executes P or Q depending on boolean b. Process P_x below is a possible implementation of type S along x:

$$P_x = x\langle \mathsf{book}\rangle.\, x(z).\, (z \leq 20)?\, x \triangleleft \mathit{buy}.\, x\langle 5406\rangle.\, x(\mathit{inv}).\mathbf{0} : x \triangleleft \mathit{quit}.\, x\langle \mathit{end}\rangle.\, \mathbf{0} \qquad (1)$$

$$P_y = y(w).y\langle \mathsf{price}(w)\rangle.y \triangleright \{\mathit{buy} : y(w').y\langle \mathsf{invoice}\rangle.\mathbf{0}, \mathit{quit} : y(w'').\mathbf{0}\} \qquad (2)$$

Process P_x uses a conditional to implement the decision of which option offered by the shop is chosen: the purchase will take place only if the item (a book) is within a $20 budget. We assume that *end* is a value of type bye. Similarly, P_y is a process that implements the shop's intended protocol along y: it first expects a petition for an item (w), and after that returns the item's current price. Then, it offers the buyer a (labeled) choice: either to buy the item or to quit the transaction.

Sessions with Declarative Conditions. The session-based calculus π is a language with point-to-point, synchronous communication. Hence, π processes can appropriately describe protocol actions, but can be less adequate to express contextual conditions on partners and their interactions, which are usually hard to know and predict. Consider a variation of the above protocol, in which the last step is specified as follows:

3'. Depending on the item's price *and* whether the purchase occurs in a given time interval (say, a discount period), the client may either purchase the item or end the transaction.

This kind of time constraints have been studied in [36], where a number of *timed patterns* in communication protocols are identified and analyzed. These patterns add flexibility to specifications by describing the protocol's behavior with respect to external sources (e.g., non-interacting components like clocks and the communication infrastructure). For example, a protocol step such as 3' dictates that communication actions will be executed only within a given time interval. Hence, even though timed requirements do not necessarily enact a communicating action, they may influence interactions between partners.

Timed patterns are instances of *declarative requirements*, which are difficult to express in the (session) π-calculus. Formalizing Step 3' in π is not trivial, because one must necessarily represent time units by using synchronizations—a far-fetched relationship. The (session) π-calculus does not naturally lend itself to specify the combination of operational descriptions of structured interactions (typical of sessions) and declarative requirements (as in, e.g., protocol and workflow specifications). Given this, our aim is to use `lcc` as a unified basis for both operational and declarative requirements in session-based concurrency.

`ccp` *and* `lcc`. `lcc` is based on *concurrent constraint programming* (`ccp`) [43]. In `ccp`, processes interact via a *constraint store* (*store*, in the sequel) by means of *tell* and *ask* operations. Processes may add *constraints* (pieces of partial information) to the store via tell operations; using ask operations processes may query the store about some constraint and, depending on the result of the query, execute a process or suspend. These queries are governed by a *constraint system*, a parametric structure that specifies the entailment relation between constraints. The constraint store thus defines an asynchronous synchronization mechanism; both communication-based and external events can be modeled as constraints in the store.

In `lcc`, tell and ask operations work as follows. Let c and d denote constraints and let $\widetilde{x}$ denote a (possibly empty) vector of variables. While the tell process $\overline{c}$ can be seen as the output of c to the store, the parametric ask operator $\forall\widetilde{x}(d \rightarrow P)$ may be read as: if d can be inferred from the store then P will be executed; hence, P depends on the *guard* d. These parametric ask operators are called *abstractions*. Resource-awareness in `lcc` is crucial: not only the inference consumes the abstraction (i.e., it is *linear*), it may also involve the consumption of constraints in the store as well as substitution of parameters $\widetilde{x}$ in P.

In `lcc`, parametric asks can express name mobility as in the π-calculus [44,27]. That is, the key operational mechanisms of the (session) π-calculus (name communication, scope extrusion) admit (low-level) declarative representations as `lcc` processes.

Our Encoding To illustrate our encoding, let us consider the most elementary computation step in π, which is given by the following reduction rule:

$$(\boldsymbol{\nu} xy)(x\langle v\rangle.P \mid y(z).Q) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q\{^{v}/_{z}\})$$

where v is a value or a variable. This rule specifies the synchronous communication between two complementary *(session) endpoints*, represented by the output process $x\langle v\rangle.P$ and the input process $y(z).Q$. In session-based concurrency, no races in communications between endpoints can occur. We write $(\boldsymbol{\nu} xy)P$ to denote that (bound) variables x and y are reciprocal endpoints for the same session protocol in P.

Our encoding $[\![\cdot]\!]$ translates π processes into `lcc` processes (cf. Fig. 8). The essence of this declarative interpretation is already manifest in the translation of output- and input-prefixed processes:

$$[\![x\langle v\rangle.P]\!] = \overline{\mathsf{snd}(x,v)} \parallel \forall z\big(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]\big)$$
$$[\![x(y).Q]\!] = \forall y,w\big(\mathsf{snd}(w,y) \otimes \{w{:}x\} \to \overline{\mathsf{rcv}(x,y)} \parallel [\![Q]\!]\big)$$

where $\otimes$ denotes multiplicative conjunction in linear logic. We use predicates $\mathsf{snd}(x,v)$ and $\mathsf{rcv}(x,y)$ to represent synchronous communication in π using the asynchronous communication model of `lcc`; also, we use the constraint $\{x{:}z\}$ to indicate that x and z are dual endpoints. These pieces of information are treated as linear resources by `lcc`; this is key to ensure operational correspondence (cf. Thm. 11 and Thm. 12). As we will see, $[\![x\langle v\rangle.P]\!]$ and $[\![x(y).Q]\!]$ synchronize in two steps. First, constraint $\mathsf{snd}(x,v)$ is consumed by the abstraction in $[\![x(y).Q]\!]$, thus enabling $[\![Q]\!]$ and adding $\mathsf{rcv}(x,y)$ to the store. Then, constraint $\mathsf{rcv}(x,y)$ is consumed by the abstraction $\forall z\big(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]\big)$, thus enabling $[\![P]\!]$.

Encoding Correctness using Session Types. To contrast the rôle of linearity in π and in `lcc`, we focus on π processes which are *well-typed* in the type system by Vasconcelos [45]. Type soundness in [45] ensures that well-typed processes never reduce to ill-formed processes that do not respect their intended session protocols.

The type system in [45] offers flexible support for processes with *infinite behavior*, in the form of recursive session types that can be shared among multiple threads. Using recursive session types, the type system in [45] admits π processes with *output races*, i.e., processes in which two or more sub-processes in parallel have output actions on the same variable. Here is a simple example of a process with an output race (on x), which is typable in [45]:

$$R_1 = (\boldsymbol{\nu} xy)(x\langle v_1\rangle.Q_1 \mid x\langle v_2\rangle.Q_2 \mid *\, y(z).Q_3) \tag{3}$$

Even though our translation $[\![\cdot]\!]$ works fine for the set of well-typed π processes as defined in [45], the class of typed processes with output races represents a challenge for proving that the translation $[\![\cdot]\!]$ is correct. We aim at correctness in the sense of Gorla's encodability criteria [26], which define a general and widely used framework for studying relative expressiveness. Roughly speaking, π processes with output races induce ambiguities in the `lcc` processes that are obtained via $[\![\cdot]\!]$. To illustrate this, consider the translation of R_1:

$$\begin{aligned}[\![R_1]\!] = C[&\overline{\mathsf{snd}(x,v_1)} \parallel \forall w_1\big(\mathsf{rcv}(w_1,v_1) \otimes \{x{:}w_1\} \to [\![Q_1]\!]\big) \parallel \\ &\overline{\mathsf{snd}(x,v_2)} \parallel \forall w_2\big(\mathsf{rcv}(w_2,v_2) \otimes \{x{:}w_2\} \to [\![Q_2]\!]\big) \parallel \\ &!\,\forall z,w_3\big(\mathsf{snd}(w_3,z) \otimes \{w_3{:}y\} \to \overline{\mathsf{rcv}(y,z)} \parallel [\![Q_3]\!]\big)]\end{aligned}$$

where context $C[-]$ includes the constraints needed for interaction (i.e., $\{x{:}y\}$) and '!' denotes replication. The ambiguities concern the values involved as objects in the output races. If we assume $v_1 \neq v_2$ then there are no ambiguities and translation correctness as in [26] can be established. Now, if $v_1 = v_2$ then $\mathsf{snd}(x,v_1) = \mathsf{snd}(x,v_2)$, which is problematic for translation correctness (in particular, for proving operational correspondence): once process $!\,\forall z,w_3\big(\mathsf{snd}(w_3,z) \otimes \{w_3{:}y\} \to \overline{\mathsf{rcv}(y,z)} \parallel [\![Q_3]\!]\big)$ consumes either constraint, we cannot precisely determine which continuation should be enabled with constraint $\mathsf{rcv}(x,v_i)$—both $[\![Q_1]\!]$ and $[\![Q_2]\!]$ could be spawned at that point.

To establish translation correctness following the criteria in [26], we narrow down the class of typable π processes in [45] by disallowing processes with output races. To this

end, we introduce a type system in which a recursive type involving an output behavior (a potential output race) can be associated to at most one thread. This is a conservative solution, which allows us to retain useful forms of infinite behavior. Although process R_1 in (3) is not typable in our type system, it allows processes such as

$$(\boldsymbol{\nu} xy)(x\langle v_1\rangle.x\langle v_2\rangle.Q_1 \mid *\, y(z).Q_2)$$

in which the parallel server invocations exhibited by R_1 have been sequentialized.

3 Preliminaries

We start by introducing the source and target languages (§ 3.1 and § 3.2) and the encodability criteria we shall use as reference for establishing translation correctness (§ 3.3).

3.1 A Session π-Calculus Without Output Races (π)

We present the session π-calculus (π) and its associated type system, a specialization of that by Vasconcelos [45] that disallows output races.

3.1.1 Syntax and Semantics

We assume a countably infinite set of *variables* $\mathcal{V}_\pi$, ranged over by $x, y, \ldots$. Channels are represented as pairs of variables, called *co-variables*. Messages are represented by *values*, ranged over by $v, v', u, u', \ldots$ and whose base set is called $\mathcal{U}_\pi$. Values can be both variables and the boolean constants $\mathtt{tt}, \mathtt{ff}$. We also use $l, l', \ldots$ to range over a countably infinite set of *labels*, denoted $\mathcal{B}_\pi$. We write $\widetilde{x}$ to denote a finite sequence of variables $x_1, \ldots, x_n$ with $n \geq 0$ (and similarly for other elements).

Definition 1 **(π)** The set of π processes is defined by grammar below:

$$\begin{aligned} P, Q ::= {} & x\langle v\rangle.P \mid x(y).P \mid x \triangleleft l.P \mid x \triangleright \{l_i : P_i\}_{i\in I} \mid b?\, P : Q \mid *\, x(y).P \\ & \mid (\boldsymbol{\nu} xy)P \mid P \mid Q \mid \mathbf{0} \end{aligned}$$

Process $x\langle v\rangle.P$ sends value v over x and then continues as P; dually, process $x(y).Q$ expects a value v on x that will replace all free occurrences of y in Q. Processes $x \triangleleft l_j.P$ and $x \triangleright \{l_i : Q_i\}_{i\in I}$ define a labeled choice mechanism, with labels indexed by the finite set I: given $j \in I$, the selection process $x \triangleleft l_j.P$ uses x to select l_j from the branching process $x \triangleright \{l_i : Q_i\}_{i\in I}$, thus triggering process Q_j. We assume pairwise distinct labels. The conditional process $v?\, P : Q$ behaves as P if v evaluates to $\mathtt{tt}$; otherwise it behaves as Q. Process $*\, x(y).P$ denotes a replicated input process, which allows us to specify persistent servers. The restriction $(\boldsymbol{\nu} xy)P$ binds together x and y in P, thus indicating that they are two endpoints of the same channel (i.e., the same session). Processes for parallel composition $P \mid Q$ and inaction $\mathbf{0}$ are standard.

We write $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})P$ to stand for $(\boldsymbol{\nu} x_1, \ldots, x_n\, y_1, \ldots, y_n)P$, for some $n \geq 1$. We often write $\prod_{i=1}^{n} P_i$ to stand for $P_1 \mid \cdots \mid P_n$, and refer to the parallel sub-processes of $P_1, \ldots, P_n$ as *threads*.

In $x(y).P$ and $*\, x(y).P$ (resp. $(\boldsymbol{\nu} yz)P$) occurrences of y (resp. y, z) are bound with scope P. The set of free variables of P, denoted $\mathsf{fv}_\pi(P)$, is standardly defined.

$$\lfloor \textsc{Com} \rfloor \quad (\boldsymbol{\nu} xy)(x\langle v\rangle.P \mid y(z).Q \mid R) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q\{^{v}/_{z}\} \mid R)$$

$$\lfloor \textsc{Sel} \rfloor \quad \frac{j \in I}{(\boldsymbol{\nu} xy)(x \lhd l_j.P \mid y \rhd \{l_i{:}Q_i\}_{i\in I} \mid R) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q_j \mid R)}$$

$$\lfloor \textsc{Rep} \rfloor \quad (\boldsymbol{\nu} xy)(x\langle v\rangle.P \mid *\, y(z).Q \mid R) \longrightarrow (\boldsymbol{\nu} xy)(P \mid Q\{^{v}/_{z}\} \mid *\, y(z).Q \mid R)$$

$$\lfloor \textsc{IfT} \rfloor \quad \mathtt{tt}?\, P{:}Q \longrightarrow P \qquad \lfloor \textsc{IfF} \rfloor \quad \mathtt{ff}?\, P{:}Q \longrightarrow Q$$

$$\lfloor \textsc{Str} \rfloor \quad \frac{P \equiv_\pi P' \quad P' \longrightarrow Q' \quad Q' \equiv_\pi Q}{P \longrightarrow Q} \qquad \lfloor \textsc{Par} \rfloor \quad \frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \lfloor \textsc{Res} \rfloor \quad \frac{P \longrightarrow P'}{(\boldsymbol{\nu} x)P \longrightarrow (\boldsymbol{\nu} x)P'}$$

Fig. 1: Reduction relation for π processes.

Remark 1 (Barendregt's variable convention) Throughout the paper, in both π and `lcc`, we shall work up to α-equivalence, as usual; in definitions and proofs we assume that all bound variables are distinct from each other and from all free variables.

The operational semantics for π is given as a *reduction relation* $\longrightarrow$, the smallest relation generated by the rules in Fig. 1. Reduction expresses the computation steps that a process performs on its own. It relies on a *structural congruence* on processes, given below.

Definition 2 (Structural Congruence) The structural congruence relation for π processes is the smallest congruence relation $\equiv_\pi$ that satisfies the following axioms and identifies processes up to renaming of bound variables (i.e., α-conversion, denoted $\equiv_\alpha$).

$$P \mid \mathbf{0} \equiv_\pi P \qquad P \mid Q \equiv_\pi Q \mid P \qquad (P \mid Q) \mid R \equiv_\pi P \mid (Q \mid R)$$
$$(\boldsymbol{\nu} xy)(\boldsymbol{\nu} wz)P \equiv_\pi (\boldsymbol{\nu} wz)(\boldsymbol{\nu} xy)P \qquad (\boldsymbol{\nu} yx)P \equiv_\pi (\boldsymbol{\nu} xy)P \qquad (\boldsymbol{\nu} xy)\mathbf{0} \equiv_\pi \mathbf{0}$$
$$P \equiv_\alpha Q \implies P \equiv_\pi Q \qquad x, y \notin \mathsf{fv}_\pi(Q) \implies (\boldsymbol{\nu} xy)P \mid Q \equiv_\pi (\boldsymbol{\nu} xy)(P \mid Q)$$

Intuitions on the rules in Fig. 1 follow. Reduction requires an enclosing restriction $(\boldsymbol{\nu} xy)(\cdots)$; this represents that a session connecting endpoints x and y has been already established. Hence, communication cannot occur on free variables, as there is no way to tell what is the pair of interacting co-variables. In Rules $\lfloor$Com$\rfloor$, $\lfloor$Sel$\rfloor$, and $\lfloor$Rep$\rfloor$, the restriction is *persistent* after each reduction, to allow further synchronizations on x and y. In the same rules, process R stands for all the threads that may share x and y.

Rule $\lfloor$Com$\rfloor$ represents the synchronous communication of value v through endpoint x to endpoint y. Furthermore, Rule $\lfloor$Sel$\rfloor$ formalizes a labeled choice mechanism, in which communication of a label l_j is used to choose which of the Q_i will be executed, Rule $\lfloor$Rep$\rfloor$ is similar to Rule $\lfloor$Com$\rfloor$, and used to spawn a new copy of Q, available as a replicated server. Rules $\lfloor$IfT$\rfloor$ and $\lfloor$IfF$\rfloor$ are self-explanatory. Rules for reduction within parallel and restriction contexts, together with reduction up to $\equiv_\pi$, are standard.

(*Pretypes*)	p	$::=$	$?T.T$	*(Receive)*
		$\mid$	$!T.T$	*(Send)*
		$\mid$	$\oplus\{l_i : T_i\}_{i\in I}$	*(Select)*
		$\mid$	$\&\{l_i : T_i\}_{i\in I}$	*(Branch)*
(*Qualifiers*)	q	$::=$	$\mathtt{lin}$	*(Linear)*
		$\mid$	$\mathtt{un}$	*(Unrestricted)*
(*Types*)	T	$::=$	$\mathtt{bool}$	*(Boolean)*
		$\mid$	$\mathtt{end}$	*(Termination)*
		$\mid$	qp	*(Qualified Pretype)*
		$\mid$	$\mathtt{a}$	*(Type Variable)*
		$\mid$	$\mu\mathtt{a}.T$	*(Recursive Type)*
(*Typing contexts*)	Γ	$::=$	$\emptyset$	*(Empty context)*
		$\mid$	$\Gamma, x : T$	*(Assumption)*

Fig. 2: Session Types: Qualifiers, Pre-types, Types, and Typing Contexts.

To reason compositionally about the syntactic structure of processes, we introduce *(evaluation) contexts*. A context represents a process with a "hole", denoted '$-$', which may be filled by another process.

Definition 3 (Contexts for π) The syntax of (evaluation) contexts in π is given by the following grammar:

$$E ::= - \mid E \mid P \mid P \mid E \mid (\boldsymbol{\nu} xy)(E)$$

where P is a π process. We write $C[-]$ to range over contexts of the form $(\boldsymbol{\nu}\widetilde{xy})(-)$. Also, we write $E[P]$ (resp. $C[P]$) to denote the process obtained by filling '$-$' with P.

3.1.2 Type System

We now present the type system for π, a variant of the system in [45]. Rem. 2 discusses the differences of our type system with respect to the one in [45].

We use $q, q', \ldots,$ to range over *qualifiers*; $p, p', \ldots,$ to range over *pre-types*; $T, U, \ldots$ to range over *types*, and $\Gamma, \Gamma', \ldots$ to range over the *typing contexts* which gather assignments of the form $x : T$, where x is a variable and T is a type. As usual, we treat contexts up to the exchange of entries; the variables that appear in a context are required to be pairwise distinct. The concatenation of typing contexts Γ_1 and Γ_2 is written Γ_1, Γ_2.

Definition 4 (Syntax of Types) The syntax of types and typing contexts is in Fig. 2.

Intuitively, pre-types represent pure communication behavior (e.g., send, receive, selection, and branching). Pre-type $!T_1.T_2$ represents a protocol that sends a value of type T_1 and then continues according to type T_2. Dually, pre-type $?T_1.T_2$ represents a protocol that receives a value of type T_1 and then proceeds according to type T_2. Pre-types $\oplus\{l_i : T_i\}_{i\in I}$ and $\&\{l_i : T_i\}_{i\in I}$ denote labeled selection (internal choice) and branching (external choice), respectively.

Pre-types are given a qualifier q to indicate whether the communication behavior is *unrestricted* or *linear*. Linearly qualified pre-types can only be assigned to variables that *do not* appear shared among threads, whereas unrestricted pre-types may be assigned to variables shared among different threads.

Types can be one of the following: (1) $\mathtt{bool}$, used for constants and variables; (2) $\mathtt{end}$, which indicates a terminated behavior; (3) qualified pre-types; or (4) recursive types for disciplining potentially infinite communication patterns. Recursive types are considered equi-recursive (i.e., a recursive type and its unfolding are considered equal because they represent the same regular infinite tree) and contractive (i.e., containing no subexpression of the form $\mu\mathsf{a}_1.\cdots.\mu\mathsf{a}_n.\mathsf{a}_1$) [42]. The qualifier of a recursive type $T = \mu\mathsf{a}.T'$ is obtained via unfolding and by assigning the qualifier of the body T' to type T.

As in [45], we omit $\mathtt{end}$ at the end of types whenever it is not needed; we also write recursive types $\mu\mathsf{a}.\,\mathtt{un}!T.\mathsf{a}$ and $\mu\mathsf{a}.\,\mathtt{un}?T.\mathsf{a}$ as $*\,!T$ and $*\,?T$, respectively.

We use predicates over types to control which types can be shared among variables. While in [45] all unrestricted types can be shared, we proceed differently: to rule out output races, we enforce that only unrestricted input-like types can be shared. We start by presenting an auxiliary predicate that allows us to distinguish output-like types—even when the output behavior in the type is not immediate:

Definition 5 (Output-Like Unrestricted Types) We define the predicate $\mathtt{out}(p)$ on pre-types inductively:

$$\mathtt{out}(!T.U) \stackrel{\text{def}}{=} \mathtt{tt} \qquad \mathtt{out}(?T.U) \stackrel{\text{def}}{=} \mathtt{out}(U)$$
$$\mathtt{out}(\oplus\{l_i : T_i\}_{i\in I}) \stackrel{\text{def}}{=} \mathtt{tt} \qquad \mathtt{out}(\&\{l_i : T_i\}_{i\in I}) \stackrel{\text{def}}{=} \bigvee_{i\in I} \mathtt{out}(T_i)$$

The predicate is lifted to types as follows:

$$\mathtt{out}(\mathtt{bool}) \stackrel{\text{def}}{=} \mathtt{ff} \qquad \mathtt{out}(\mathtt{end}) \stackrel{\text{def}}{=} \mathtt{ff} \qquad \mathtt{out}(\mathsf{a}) \stackrel{\text{def}}{=} \mathtt{ff}$$
$$\mathtt{out}(qp) \stackrel{\text{def}}{=} \mathtt{out}(p) \qquad \mathtt{out}(\mu\mathsf{a}.T) \stackrel{\text{def}}{=} \mathtt{out}(T)$$

Using this predicate, we have the following definition, which specializes the one in [45]:

Definition 6 (Predicates for Types and Contexts) Let T be a session type (cf. Fig. 2). We define $\mathtt{un}^\star(T)$ as follows:

- $\mathtt{un}^\star(T)$ if and only if $(T = \mathtt{bool}) \vee (T = \mathtt{end}) \vee (T = \mathtt{un}\,p \wedge \neg\mathtt{out}(p))$.

Also, we define $\mathtt{un}^\star(\Gamma)$ if and only if $x : T \in \Gamma$ implies $\mathtt{un}^\star(T)$.

Above, predicate $\mathtt{un}^\star(T)$ modifies the $\mathtt{un}(T)$ predicate in [45] to rule out the sharing of output-like types: it requires that pre-types qualified with '$\mathtt{un}$' do not satisfy $\mathtt{out}(\cdot)$.

Session type systems use *duality* to relate types with complementary (or opposite) behaviors: e.g., the dual of input is output (and vice versa); branching is the dual of selection (and vice versa). We define duality by induction on the structure of types.

Definition 7 (Duality of Session Types) For every type T except $\mathtt{bool}$, we define its dual $\overline{T}$ inductively:

$$\overline{\mathtt{end}} \stackrel{\text{def}}{=} \mathtt{end} \qquad \overline{\mathsf{a}} \stackrel{\text{def}}{=} \mathsf{a} \qquad \overline{!T.U} \stackrel{\text{def}}{=} ?T.\overline{U} \qquad \overline{?T.U} \stackrel{\text{def}}{=} !T.\overline{U}$$
$$\overline{\oplus\{l_i : T_i\}_{i\in I}} \stackrel{\text{def}}{=} \&\{l_i : \overline{T_i}\}_{i\in I} \qquad \overline{\&\{l_i : T_i\}_{i\in I}} \stackrel{\text{def}}{=} \oplus\{l_i : \overline{T_i}\}_{i\in I} \qquad \overline{\mu\mathsf{a}.T} \stackrel{\text{def}}{=} \mu\mathsf{a}.\overline{T}$$

Duality in the presence of recursive types is delicate [7]. While intuitive, the inductive definition above is correct only for *tail recursive types*, in which all message types are closed. To account also for non-tail-recursive types (e.g., $\mu \mathtt{a}.?\mathtt{a}.\mathtt{a}$) a more involved *co-inductive* definition is required, cf. Def. 37 in the Appendix. The reader is referred to [23] for a detailed treatment of duality, where Def. 7 is called "naive duality". Notice that using naive duality does not undermine the correctness of our results.

We shall use a *splitting* operator on typing contexts, denoted '$\circ$', to maintain the linearity invariant for variables on typing derivations. Because of predicate $\mathtt{un}^\star(\cdot)$ the splitting operation will not allow to share unrestricted output-like types.

Definition 8 (Typing Context Splitting) Let Γ_1 and Γ_2 be two typing contexts. The (typing) context splitting of Γ_1 and Γ_2, written $\Gamma_1 \circ \Gamma_2$, is defined as follows:

$$\emptyset \circ \emptyset = \emptyset \qquad \frac{\Gamma_1 \circ \Gamma_2 = \Gamma \qquad \mathtt{un}^\star(T)}{(\Gamma_1, x:T) \circ (\Gamma_2, x:T) = \Gamma, x:T}$$

$$\frac{\Gamma_1 \circ \Gamma_2 = \Gamma}{(\Gamma_1, x:T) \circ \Gamma_2 = \Gamma, x:T} \qquad \frac{\Gamma_1 \circ \Gamma_2 = \Gamma}{\Gamma_1 \circ (\Gamma_2, x:T) = \Gamma, x:T}$$

We also define a '$+$' operation to correctly update typing contexts during derivations:

$$\frac{x:T \notin \Gamma}{\Gamma + x:T = \Gamma, x:T} \qquad \frac{\mathtt{un}^\star(T)}{(\Gamma, x:T) + x:T = \Gamma, x:T}$$

There are two typing judgments. We write $\Gamma \vdash v : T$ to denote that value v has type T under Γ. Also, we write $\Gamma \vdash P$ to denote that process P is well typed under Γ.

Fig. 3 gives the typing rules for constants, variables, and processes; some intuitions follow. Rules (T:BOOL) and (T:VAR) are for variables; in both cases, we require $\mathtt{un}^\star(\Gamma)$ to ensure that all variables assigned to types that do not satisfy predicate $\mathtt{un}^\star(\cdot)$ are consumed. Rule (T:IN) types an input process: it checks whether x has the right type and checks the continuation; it also adds variable y with type T and updates x in Γ with type U. To type-check a process $x\langle v\rangle.P$, Rule (T:OUT) splits the typing context in three parts: the first checks the type of the subject x; the second checks the type of the object v; the third checks the continuation P. Rules (T:SEL) and (T:BRA) type-check selection and branching processes, and work similarly to Rules (T:OUT) and (T:IN), respectively.

Rule (T:RIN) types a replicated input $*\,x(y).P$ under the context Γ; it presents several differences with respect to the rule in [45]. Our rule requires Γ to satisfy predicate $\mathtt{un}^\star(\cdot)$. Also, the type T of y must either satisfy $\mathtt{un}^\star(\cdot)$ or be linear. The rule also requires that Γ assigns x an input type qualified with $\mathtt{un}$, and that the continuation P is typed with a context that contains $y : T$ and $x : U$.

Rule (T:PAR) types parallel composition using the (context) splitting operation to divide resources among the two threads. Rule (T:RES) types the restriction operator by performing a duality check on the types of the co-variables. Rule (T:IF) type-checks the conditional process. Given the inactive process $\mathbf{0}$, Rule (T:NIL) checks that the context satisfies $\mathtt{un}^\star(\cdot)$ and Rule (T:WKNIL) ensures that unrestricted types that are output-like (cf. Def. 5) can be weakened when needed. The following example illustrates the need for this rule:

*Example 1 (Recursive Types and Rule (*T:WKNIL*))* We show the kind of recursive processes typable in our system, and the most glaring differences with respect to [45]. Process $P_1 = x\langle\mathtt{tt}\rangle.x\langle\mathtt{ff}\rangle.\mathbf{0}$ is typable both in our system and in the one in [45] under a context

$$\text{(T:Bool)}\ \frac{\mathtt{un}^\star(\Gamma)}{\Gamma \vdash \mathtt{ff}, \mathtt{tt} : \mathtt{bool}} \qquad \text{(T:Var)}\ \frac{\mathtt{un}^\star(\Gamma_1, \Gamma_2)}{\Gamma_1, x : T, \Gamma_2 \vdash x : T}$$

$$\text{(T:In)}\ \frac{\Gamma_1 \vdash x : q?T.U \quad (\Gamma_2 + x : U), y : T \vdash P}{\Gamma_1 \circ \Gamma_2 \vdash x(y).P}$$

$$\text{(T:Out)}\ \frac{\Gamma_1 \vdash x : q!T.U \quad \Gamma_2 \vdash v : T \quad \Gamma_3 + x : U \vdash P}{\Gamma_1 \circ \Gamma_2 \circ \Gamma_3 \vdash x\langle v\rangle.P}$$

$$\text{(T:Sel)}\ \frac{\Gamma_1 \vdash x : q\oplus\{l_i : T_i\}_{i\in I} \quad \Gamma_2 + x : T_j \vdash P \quad j \in I}{\Gamma_1 \circ \Gamma_2 \vdash x \lhd l_j.P}$$

$$\text{(T:Bra)}\ \frac{\Gamma_1 \vdash x : q\&\{l_i : T_i\}_{i\in I} \quad \forall i \in I.\, \Gamma_2 + x : T_i \vdash P_i}{\Gamma_1 \circ \Gamma_2 \vdash x \rhd \{l_i : P_i\}_{i\in I}}$$

$$\text{(T:RIn)}\ \frac{\mathtt{un}^\star(\Gamma) \wedge (\mathtt{un}^\star(T) \vee T = \mathtt{lin}\, p) \quad \Gamma \vdash x : \mathtt{un}?T.U \quad (\Gamma + x : U), y : T \vdash P}{\Gamma \vdash *x(y).P}$$

$$\text{(T:Par)}\ \frac{\Gamma_1 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 \circ \Gamma_2 \vdash P \mid Q} \qquad \text{(T:Res)}\ \frac{\Gamma, x : T, y : \overline{T} \vdash P}{\Gamma \vdash (\boldsymbol{\nu} xy)P}$$

$$\text{(T:If)}\ \frac{\Gamma_1 \vdash v : \mathtt{bool} \quad \Gamma_2 \vdash P \quad \Gamma_2 \vdash Q}{\Gamma_1 \circ \Gamma_2 \vdash v?\, P : Q} \qquad \text{(T:Nil)}\ \frac{\mathtt{un}^\star(\Gamma)}{\Gamma \vdash \mathbf{0}} \qquad \text{(T:WkNil)}\ \frac{\Gamma \vdash \mathbf{0} \quad (T = \mathtt{un}\, p) \wedge \mathtt{out}(p)}{\Gamma, x : T \vdash \mathbf{0}}$$

Fig. 3: Session types: typing rules for π processes.

in which x is assigned the recursive type $T = \mu\mathtt{a}.\,\mathtt{un}!\mathtt{bool}.\mathtt{a}$. Let us consider the typing derivation for P_1 in our system:

$$\text{(T:Out)}\ \frac{\text{(T:Bool)}\ \dfrac{\mathtt{un}^\star(\emptyset)}{\emptyset \vdash \mathtt{tt}} \quad \text{(T:Var)}\ \dfrac{\mathtt{un}^\star(\emptyset)}{x : T \vdash x : q!\mathtt{bool}.U} \quad \text{(T:Out)}\ \dfrac{D}{\emptyset + x : U \vdash x\langle\mathtt{ff}\rangle.\mathbf{0}}}{\emptyset \circ x : \mu\mathtt{a}.\,\mathtt{un}!\mathtt{bool}.\mathtt{a} \circ \emptyset \vdash x\langle\mathtt{tt}\rangle.x\langle\mathtt{ff}\rangle.\mathbf{0}}$$

Notice that, by Def. 6, $\mathtt{un}^\star(T)$ does not hold because $\mathtt{out}(T)$ holds. This in turn influences the context splitting (Def. 8) required by Rule (T:Out): the assignment $x : T$ can only appear in one of the branches of the split (the middle one). Let us consider U and D, which appear unspecified above. Because we use equi-recursive types (as in [45]), T is equivalent to $U = !\mathtt{bool}.\mu\mathtt{a}.\,\mathtt{un}!\mathtt{bool}.\mathtt{a}$, which means that the judgment in the rightmost branch becomes $x : T \vdash x\langle\mathtt{ff}\rangle.\mathbf{0}$. To determine its derivation D, we use Rule (T:WkNil):

$$D = \ \text{(T:Out)}\ \frac{\text{(T:Bool)}\ \dfrac{\mathtt{un}^\star(\emptyset)}{\emptyset \vdash \mathtt{ff}} \quad \text{(T:Var)}\ \dfrac{\mathtt{un}^\star(\emptyset)}{x : T \vdash x : q!\mathtt{bool}.T} \quad \text{(T:WkNil)}\ \dfrac{\text{(T:Nil)}\ \dfrac{\mathtt{un}^\star(\emptyset)}{\emptyset \vdash \mathbf{0}}}{\emptyset + x : T \vdash \mathbf{0}}}{\emptyset \circ x : \mu\mathtt{a}.\,\mathtt{un}!\mathtt{bool}.\mathtt{a} \circ \emptyset \vdash x\langle\mathtt{ff}\rangle.\mathbf{0}}$$

Indeed, before concluding the derivation for the rightmost branch, we are left with the judgment $x : T \vdash \mathbf{0}$. Because $\mathtt{un}^\star(T)$ does not hold, we cannot apply Rule (T:Nil): to com-

plete the derivation, we first apply Rule (T:WKNIL) and then apply Rule (T:NIL). This way, Rule (T:WKNIL) enforces a limited weakening principle, required in the specific case of process $\mathbf{0}$ and an unrestricted type that is output-like.

Consider now process $P_2 = x\langle \mathtt{tt} \rangle.\mathbf{0} \mid x\langle \mathtt{ff} \rangle.\mathbf{0}$, which is typable in [45] under the context $x : T$. This process is not typable in our system because it has an output race on x:

$$\text{(T:PAR)}\ \frac{\Gamma_1 \vdash x\langle \mathtt{tt} \rangle.\mathbf{0} \quad \Gamma_2 \vdash x\langle \mathtt{ff} \rangle.\mathbf{0}}{x : \mu \mathtt{a}.\, \mathtt{un!bool.a} \vdash x\langle \mathtt{tt} \rangle.\mathbf{0} \mid x\langle \mathtt{ff} \rangle.\mathbf{0}}$$

Because $\mathtt{un}^\star(T)$ does not hold, context splitting allows $x : T$ to appear in Γ_1 or Γ_2 but not in both of them. As a result, either Γ_1 or Γ_2 should be empty, which in turn implies that the typing derivation will not be completed.

3.1.3 Type Safety

Our type system enjoys *type safety*, which ensures that well-typed processes do not have communication errors. Type safety depends on the *subject reduction* property, stated next, which ensures that typing is preserved by the reduction relation given in Fig. 1. The proof follows by induction on the derivation of the reduction (cf. App. B.2).

Theorem 1 (Subject Reduction) *If $\Gamma \vdash P$ and $P \longrightarrow Q$ then $\Gamma \vdash Q$.*

To establish type safety, we require auxiliary notions for *pre-redexes* and *redexes*, given below. We use the following notation:

Notation 2 *We write $P = \diamond\, y(z).P'$ to stand for either $P = y(z).P'$ or $P = *\, y(z).P'$.*

We now have:

Definition 9 (Pre-redexes and Redexes) We shall use the following terminology:

- We say $x\langle v \rangle.P$, $x(y).P$, $x \triangleleft l.P$, $x \triangleright \{l_i : P_i\}_{i \in I}$, and $*\, x(y).P$ are *pre-redexes (at variable x)*.
- A *redex* is a process R such that $(\boldsymbol{\nu} xy)R \longrightarrow$ and:
  1. $R = v?\, P : Q$ with $v \in \{\mathtt{tt}, \mathtt{ff}\}$ (or)
  2. $R = x\langle v \rangle.P \mid \diamond\ y(z).Q$ (or)
  3. $R = x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i \in I}$, with $j \in I$.
- A redex R is either *conditional* (if $R = v?\, P : Q$) or *communicating* (otherwise).

We follow [45] in formalizing safety using a *well-formedness* property, which characterizes the set of processes that should be considered correct.

Definition 10 (Well-Formed Process) A process P_0 is *well-formed* if for each of its structural congruent processes $P_0 \equiv_\pi (\boldsymbol{\nu} x_1 y_1) \ldots (\boldsymbol{\nu} x_n y_n)(P \mid Q \mid R)$, with $n \geq 0$, the following conditions hold:

1. If $P \equiv_\pi v?\, P' : P''$ then $v = \mathtt{tt}$ or $v = \mathtt{ff}$.
2. If P and Q are prefixed at the same variable, then they are of the same input-like nature (inputs, replicated inputs, or branchings).
3. If P is prefixed at x_i and Q is prefixed at y_i, $1 \leq i \leq n$, then $P \mid Q$ is a redex.

Unlike the definition in [45], Def. 10(2) excludes processes with output races, i.e., parallel processes can only be prefixed on the same variable if they are input-like. This is how we exclude processes with output races. We now introduce a notation for *programs*:

Notation 3 ((Typable) Programs) *A process P such that $\mathsf{fv}_\pi(P) = \emptyset$ is called* a program. *Therefore, program P is typable if it is well-typed under the empty context ($\vdash P$).*

We can now state type safety, which ensures that every well-typed program is well-formed—hence, well-typed processes have no output races. The proof follows by contradiction (cf. App. B.2).

Theorem 4 (Type Safety) *If $\vdash P$ then P is well-formed.*

Observe that because of Thm. 1, well-formedness is preserved by reduction. Hence:

Corollary 1 *If $\vdash P$ and $P \longrightarrow^* Q$ then Q is well-formed with respect to Def. 10.*

Remark 2 (Differences with respect to [45]) There are three differences between our type system and the one in [45]. First, the modified predicates in Def. 6 enable us to rule out processes with output races, which are typable in [45]. This required adding Rule (T:WKNIL) (cf. Ex. 1). Second, our notion of well-formed processes (Def. 10) excludes processes with output races, which are admitted as well-formed in [45]. Finally, as already discussed, our typing rule for replicated inputs (Rule (T:RIN)) is less permissive than in [45], also for the purpose of ruling out output races.

3.2 Linear Concurrent Constraint Programming (`lcc`)

We now introduce `lcc`, following Haemmerlé [27].

3.2.1 Syntax and Semantics

Variables, ranged over by $x, y, \ldots$, belong to the countably infinite set $\mathcal{V}_l$. We assume that Σ_c and Σ_f correspond to sets of predicate and function symbols, respectively. First-order terms, built from $\mathcal{V}_l$ and Σ_f, will be denoted by $t, t', \ldots$. An arbitrary predicate in Σ_c is denoted $\varphi(\widetilde{t})$.

Definition 11 (Syntax) The syntax for `lcc` is given by the grammar in Fig. 4.

Constraints represent the pieces of information that can be posted to and asked from the store. Constant `tt`, the multiplicative identity, denotes truth; constant `ff` denotes falsehood. Logic connectives used as constructors include the multiplicative conjunction ($\otimes$), bang (!), and the existential quantifier ($\exists\widetilde{x}$). Notation $c\{\widetilde{t}/\widetilde{x}\}$ denotes the constraint obtained by the (capture-avoiding) substitution of the free occurrences of x_i for t_i in c, with $|\widetilde{t}| = |\widetilde{x}|$ and pairwise distinct x_i's. Process substitution $P\{\widetilde{t}/\widetilde{x}\}$ is defined analogously.

The syntax for guards includes non-deterministic choices, denoted $G_1 + G_2$, and *parametric asks* (also called *abstractions*). A parametric ask $\forall\widetilde{x}(c \to P)$ spawns process $P\{\widetilde{t}/\widetilde{x}\}$ if the current store entails constraint $c\{\widetilde{t}/\widetilde{x}\}$; the exact operational semantics for these ask operators (and its interplay with linear constraints) is detailed below. When $\widetilde{x}$ is empty (a parameterless ask), $\forall\widetilde{x}(c \to P)$ is written $\forall\epsilon(c \to P)$.

The syntax of processes includes guards and the *tell operator* $\overline{c}$, which adds constraint c to the current store; *hiding* $\exists\widetilde{x}.\, P$, which declares x as being local to P; parallel composition $P \parallel Q$, which has the expected reading; and replication $!\, P$, which provides infinitely many copies of P. Notation $\prod_{1\leq i\leq n} P_i$ (with $n \geq 1$) stands for process $P_1 \parallel \cdots \parallel P_n$. Universal quantifiers in parametric ask operators and existential quantifiers in hiding operators bind

(Constraints)	c, d	$::=$	$\mathtt{ff}$	*(False)*
		$\mid$	$\mathtt{tt}$	*(True)*
		$\mid$	$\varphi(\widetilde{t})$	*(Predicates)*
		$\mid$	$c \otimes d$	*(Linear Conjunction)*
		$\mid$	$\exists \widetilde{x}.c$	*(Existential)*
		$\mid$	$!\, c$	*(Bang)*
(Guards)	G, G'	$::=$	$\forall \widetilde{x}(c \to P)$	*(Parametric Ask / Abstraction)*
		$\mid$	$G + G'$	*(Non-deterministic Sum)*
(Processes)	P, Q	$::=$	$\overline{c}$	*(Tell)*
		$\mid$	G	*(Guards)*
		$\mid$	$\exists \widetilde{x}.\, P$	*(Hiding)*
		$\mid$	$P \parallel Q$	*(Parallel)*
		$\mid$	$!\, P$	*(Replication)*

Fig. 4: Syntax of `lcc`.

$$(\textsc{Ax})\ \frac{}{c \vdash c} \qquad (\textsc{T1})\ \frac{}{\vdash \mathtt{tt}} \qquad (\textsc{Cut})\ \frac{\Gamma, c \vdash d \quad \Delta \vdash c}{\Gamma, \Delta \vdash d} \qquad (\textsc{T2})\ \frac{\Gamma \vdash c}{\Gamma, \mathtt{tt} \vdash c}$$

$$(\textsc{L}_\otimes)\ \frac{\Gamma, c_1, c_2 \vdash c}{\Gamma, c_1 \otimes c_2 \vdash c} \qquad (\textsc{R}_\otimes)\ \frac{\Gamma \vdash c_1 \quad \Delta \vdash c_2}{\Gamma, \Delta \vdash c_1 \otimes c_2} \qquad (\textsc{L}_\exists)\ \frac{\Gamma, c \vdash d}{\Gamma, \exists x.c \vdash d}\ x \notin \mathsf{fv}(\Gamma, d)$$

$$(\textsc{R}_\exists)\ \frac{\Gamma \vdash c\{^t/_x\}}{\Gamma \vdash \exists x.c} \qquad (!_1)\ \frac{\Gamma, c \vdash d}{\Gamma, !\, c \vdash d} \qquad (!_2)\ \frac{!\,\Gamma \vdash d}{!\,\Gamma \vdash !\, d} \qquad (!_3)\ \frac{\Gamma \vdash d}{\Gamma, !\, c \vdash d} \qquad (!_4)\ \frac{\Gamma, !\, c, !\, c \vdash d}{\Gamma, !\, c \vdash d}$$

Fig. 5: Intuitionistic Sequent Calculus for `lcc` (cf. Def. 12).

their respective variables. Given this, the set of free variables in constraints and processes is defined as expected, and denoted $\mathsf{fv}(\cdot)$.

The semantics of processes is defined as a Labeled Transition System (LTS), which relies on a structural congruence on processes. The semantics is parametric in a *constraint system*, as defined next.

Definition 12 (Constraint System) A *constraint system* is a triplet $(\mathcal{C}, \Sigma, \vdash)$, where Σ contains Σ_c (i.e., the set of predicates) and Σ_f (i.e., the set of functions and constants). $\mathcal{C}$ is the set of constraints obtained by using the grammar in Def. 11 and Σ. Relation $\Vdash$ is a subset of $\mathcal{C} \times \mathcal{C}$ that defines the non-logical axioms of the constraint system. Relation $\vdash$ is the least subset of $\mathcal{C}^* \times \mathcal{C}$ containing $\Vdash$ and closed by the deduction rules of intuitionistic linear logic (see Fig. 5). We write $c \dashv\vdash d$ whenever both $c \vdash d$ and $d \vdash c$ hold.

Definition 13 (Structural Congruence) The structural congruence relation is the smallest equivalence relation $\equiv$ that satisfies α-renaming of bound variables, commutativity and

$$\lfloor \text{C:Out} \rfloor \quad \frac{c \vdash \exists \widetilde{x}.(d \otimes e) \quad \exists \widetilde{x}.d \vdash \exists \widetilde{x'}.d' \quad \mathbf{mgc}\big(c, \exists \widetilde{x}.(d \otimes e)\big) \quad (\widetilde{x} \cup \widetilde{x'}) \cap \mathsf{fv}(c) = \emptyset}{\overline{c} \xrightarrow{(\widetilde{x'})\overline{d'}}_{\ell} \overline{e}}$$

$$\lfloor \text{C:Sync} \rfloor \quad \frac{c \vdash \exists \widetilde{y}(d\{\widetilde{t}/\widetilde{x}\} \otimes e) \qquad \widetilde{y} \cap \mathsf{fv}(c,d,P) = \emptyset \qquad \mathbf{mgc}\big(c, \exists \widetilde{y}.(d\{\widetilde{t}/\widetilde{x}\} \otimes e)\big)}{\overline{c} \parallel \forall \widetilde{x}(d \to P) \xrightarrow{\tau}_{\ell} \exists \widetilde{y}.\, (P\{\widetilde{t}/\widetilde{x}\} \parallel \overline{e})}$$

$$\lfloor \text{C:In} \rfloor \quad \frac{}{\overline{\mathtt{tt}} \xrightarrow{c}_{\ell} \overline{c}} \qquad \lfloor \text{C:Comp} \rfloor \quad \frac{P \xrightarrow{\alpha}_{\ell} P' \quad ev(\alpha) \cap \mathsf{fv}(Q) = \emptyset}{P \parallel Q \xrightarrow{\alpha}_{\ell} P' \parallel Q} \qquad \lfloor \text{C:Sum} \rfloor \quad \frac{P \parallel G_i \xrightarrow{\alpha}_{\ell} P' \quad i \in \{1,2\}}{P \parallel G_1 + G_2 \xrightarrow{\alpha}_{\ell} P'}$$

$$\lfloor \text{C:Ext} \rfloor \quad \frac{P \xrightarrow{(\widetilde{x})\overline{c}}_{\ell} Q}{\exists y.\, P \xrightarrow{(y\widetilde{x})\overline{c}}_{\ell} Q} \qquad \lfloor \text{C:Res} \rfloor \quad \frac{P \xrightarrow{\alpha}_{\ell} P' \quad y \notin \mathsf{fv}(\alpha)}{\exists y.\, P \xrightarrow{\alpha}_{\ell} \exists y.\, P'} \qquad \lfloor \text{C:Cong} \rfloor \quad \frac{P \equiv P' \quad P' \xrightarrow{\alpha}_{\ell} Q' \quad Q' \equiv Q}{P \xrightarrow{\alpha}_{\ell} Q}$$

Fig. 6: Labeled Transition System (LTS) for `lcc` processes.

associativity for parallel composition and summation, together with the following identities:

$$(\text{SC}_\ell\text{:1}) \quad P \parallel \overline{\mathtt{tt}} \equiv P \qquad (\text{SC}_\ell\text{:2}) \quad \exists z.\, \overline{\mathtt{tt}} \equiv \overline{\mathtt{tt}} \qquad (\text{SC}_\ell\text{:3}) \quad \exists x.\, \exists y.\, P \equiv \exists y.\, \exists x.\, P \qquad (\text{SC}_\ell\text{:4}) \quad !\, P \equiv P \parallel !\, P$$

$$(\text{SC}_\ell\text{:5}) \quad \frac{c \otimes d \dashv\vdash e}{\overline{c} \parallel \overline{d} \equiv \overline{e}} \qquad (\text{SC}_\ell\text{:6}) \quad \frac{P \equiv P'}{P \parallel Q \equiv P' \parallel Q} \qquad (\text{SC}_\ell\text{:7}) \quad \frac{z \notin \mathsf{fv}(P)}{P \parallel \exists z.\, Q \equiv \exists z.\, (P \parallel Q)} \qquad (\text{SC}_\ell\text{:8}) \quad \frac{P \equiv P'}{\exists x.\, P \equiv \exists x.\, P'}$$

As customary, a (strong) transition $P \xrightarrow{\alpha}_{\ell} P'$ denotes the evolution of process P to P' by performing the action denoted by the transition label α:

$$\alpha ::= \tau \mid c \mid (\widetilde{x})\overline{c}$$

Label τ denotes a silent (internal) action. Label $c \in \mathcal{C}$ denotes a constraint "received" as an input action (but see below) and $(\widetilde{x})\overline{c}$ denotes an output (tell) action in which $\widetilde{x}$ are extruded variables and $c \in \mathcal{C}$. We write $ev(\alpha)$ to refer to these extruded variables.

Before discussing the transition rules (cf. Fig. 6), we introduce a key notion: the *most general choice* predicate:

Definition 14 (Most General Choice (mgc) [27]) Let c, d, and e be constraints, $\widetilde{x}, \widetilde{y}$ be vectors of variables, and $\widetilde{t}$ be a vector of terms. We write

$$\mathbf{mgc}\big(c, \exists \widetilde{y}.(d\{\widetilde{t}/\widetilde{x}\} \otimes e)\big)$$

whenever for any constraint e', all terms $\widetilde{t'}$ and all variables $\widetilde{y'}$, if $c \vdash \exists \widetilde{y'}.(d\{\widetilde{t'}/\widetilde{x}\} \otimes e')$ and $\exists \widetilde{y'}.e' \vdash \exists \widetilde{y}.e$ hold, then $\exists \widetilde{y}.(d\{\widetilde{t}/\widetilde{x}\}) \vdash \exists \widetilde{y'}.(d\{\widetilde{t'}/\widetilde{x}\})$ and $\exists \widetilde{y}.e \vdash \exists \widetilde{y'}.e'$.

Intuitively, the $\mathbf{mgc}$ predicate allows us to refer formally to decompositions of a constraint c (seen as a linear resource) that do not "lose" or "forget" information in c. This is essential in the presence of linear constraints. For example, assuming that $c \vdash d \otimes e$ holds, we can see that $\mathbf{mgc}(c, d \otimes e)$ holds too, because c is the precise amount of information

necessary to obtain $d \otimes e$. However, $\mathbf{mgc}(c \otimes f, d \otimes e)$ does not hold, assuming $f \neq \mathtt{tt}$, since $c \otimes f$ produces more information than the necessary to obtain $d \otimes e$.

We briefly discuss the transition rules of Fig. 6. Rule $\lfloor$C:IN$\rfloor$ asynchronously receives a constraint; it represents the separation between observing an output and its (asynchronous) reception, which is not directly observable.

Rule $\lfloor$C:OUT$\rfloor$ formalizes asynchronous tells: using the $\mathbf{mgc}$ predicate, the emitted constraint is decomposed in two parts: the first one is actually sent (as recorded in the label); the second part is kept as a continuation. (In the rule, these two parts are denoted as d' and e, respectively.) Rule $\lfloor$C:SYNC$\rfloor$ formalizes the synchronization between a tell (i.e., an output) and a parametric ask. The constraint mentioned in the tell is decomposed using the $\mathbf{mgc}$ predicate: in this case, the first part is used (consumed) to "trigger" the processes guarded by the ask, while the second part is the remaining continuation.

Rule $\lfloor$C:COMP$\rfloor$ enables the parallel composition of two processes P and Q, provided that the variables extruded in an action by P are disjoint from the free variables of Q. Rule $\lfloor$C:SUM$\rfloor$ enables non-deterministic choices at the level of guards.

Rules $\lfloor$C:EXT$\rfloor$ and $\lfloor$C:RES$\rfloor$ formalize hiding: the former rule makes local variables explicit in the transition label; the latter rule avoids the hiding of free variables in the label.

Finally, Rule $\lfloor$C:CONG$\rfloor$ closes transitions under structural congruence (cf. Def. 13).

Notation 5 (τ-transitions) *Some terminology and notation for τ-transitions in* `lcc`*:*

- *We shall write $\xrightarrow{\tau}{}^*_\ell$ to denote a sequence of zero or more τ-labeled transitions. Whenever the number $k \geq 1$ of τ-transitions is fixed, we write $\xrightarrow{\tau}{}^k_\ell$.*
- *When τ-labels are unimportant (or clear from the context) we shall write $\longrightarrow_\ell$, $\longrightarrow^*_\ell$, and $\longrightarrow^k_\ell$ to stand for $\xrightarrow{\tau}{}_\ell$, $\xrightarrow{\tau}{}^*_\ell$, and $\xrightarrow{\tau}{}^k_\ell$, respectively.*
- Weak transitions *are standardly defined: we write $P \stackrel{\tau}{\Longrightarrow} Q$ if and only if $(P \xrightarrow{\tau}{}^*_\ell Q)$; similarly, we write $P \stackrel{\alpha}{\Longrightarrow} Q$ if and only if $(P \xrightarrow{\tau}{}^*_\ell P' \xrightarrow{\alpha}{}_\ell P'' \xrightarrow{\tau}{}^*_\ell Q)$.*

3.2.2 Observational Equivalences

We require the following auxiliary definition from [27]:

Definition 15 ($\mathcal{D}$-Accessible Constraints) Let $\mathcal{D} \subset \mathcal{C}$, where $\mathcal{C}$ is the set of all constraints. The observables of a process P are the set of all $\mathcal{D}$-accessible constraints defined as follows:

$$\mathcal{O}^{\mathcal{D}}(P) \stackrel{\mathtt{def}}{=} \{(\exists \widetilde{x}.c) \in \mathcal{D} \mid \text{there exists } P'.\, P \stackrel{\tau}{\Longrightarrow} \exists \widetilde{x}.(P' \parallel \overline{c})\}$$

Next, we introduce a notion of equivalence for `lcc` processes: *weak barbed congruence*, as in [27]. We first need a notation that parameterizes processes in terms of the constraints they can tell and ask (see below). Then, we introduce (evaluation) contexts for `lcc`.

Notation 6 ($\mathcal{DE}$-Processes) *Let $\mathcal{D} \subseteq \mathcal{C}$ and $\mathcal{E} \subseteq \mathcal{C}$. Also, let P be a process (cf. Def. 11).*

- *P is* $\mathcal{D}$-ask restricted *if for every sub-processs $\forall \widetilde{x}(c \to P')$ in P, we have $\exists \widetilde{z}.c \in \mathcal{D}$.*
- *P is* $\mathcal{E}$-tell restricted *if for every sub-process $\overline{c}$ in P, we have $\exists \widetilde{z}.c \in \mathcal{E}$.*
- *If P is both $\mathcal{D}$-ask restricted and $\mathcal{E}$-tell restricted then we call P a* $\mathcal{DE}$-process.

Definition 16 (Contexts in `lcc`) Let E be the evaluation contexts for `lcc` as given by the following grammar, where '$-$' represents a hole and P is a process:

$$E ::= - \mid P \parallel E \mid E \parallel P \mid \exists \widetilde{x}.E$$

Given an evaluation context $E[-]$, we write $E[P]$ to denote the process that results from filling in the occurrences of the hole with process P.

Given $\mathcal{D} \subseteq \mathcal{C}$ and $\mathcal{E} \subseteq \mathcal{C}$, we will say that a context is a $\mathcal{DE}$*-context*, ranged over $C, C', \ldots$, if it is formed only by $\mathcal{DE}$-processes.

We may now define *weak barbed bisimulation* and *weak barbed congruence*:

Definition 17 (Weak $\mathcal{DE}$-Barbed Bisimulation) Let $\mathcal{D} \subseteq \mathcal{C}$ and $\mathcal{E} \subseteq \mathcal{C}$. A *symmetric* relation $\mathcal{R}$ is a $\mathcal{DE}$-barbed bisimulation if, for $\mathcal{DE}$-processes P and Q, $(P, Q) \in \mathcal{R}$ implies:

(1) $\mathcal{O}^{\mathcal{D}}(P) = \mathcal{O}^{\mathcal{D}}(Q)$ (and),
(2) whenever $P \xrightarrow{\tau}_{\ell} P'$ there exists Q' such that $Q \stackrel{\tau}{\Longrightarrow} Q'$ and $P' \mathcal{R} Q'$.

The largest weak barbed $\mathcal{DE}$-bisimulation is called $\mathcal{DE}$-bisimilarity and is denoted by $\approx_{\mathcal{DE}}$.

Definition 18 (Weak $\mathcal{DE}$-Barbed Congruence) We say that two processes P, Q are weakly barbed $\mathcal{DE}$-congruent, denoted by $P \cong_{\mathcal{DE}} Q$, if for every $\mathcal{DE}$-context $E[-]$ it holds that $E[P] \approx_{\mathcal{DE}} E[Q]$. We define the weak barbed $\mathcal{DE}$-congruence $\cong_{\mathcal{DE}}$ as the largest $\mathcal{DE}$-congruence that is a weak barbed $\mathcal{DE}$-bisimilarity.

3.3 Relative Expressiveness

We shall work with *(valid) encodings*, i.e., language translations that satisfy some correctness (or encodability) criteria. We follow the encodability criteria defined by Gorla [26], which define a general and widely used framework for studying relative expressiveness.

3.3.1 Languages and Translations

Definition 19 (Languages and Translations) We define:

- A *language* $\mathcal{L}$ is a triplet $\langle \mathsf{P}, \longrightarrow, \approx \rangle$, where P is a set of terms (i.e., expressions, processes), $\longrightarrow$ is a relation on P defining its operational semantics, and $\approx$ is an equivalence on P. We use $\Longrightarrow$ to denote the reflexive-transitive closure of $\longrightarrow$.
- A *translation* from $\mathcal{L}_s = \langle \mathsf{P}_s, \longrightarrow_s, \approx_s \rangle$ into $\mathcal{L}_t = \langle \mathsf{P}_t, \longrightarrow_t, \approx_t \rangle$ (each with countably infinite sets of variables V_s and V_t, respectively) is a pair $\langle [\![\cdot]\!], \psi_{[\![\cdot]\!]} \rangle$, where $[\![\cdot]\!] : \mathsf{P}_s \to \mathsf{P}_t$ is defined as a mapping from source terms to target terms, and $\psi_{[\![\cdot]\!]} : \mathsf{V}_s \to \mathsf{V}_t$ is a *renaming policy* for $[\![\cdot]\!]$, which maps source variables to target variables.

In a language $\mathcal{L}$, the set P of terms is defined as a formal grammar that gives the formation rules. The operational semantics $\longrightarrow$ is given as a relation on terms, finitely denoted by sets of rules. We write $P \longrightarrow P'$ to represent a pair (P, P') that is included in the relation; we call each one of these pairs a *step*. Each step represents the fact that term P *reduces* to term P'. For the rest of this section, we will refer to $P \longrightarrow P'$ as a reduction step. Moreover, we use $P \Longrightarrow P'$ to say that P reduces to P' in zero or more steps (i.e., a "multi-step" reduction). Finally, $\approx$ denotes an equivalence on terms of the language.

In $\langle [\![\cdot]\!], \psi_{[\![\cdot]\!]} \rangle$, the mapping $[\![\cdot]\!]$ assigns each source term a corresponding target term. It is usually defined inductively over the structure of source terms. The renaming policy $\psi_{[\![\cdot]\!]}$ translates variables. In our translation, a variable is simply translated into itself; the general formulation of a renaming policy given in [26] is not needed. When referring to translations, we often use $[\![\cdot]\!]$ instead of $\langle [\![\cdot]\!], \psi_{[\![\cdot]\!]} \rangle$.

We now introduce some terminology regarding translations.

Notation 7 *Let $\langle [\![\cdot]\!], \psi_{[\![\cdot]\!]} \rangle$ be a translation from $\mathcal{L}_s = \langle \mathsf{P}_s, \rightarrow_s, \approx_s \rangle$ into $\mathcal{L}_t = \langle \mathsf{P}_t, \rightarrow_t, \approx_t \rangle$.*

- *We will refer to $\mathcal{L}_s$ and $\mathcal{L}_t$ as* source *and* target *languages of the translation, respectively. Whenever it does not create any confusion, we will only refer to source and target languages as* source *and* target.
- *We say that any process $S \in \mathsf{P}_s$ is a* source term. *Similarly, given a source term S, any process $T \in \mathsf{P}_t$ that is reachable from $[\![S]\!]$ using $\Longrightarrow_t$ is called a* target term.

3.3.2 Correctness Criteria

To focus on meaningful translations, we define *correctness criteria*: a set of properties that determine whether a translation is a *valid encoding* or not. Following [26], we shall be interested in *name invariance*, *compositionality*, *operational completeness*, *operational soundness*, and *success sensitiveness*.

Definition 20 (Valid Encoding) Let $\mathcal{L}_s = \langle \mathsf{P}_s, \rightarrow_s, \approx_s \rangle$ and $\mathcal{L}_t = \langle \mathsf{P}_t, \rightarrow_t, \approx_t \rangle$ be languages. Also, let $\langle [\![\cdot]\!], \psi_{[\![\cdot]\!]} \rangle$ be a translation between them (cf. Def. 19). Such a translation is a *valid encoding* if it satisfies the following criteria:

1. **Name invariance**: For all $S \in \mathsf{P}_s$ and substitution σ, there exists σ' such that $[\![S\sigma]\!] = [\![S]\!]\sigma'$, with $\psi_{[\![\cdot]\!]}(\sigma(x)) = \sigma'(\psi_{[\![\cdot]\!]}(x))$, for any $x \in \mathsf{V}_s$.
2. **Compositionality**: For every k-ary operator op of P_s there exists a k-ary context C_{op} in P_t such that for all $S_1, \ldots, S_k \in \mathsf{P}_s$, it holds that
$$[\![\mathsf{op}(S_1, \ldots, S_k)]\!] = C_{\mathsf{op}}([\![S_1]\!], \ldots, [\![S_k]\!]).$$
3. **Operational Completeness:** For every $S, S' \in \mathsf{P}_s$ such that $S \Longrightarrow_s S'$, it holds that $[\![S]\!] \Longrightarrow_t T$ and $T \approx_t [\![S']\!]$, for some $T \in \mathsf{P}_t$.
4. **Operational Soundness:** For every $S \in \mathsf{P}_s$ and $T \in \mathsf{P}_t$ such that $[\![S]\!] \Longrightarrow_t T$, there exist S', T' such that $S \Longrightarrow_s S'$ and $T \Longrightarrow_t T'$ and $T' \approx_t [\![S']\!]$.
5. **Success Sensitiveness**: Given $\Downarrow_s$ (resp. $\Downarrow_t$) the unary *success predicate* for P_s (resp. P_t), for every $S \in \mathsf{P}_s$ it holds that $S \Downarrow_s$ if and only if $[\![S]\!] \Downarrow_t$.

Name invariance ensures that substitutions are well-behaved in translated terms. Condition $\psi_{[\![\cdot]\!]}(\sigma(x)) = \sigma'(\psi_{[\![\cdot]\!]}(x))$ ensures that for every variable substituted in the source term (i.e., $\sigma(x)$), there exists a substitution σ' such that the translation of x (i.e., $\psi_{[\![\cdot]\!]}(x)$) is substituted by the translation of $\sigma(x)$. The renaming policy $\psi_{[\![\cdot]\!]}(x)$ is particularly important in translations that fix some variables to play a specific role or that translate a single variable into a vector of variables. This is not the case here: as already mentioned, we shall require a simple renaming policy that translates a variable into itself.

Compositionality ensures that the translation of a composite term depends on the translation of its sub-terms. These sub-terms should be combined in a unique target context that ensures that their interactions are preserved. Unlike Gorla's definition of compositionality, we do not need the target context to be parametric on a set of free names.

Together, operational completeness and soundness form the *operational correspondence* criterion, which deals with preservation and reflection of process behavior. Intuitively, operational completeness is about preserving the behavior of the source semantics: it requires that for every multi-step reduction in the source language there exists a corresponding multi-step reduction in the target language. The equivalence $\approx_t$ then ensures that the target term

$$\Sigma ::= \mathsf{rcv}(x,y) \mid \mathsf{snd}(x,y) \mid \mathsf{sel}(x,l) \mid \mathsf{bra}(x,l) \mid \{x{:}y\}$$

Fig. 7: Session constraint system: Predicates (cf. Def. 21).

thus obtained is behaviorally equivalent to the translation of the reduced source term. Operational soundness, on the other hand, ensures that the target semantics does not introduce extraneous steps that do not correspond to any source behaviors: it requires that every reduction in the target language corresponds to a reduction in the source language, using $\approx_t$ to ensure that the reduced target term is behaviorally equivalent to the reduced source term.

Success sensitiveness assumes a "success" predicate, definable on source processes (denoted $S{\Downarrow}$) and on target processes (denoted $T{\Downarrow}$). In the name-passing calculi considered in [26], this predicate naturally corresponds to the notion of *observable* (or *barb*). In our setting, we will define a success predicate based on the potential that a process has of reducing to a process with an unguarded occurrence of the *success process*, denoted $\checkmark$.

Having introduced the two languages and a framework for comparing their relative expressiveness, we now present our translation of π into `lcc` and establish its correctness.

4 Encoding π into `lcc`

We present the encoding from π into `lcc`, the main contribution of our work. This section is structured as follows. In § 4.1, we define the translation from π into `lcc` and illustrate it by means of examples. We prove that the translation is a valid encoding: first, in § 4.2 we prove name invariance, compositionality, and operational completeness properties; then, operational soundness and success sensitiveness are proven in § 4.3 and § 4.4, respectively.

4.1 The Translation

Our translation relies on the constraint system defined next.

Definition 21 (Session Constraint System) A *session constraint system* is represented by the tuple $\langle \mathcal{C}, \Sigma, \vdash_{\mathcal{S}} \rangle$, where:

- Σ is the set of predicates given in Fig. 7;
- $\mathcal{C}$ is the set of constraints obtained by using linear logic operators !, $\otimes$ and $\exists$ over the predicates of Σ;
- $\vdash_{\mathcal{S}}$ is given by the rules in Fig. 5 (cf. Def. 12), extended with the syntactic equality '$=$'.

The first four predicates in Fig. 7 serve as acknowledgments of actions in the source π process: predicate $\mathsf{rcv}(x,y)$ signals an input action on x of a value denoted by y; conversely, predicate $\mathsf{snd}(x,y)$ signals an output action on x of a value denoted by y. Predicates $\mathsf{sel}(x,l)$ and $\mathsf{bra}(x,l)$ signal selection and branching actions on x involving label l, respectively. Finally, predicate $\{x{:}y\}$ indicates that x and y denote dual endpoints, as required to translate restriction in π. To ensure alignment with the properties of restricted co-variables in π (cf. Def. 2), we assume $\{x{:}y\} \dashv\vdash_{\mathcal{S}} \{y{:}x\}$ for every pair of variables x, y.

Defining `lcc` as a language in the sense of Def. 24 requires setting up observational equivalences (cf. Def. 17 and Def. 18). To this end, we first define two sets of observables: the *output* and *complete* observables of `lcc` processes under the constraint system in Def. 21.

Definition 22 (Output and Complete Observables) Let $\mathcal{C}$ be the constraint system in Def. 21. We define $\mathcal{D}_\pi$, the set of *output observables* of `lcc`, as follows:

$$\begin{aligned}\mathcal{D}_\pi \stackrel{\text{def}}{=} &\{\exists \widetilde{z}.\mathsf{snd}(x,v) \mid x \in \mathcal{V}_\pi \wedge x \in \widetilde{z} \wedge (v \in \widetilde{z} \vee v \notin \mathcal{V}_\pi)\}\\ &\cup \{\exists \widetilde{z}.\mathsf{sel}(x,l) \mid x \in \mathcal{V}_\pi \wedge l \in \mathcal{B}_\pi \wedge x \in \widetilde{z}\}\end{aligned}$$

We define $\mathcal{D}^\star_\pi$, the set of *complete observables* of `lcc`, as the following extension of $\mathcal{D}_\pi$:

$$\begin{aligned}\mathcal{D}^\star_\pi \stackrel{\text{def}}{=} \mathcal{D}_\pi \cup \{\mathtt{tt}\} \cup &\{\exists \widetilde{z}.\mathsf{rcv}(x,y) \mid x,y \in \mathcal{V}_\pi \wedge x \neq y \wedge x \in \widetilde{z}\}\\ &\cup \{\exists \widetilde{z}.\mathsf{bra}(x,l) \mid x \in \mathcal{V}_\pi \wedge l \in \mathcal{B}_\pi \wedge x \in \widetilde{z}\}\end{aligned}$$

Notice that constraints such as $\{x{:}y\}$ are not part of the observables. As we will see, co-variable predicates will be persistent, and so the information on co-variables can be derived by using other constraints. In particular, as we will show later, if $\exists x, y.\mathsf{snd}(x,v)$ and $\exists x, y.\mathsf{rcv}(y,v)$ are in the complete observables of a process then constraint $!\{x{:}y\}$ must be in the corresponding store too. This will become clear when analyzing the shape of translated processes (cf. Lem. 1).

We are now ready to instantiate sets $\mathcal{D}$ and $\mathcal{E}$ for the barbed bisimilarity (cf. Def. 17) and the barbed congruence for `lcc` (cf. Def. 18). To this end, we let $\mathcal{D} = \mathcal{D}_\pi$, and $\mathcal{E} = \mathcal{C}$ (cf. Def. 21).

Definition 23 (Weak o-barbed bisimilarity and congruence) We define *weak o-barbed bisimilarity* and *weak o-barbed congruence* as follows:

1. Weak o-barbed bisimilarity, denoted $\approx^\pi_\ell$, arises from Def. 17 as the weak $\mathcal{D}_\pi\mathcal{C}$-barbed bisimilarity.
2. Weak o-barbed congruence, denoted $\cong^\pi_\ell$, arises from Def. 18 as the weak $\mathcal{D}_\pi\mathcal{C}$-barbed congruence.

We define π and `lcc` as the source and target languages for our translation, respectively:

Definition 24 (Source and Target Language)

(1) The language $\mathcal{L}_\pi$ is defined by the triplet $\langle \pi, \longrightarrow, \equiv_\pi \rangle$, where π is as in Def. 3.1.1, $\longrightarrow$ is as in Fig. 1, and $\equiv_\pi$ is as in Def. 2.
(2) The language $\mathcal{L}_{\mathtt{lcc}}$ is given by the triplet $\langle \mathtt{lcc}, \longrightarrow_\ell, \cong^\pi_\ell \rangle$, where `lcc` is as in Def. 11, $\longrightarrow_\ell$ is the relation given only by τ-transitions (cf. Fig. 6), and $\cong^\pi_\ell$ is the behavioral equivalence in Def. 23.

The translation of $\mathcal{L}_\pi$ into $\mathcal{L}_{\mathtt{lcc}}$ is defined as follows:

Definition 25 (Translation of π into `lcc`) The translation from $\mathcal{L}_\pi$ into $\mathcal{L}_{\mathtt{lcc}}$ (cf. Def. 24) is the pair $\langle [\![\cdot]\!], \varphi_{[\![\cdot]\!]} \rangle$, where $[\![\cdot]\!]$ is the process mapping defined in Fig. 8 and $\varphi_{[\![\cdot]\!]}(x) = x$.

Let us discuss some of the cases of the definition in Fig. 8:

– The output process $x\langle v \rangle.P$ is translated by using both tell and abstraction constructs:

$$\overline{\mathsf{snd}(x,v)} \parallel \forall z\big(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \rightarrow [\![P]\!]\big)$$

The translation posts predicate $\mathsf{snd}(x,v)$ in the store, signaling that an output has taken place, and can be received by the translation of an input process. The translation of the continuation is activated once predicate $\mathsf{rcv}(y,v)$ has been received: this signals that the

$$
\begin{aligned}
[\![x\langle v\rangle.P]\!] &\stackrel{\text{def}}{=} \overline{\mathsf{snd}(x,v)} \parallel \forall z\big(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]\big)\\
[\![x(y).P]\!] &\stackrel{\text{def}}{=} \forall y,w\big(\mathsf{snd}(w,y) \otimes \{w{:}x\} \to \overline{\mathsf{rcv}(x,y)} \parallel [\![P]\!]\big)\\
[\![x \triangleleft l.P]\!] &\stackrel{\text{def}}{=} \overline{\mathsf{sel}(x,l)} \parallel \forall z\big(\mathsf{bra}(z,l) \otimes \{x{:}z\} \to [\![P]\!]\big)\\
[\![x \triangleright \{l_i{:}P_i\}_{i\in I}]\!] &\stackrel{\text{def}}{=} \forall l,w\big(\mathsf{sel}(w,l) \otimes \{w{:}x\} \to \overline{\mathsf{bra}(x,l)} \parallel \prod_{i\in I} \forall\epsilon(l = l_i \to [\![P_i]\!])\big)\\
[\![v?\,P{:}Q]\!] &\stackrel{\text{def}}{=} \forall\epsilon(v = \mathtt{tt} \to [\![P]\!]) \parallel \forall\epsilon(v = \mathtt{ff} \to [\![Q]\!])\\
[\![(\boldsymbol{\nu} xy)P]\!] &\stackrel{\text{def}}{=} \exists x,y.\,(!\overline{\{x{:}y\}} \parallel [\![P]\!])\\
[\![*\,x(y).P]\!] &\stackrel{\text{def}}{=} !\,[\![x(y).P]\!]\\
[\![P \mid Q]\!] &\stackrel{\text{def}}{=} [\![P]\!] \parallel [\![Q]\!]\\
[\![\mathbf{0}]\!] &\stackrel{\text{def}}{=} \overline{\mathtt{tt}}
\end{aligned}
$$

Fig. 8: Translation from π to `lcc` (cf. Def. 25).

message has been correctly received by a translated process that contains its co-variable (e.g., predicate $\{x{:}y\}$). Therefore, input-output interactions are represented in $[\![\cdot]\!]$ as a two-step synchronization. As we stick to the variable convention in Rem. 1, a proviso such as "$z \notin \mathsf{fv}_\pi(P)$" is redundant here (and in the cases below).

– Accordingly, the translation of an input process $x(y).P$ is defined as follows:

$$\forall y,w\big(\mathsf{snd}(w,y) \otimes \{w{:}x\} \to \overline{\mathsf{rcv}(x,y)} \parallel [\![P]\!]\big)$$

Whenever a predicate $\mathsf{snd}(x,v)$ is detected by the abstraction, constraint $\mathsf{snd}(x,v)$ is consumed to obtain both the subject x and the object y. Then, the co-variable restriction is checked: this enforces synchronization between intended endpoints. Subsequently, the translation emits a message $\mathsf{rcv}(\cdot,\cdot)$ and spawns its continuation.

– The translation of branching-selection synchronizations is similar, using $\mathsf{bra}(\cdot,\cdot)$ and $\mathsf{sel}(\cdot,\cdot)$ as acknowledgment messages. In this case, the exchanged value is one of the pairwise distinct labels, say l_j; depending on the received label, the translation of branching will spawn exactly one continuation. The continuations corresponding to labels different from l_j get blocked, as their equality guard can never be satisfied. Similarly, the translation of conditionals makes both branches available for execution; we use a parameterized ask as guard to ensure that only one of them will be executed.
– The translation of process $(\boldsymbol{\nu} xy)P$ provides infinitely many copies of the co-variable constraint $\{x{:}y\}$, using hiding in `lcc` to appropriately regulate the scope of the involved endpoints.
– The translation of replicated processes simply corresponds to the replication of the translation of the given input-guarded process. Finally, the translations of parallel composition and inaction are self-explanatory.

The following examples illustrate our translation.

Example 2 (Translating Session Delegation) We show how our translation captures *session delegation*. Consider the following π process:

$$P_1 = (\boldsymbol{\nu} wz)(\boldsymbol{\nu} xy)(x\langle z\rangle.w(u').\mathbf{0} \mid y(u).u\langle\mathtt{tt}\rangle.\mathbf{0})$$

Above, endpoint z is being sent over x, to be received by endpoint y, which then enables the communication between w and z. The translated process $[\![P_1]\!]$ is given below:

$$\begin{array}{l}\exists x,y,w,z.\big(\,!\,\overline{\{x{:}y\}} \parallel\, !\,\overline{\{w{:}z\}} \parallel \overline{\mathsf{snd}(x,z)} \parallel\\ \quad \forall w_1\big(\mathsf{rcv}(w_1,z)\otimes\{x{:}w_1\}\to \forall w_2,u'\big(\mathsf{snd}(w_2,u')\otimes\{w_2{:}w\}\to \overline{\mathsf{rcv}(w,u')} \parallel \overline{\mathtt{tt}}\big)\big) \parallel\\ \quad \forall w_3,u\big(\mathsf{snd}(w_3,u)\otimes\{w_3{:}y\}\to \overline{\mathsf{rcv}(y,u)} \parallel \overline{\mathsf{snd}(u,\mathtt{tt})} \parallel\\ \qquad \forall w_4\big(\mathsf{rcv}(w_4,\mathtt{tt})\otimes\{w_4{:}u\}\to \overline{\mathtt{tt}}\big)\big)\big)\end{array}$$

where, using the semantics in Fig. 6, it can be shown that:

$$\begin{array}{rl}[\![P_1]\!] \longrightarrow^2_\ell \exists x,y,w,z.\big(\,!\,\overline{\{x{:}y\}} \parallel\, !\,\overline{\{w{:}z\}} \parallel & \\ \underbrace{\forall w_2,u'\big(\mathsf{snd}(w_2,u')\otimes\{w_2{:}w\}\to \overline{\mathsf{rcv}(w,u')} \parallel \overline{\mathtt{tt}}\big)}_{[\![w(u').\mathbf{0}]\!]} & \\ \underbrace{\overline{\mathsf{snd}(z,\mathtt{tt})} \parallel \forall w_4\big(\mathsf{rcv}(w_4,\mathtt{tt})\otimes\{w_4{:}z\}\to \overline{\mathtt{tt}}\big)}_{[\![z\langle\mathtt{tt}\rangle.\mathbf{0}]\!]}\big) & \end{array}$$

which can then reduce as expected.

We now show how our translation can handle non-determinism. In particular, the kind of non-determinism induced by multiple replicated servers that can interact with a single client.

Example 3 (Translating Non-Determinism) Let us consider the π program P_2 below, which is not encodable in [31]:

$$P_2 = (\boldsymbol{\nu} xy)(x\langle v_1\rangle.Q_1 \mid *\, y(z_1).Q_2 \mid *\, y(z_2).Q_3) \tag{4}$$

The translation for P_2 follows:

$$\begin{array}{rl}[\![P_2]\!] = \exists x,y.\big(\,!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{snd}(x,v_1)} \parallel \forall z(\mathsf{rcv}(z,v_1)\otimes\{x{:}z\}\to [\![Q_1]\!]) \parallel & \\ \qquad !\,\forall z_1,w_1(\mathsf{snd}(w_1,z_1)\otimes\{w_1{:}y\}\to \overline{\mathsf{rcv}(y,z_1)} \parallel [\![Q_2]\!]) \parallel & \\ \qquad !\,\forall z_2,w_2(\mathsf{snd}(w_2,z_2)\otimes\{w_2{:}y\}\to \overline{\mathsf{rcv}(y,z_2)} \parallel [\![Q_3]\!])\big) & \end{array}$$

Note that $P_2 \longrightarrow (\boldsymbol{\nu} xy)(Q_1 \mid Q_2\{^{v_1}/_{z_1}\} \mid *\, y(z_2).Q_3 \mid *\, y(z_1).Q_2) = P_2'$. Fig. 9 shows how this reduction is mimicked in `lcc`: observe that we use structural congruence twice to get a copy of process $\overline{\{x{:}y\}}$ (cf. Axiom (SC_ℓ:4) in Def. 13). The other reduction from P_2 (involving $*\, y(z_2).Q_3$) can be treated similarly.

Our next example considers a π process that implements a selection protocol, and shows how to obtain the observables of its translation. These observables can be used to showcase the behavioral equivalences in Def. 23 on translated processes (see App. A for details).

Example 4 (Translations and their observables) Let us consider process P_3, which models a simple transaction between a client and a store.

$$P_3 = (\boldsymbol{\nu} xy)(x\triangleleft \mathit{buy}.x\langle 5406\rangle.x(\mathit{inv}).\mathbf{0} \mid y\triangleright\{\mathit{buy}\colon y(w).y\langle\mathsf{invoice}\rangle.\mathbf{0}, \mathit{quit}\colon y(w').\mathbf{0}\}) \tag{5}$$

P_3 specifies a client sub-process (on the left) that wants to buy some item from a store sub-process (on the right). Intuitively, the client selects to buy, and sends its credit card number,

$$
\begin{aligned}
[\![P_2]\!] \equiv\ & \exists x, y.\big(\,!\,\overline{\{x{:}y\}} \parallel \overline{\{x{:}y\}} \parallel \overline{\mathsf{snd}(x, v_1)} \parallel \forall z(\mathsf{rcv}(z, v_1) \otimes \{x{:}z\} \to [\![Q_1]\!]) \parallel \\
& \quad !\,\forall z_1, w_1(\mathsf{snd}(w_1, z_1) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y, z_1)} \parallel [\![Q_2]\!]) \parallel \\
& \quad !\,\forall z_2, w_2(\mathsf{snd}(w_2, z_2) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, z_2)} \parallel [\![Q_3]\!])\big) \\
\longrightarrow_\ell\ & \exists x, y.\big(\,!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{rcv}(y, v_1)} \parallel \forall z(\mathsf{rcv}(z, v_1) \otimes \{x{:}z\} \to [\![Q_1]\!]) \parallel \\
& \quad [\![Q_2]\!]\{^{v_1, x}/_{z_1, w_1}\} \parallel [\![* \, y(z_1).Q_2]\!] \parallel \\
& \quad !\,\forall z_2, w_2(\mathsf{snd}(w_2, z_2) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, z_2)} \parallel [\![Q_3]\!])\big) \\
\equiv\ & \exists x, y.\big(\,!\,\overline{\{x{:}y\}} \parallel \overline{\{x{:}y\}} \parallel \overline{\mathsf{rcv}(y, v_1)} \parallel \forall z(\mathsf{rcv}(z, v_1) \otimes \{x{:}z\} \to [\![Q_1]\!]) \parallel \\
& \quad [\![Q_2]\!]\{^{v_1, x}/_{z_1, w_1}\} \parallel [\![* \, y(z_1).Q_2]\!] \parallel \\
& \quad !\,\forall z_2, w_2(\mathsf{snd}(w_2, z_2) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, z_2)} \parallel [\![Q_3]\!])\big) \\
\longrightarrow_\ell\ & \exists x, y.\big(\,!\,\overline{\{x{:}y\}} \parallel [\![Q_1]\!]\{^{y}/_{z}\} \parallel [\![Q_2]\!]\{^{v_1, x}/_{z_1, w_1}\} \parallel \\
& \quad [\![* \, y(z_1).Q_2]\!] \parallel [\![* \, y(z_2).Q_3]\!]\big) = [\![P_2']\!]
\end{aligned}
$$

Fig. 9: One possible evolution of the `lcc` translation of the π program P_2 in (4) (cf. Ex. 3).

before receiving an invoice. Dually, the store is waiting for a selection to be made. If the *buy* label is picked, the store awaits for the credit card number, before emitting an invoice.

The translation of P_3 is then given below:

$$
\begin{aligned}
[\![P_3]\!] = \exists x, y.\big(\,!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{sel}(x, \mathit{buy})} \parallel \forall u_1(\mathsf{bra}(u_1, \mathit{buy}) \otimes \{x{:}u_1\} \to \\
\overline{\mathsf{snd}(x, 5406)} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!])) \parallel \\
\forall l, w(\mathsf{sel}(w, l) \otimes \{w{:}y\} \to \overline{\mathsf{bra}(y, l)} \parallel \forall \epsilon(l = \mathit{buy} \to \\
\forall w_1, w(\mathsf{snd}(w_1, w) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y, w)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!])) \parallel \\
\forall \epsilon(l = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w').\mathbf{0}]\!])))\big)
\end{aligned}
$$

Combining Def. 15 and Def. 22, we have the following observables:

$$
\begin{aligned}
\mathcal{O}^{\mathcal{D}^\star_\pi}([\![P_3]\!]) =& \{(\exists x, y.\mathsf{sel}(x, \mathit{buy})), (\exists x, y.\mathsf{bra}(y, \mathit{buy})), (\exists x, y.\mathsf{snd}(x, 5406)), (\exists x, y.\mathtt{tt}) \\
& (\exists x, y.\mathsf{rcv}(y, 5406)), (\exists x, y.\mathsf{snd}(y, \mathsf{invoice})), (\exists x, y.\mathsf{rcv}(x, \mathsf{invoice}))\} \\
\mathcal{O}^{\mathcal{D}_\pi}([\![P_3]\!]) =& \{(\exists x, y.\mathsf{sel}(x, \mathit{buy})), (\exists x, y.\mathsf{snd}(x, 5406)), (\exists x, y.\mathsf{snd}(y, \mathsf{invoice}))\}
\end{aligned}
$$

Having introduced and illustrated our translation, we now move to establish its correctness in the sense of Def. 20.

4.2 Name Invariance, Compositionality, and Operational Completeness

First, we prove that the translation is name invariant with respect to the renaming policy in Def. 25. The proof follows by induction on the structure of process P.

Theorem 8 (Name Invariance for $[\![\cdot]\!]$) *Let P be a well-typed π process. Also, let σ be a substitution satisfying the renaming policy for $[\![\cdot]\!]$ (Def. 25(b)), and x be a variable. Then $[\![P\sigma]\!] = [\![P]\!]\sigma'$, with $\varphi_{[\![\cdot]\!]}(\sigma(x)) = \sigma'(\varphi_{[\![\cdot]\!]}(x))$ and $\sigma = \sigma'$.*

Next, we establish that $[\![\cdot]\!]$ is compositional, in the sense of Def. 20(2). The proof follows immediately from the translation definition in Fig. 8: each process in π is translated using a context in `lcc` that depends on the translation of its sub-processes. In particular, notice that parallel composition (denoted ' | ' in π) is translated homomorphically: the associated `lcc` context in that case is $C_{|}(-_1,-_2) = [-_1] \parallel [-_2]$.

Theorem 9 (Compositionality for $[\![\cdot]\!]$) *The encoding $[\![\cdot]\!]$ is compositional.*

Related to compositionality, we have the following result, which says that the translation preserves the evaluation contexts of Def. 3, which involve restriction and parallel composition. Below, we use the extension of $[\![\cdot]\!]$ to evaluation contexts, obtained by decreeing $[\![-]\!] = -$. The proof is by induction on the structure of P and a case analysis on $E[-]$.

Theorem 10 (Evaluation Contexts and $[\![\cdot]\!]$) *Let P and $E[-]$ be a well-typed π process and a π evaluation context as in Def. 3, respectively. Then we have: $[\![E[P]]\!] = [\![E]\!]\big[[\![P]\!]\big]$.*

We close this section by stating operational completeness, which holds up to barbed congruence (cf. Def. 23):

Theorem 11 (Completeness for $[\![\cdot]\!]$) *Let $[\![\cdot]\!]$ be the translation in Def. 25. Also, let P be a well-typed π program. Then, if $P \longrightarrow^* Q$ then $[\![P]\!] \stackrel{\tau}{\Longrightarrow} \cong_\ell^\pi [\![Q]\!]$.*

Proof By induction on the length of the reduction $\longrightarrow^*$, with a case analysis on the last applied rule, relying on auxiliary results to be given in § 4.3.2. For details see App. C.2. □

4.3 Operational Soundness

The most challenging part of our technical development is proving that our translation satisfies the operational soundness criterion (cf. Def. 20):

Theorem 12 (Soundness for $[\![\cdot]\!]$) *Let $[\![\cdot]\!]$ be the translation in Def. 25. Also, let P be a well-typed π program. For every S such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S$ there are Q, S' such that $P \longrightarrow^* Q$ and $S \stackrel{\tau}{\Longrightarrow} S' \cong_\ell^\pi [\![Q]\!]$.*

Our goal is to precisely identify which well-typed π program is being mimicked at any given point by a target term in `lcc`, while ensuring that such target term does not add undesired behaviors. This is a non-trivial task: programs may contain multiple redexes running at the same time, and redexes in π are mimicked in `lcc` using two-step synchronizations. Our proof draws inspiration from [41], where translated process are characterized semantically, by defining pre-processing and post-processing reduction steps, according to the effect they have over target terms and the simulation of the behavior of the source language.

We first define *target terms* for $[\![\cdot]\!]$ to set our focus on well-typed π programs:

Definition 26 (Target Terms) We define *target terms* as the set of `lcc` processes that are induced by the translation of well-typed π programs and is closed under τ-transitions: $\{S \mid [\![P]\!] \stackrel{\tau}{\Longrightarrow} S \text{ and } \vdash P\}$. We shall use $S, S', \ldots$ to range over target terms.

We start by giving a roadmap to the proof of Thm. 12 and its different ingredients. Then, these ingredients are spelled out in detail in § 4.3.2. The full proof is given in § 4.3.3.

4.3.1 Proof Roadmap

We characterize target terms using *complete* and *output observables* (Def. 22). We will first define sets of *immediate observables* (cf. Def. 29), which only contain barbs up to structural congruence, rather than to τ-transitions. We then distinguish translated processes and their so-called *intermediate redexes* (cf. Def. 30), which represent "half-steps" in the simulation of a source π synchronization.

Before detailing a proof sketch, we describe the three main ingredients in the proof.

1. *Junk processes* (Def. 28) do not add behavior in target terms (up to barbed congruence, cf. Def. 23). Recognizing junk processes simplifies the characterization of target terms, as we show that every target term that contains junk is in the same equivalence class as a target term without it (cf. Cor. 4).
2. *Invariants of target terms:* In the proof, Lem. 11 and Lem. 12 are crucial: they show how the shape of π processes can be inferred from their corresponding `lcc` translation by using immediate observables. These lemmas require us to first isolate the shape of translated programs (cf. Lem. 1), which in turn enables us to analyze the shape of target terms in Lem. 9. Once this shape has been established, Lem. 10 allows us to analyze the store of a target term after a τ-transition. By looking at the store we can identify an originating `lcc` process, which can be used to infer a corresponding π source process.
3. *A diamond property for target terms:* The last step involves analyzing the interactions between intermediate redexes and translated processes, to ensure that they do not interfere with each other. This is the content of the *diamond lemma* given as Lem. 14. Finally, Lem. 15 shows that intermediate redexes always reach the translation of a π process.

Proof Sketch for Thm. 12 By induction on n, the length of the reduction $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_1$. The base case ($n = 0$) is immediate; for the inductive step ($n > 0$) we proceed as follows. Given a redex R, we write $(\!|R|\!)^k_{\widetilde{x}\widetilde{y}}$ to denote the elements in its set of intermediate redexes (cf. Def. 30).

1. Since $n \geq 1$, there exists a target term S_0 such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_0 \longrightarrow_\ell S_1$.
2. By IH, there exist Q_0 and S'_0 such that $P \longrightarrow^* Q_0$ and $S_0 \stackrel{\tau}{\Longrightarrow} S'_0$, with $S'_0 \cong^\pi_\ell [\![Q_0]\!]$. Observe that by Lem. 15, we have that the sequence of transitions $S_0 \stackrel{\tau}{\Longrightarrow} S'_0$ is executing actions that correspond to closing labels in the labeled semantics defined in Fig. 13.
3. By Lem. 9, we have that $S_0 = C_{\widetilde{x}\widetilde{y}}[S'_1 \parallel \cdots \parallel S'_n \parallel J]$, where $S'_i = [\![R_i]\!]$ or $S'_i = (\!|R_i|\!)^k_{\widetilde{x}\widetilde{y}}$ for some R_i, $k \in \{1, 2, 3\}$.
4. We analyze the transition $S_0 \longrightarrow_\ell S_1$ in Item (1). By Item (3), S_0 has a specific shape and so will S_1. There are then two possible shapes for the transition, depending on whether one or two components evolve (we ignore junk processes using Lem. 5):
   (a) $C_{\widetilde{x}\widetilde{y}}[S'_1 \parallel \cdots \parallel S'_h \parallel \cdots \parallel S'_n] \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[S'_1 \parallel \cdots \parallel S''_h \parallel \cdots \parallel S'_n]$
   (b) $C_{\widetilde{x}\widetilde{y}}[S'_1 \parallel \cdots \parallel S'_{h_1} \parallel \cdots \parallel S'_{h_2} \parallel \cdots \parallel S'_n] \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[S'_1 \parallel \cdots \parallel S''_{h_1} \parallel \cdots \parallel S''_{h_2} \parallel \cdots \parallel S'_n]$.
5. In both cases, Lem. 11 and Lem. 12 will allow us to identify which source reduction (in Q_0) is being partially simulated by $S_0 \longrightarrow_\ell S_1$. (It is 'partial' because a π reduction is mimicked by at least two transitions in `lcc`.) Hence, we can characterize the π process Q for which $Q_0 \longrightarrow Q$.

6. We are left to show the existence of S_1', given that $S_0 \longrightarrow_\ell S_1$ and $S_0 \stackrel{\tau}{\Longrightarrow} S_0'$ (Items (1) and (2), respectively). This follows from Lem. 14, which is a diamond property for `lcc` processes induced by so-called *closing* and *opening* labeled the transitions (cf. Def. 32) and the shape of S_0 identified in Item (3), which is preserved in S_1 by Item (4). These facts combined ensure that the same transition from S_0 to S_1 can take place from S_0'. Therefore, there is an S_1' such that $S_0' \longrightarrow_\ell S_1'$ and that the same transitions from S_0 to S_0' can be made by S_1. Therefore, $S_1 \stackrel{\tau}{\Longrightarrow} S_1'$.
7. Finally, since $S_0' \cong_\ell^\pi [\![Q_0]\!]$ (IH, Item (2)) and by the reduction and transition identified in Items (5) and (6), respectively, we can infer that $S_1' \cong_\ell^\pi [\![Q]\!]$.

Using this proof sketch as a guide, we now introduce in detail all the ingredients of the proof.

4.3.2 Proof Ingredients

The Shape of Translated Programs The *enablers* of a process intuitively represent all the necessary endpoint connections required for reduction:

Definition 27 (Enablers for π Processes) Let P be a π process. We say that the vectors of variables $\widetilde{x}, \widetilde{y}$ *enable* P if there is some P' such that $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})P \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})P'$.

The enablers of a process lead to an evaluation context $E[-] = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(-)$ (cf. Def. 3). Translating the context $E[-]$ is so common that we introduce the following notation for it:

Notation 13 *Let $E[-] = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(-)$ be a π evaluation context, as in Def. 3. We will write $C_{\widetilde{x}\widetilde{y}}[-]$ to denote the translation of E:*

$$[\![E[-]]\!] \stackrel{\text{def}}{=} \exists \widetilde{x}, \widetilde{y}.\big(\,! \overline{\bigotimes_{\substack{x_i \in \widetilde{x},\\ y_i \in \widetilde{y}}} \{x_i{:}y_i\}} \parallel -\big)$$

We restrict our attention to well-typed programs (Not. 3). Programs encompass "complete" protocol implementations, i.e., processes that contain all parties and sessions required in the system. Considering programs is also convenient because their syntax facilitates reasoning about their behavior. The first invariant of our translation concerns the shape of translated π programs; it follows directly from Def. 10 and Fig. 8.

Lemma 1 (Translated Form of a Program) *Let P be a well-typed π program (Not. 3). Then*

$$[\![P]\!] \equiv C_{\widetilde{x}\widetilde{y}}[[\![R_1]\!] \parallel \cdots \parallel [\![R_n]\!]]$$

where $n \geq 1$ and $x_1, \ldots, x_n \in \widetilde{x}$, $y_1, \ldots, y_n \in \widetilde{y}$. Note that each R_i (with $1 \leq i \leq n$) is a pre-redex (Def. 9) or a conditional process in P.

Junk Processes Translations typically induce *junk processes* that do not add any meaningful (source) behavior to translated processes. In our setting, junk processes behave like $\overline{\mathtt{tt}}$, modulo $\cong_\ell^\pi$ (i.e., they do not add any information). Junk can be characterized syntactically: they are "leftovers" of the translation of conditional and branching constructs.

Definition 28 (Junk) Let P and J be `lcc` processes. Also, let b be a boolean and l_i, l_j be two distinct labels. We say that J is junk, if it belongs to the following grammar:

$$J, J' ::= \overline{\mathtt{tt}} \;\mid\; \forall\epsilon((b = \neg b) \to P) \;\mid\; \forall\epsilon((l_i = l_j) \to P) \;\mid\; J \parallel J'$$

The following statements say that junk processes cannot introduce any observable behavior in translated processes. This entails showing $J \cong^{\pi}_{\ell} \overline{\mathtt{tt}}$, for any J. The proof is divided in three statements: (1) we show that no constraint in the session constraint system (Def. 21) allows a junk process to reduce; (2) we show that junk processes cannot reduce and that $\mathcal{O}^{\mathcal{D}_\pi}(J) = \mathcal{O}^{\mathcal{D}_\pi}(\overline{\mathtt{tt}})$; (3) we prove that J and $\overline{\mathtt{tt}}$ are behaviorally equivalent under any $\mathcal{D}_\pi\mathcal{C}$-context (cf. Def. 16 and Def. 22). Their respective proofs can be found in App. C.1.

Lemma 2 *Let J be junk. Then: (1) $J \not\xrightarrow{\mathcal{I}}_{\ell}$ (and) (2) there is no $c \in \mathcal{C}$ (cf. Def. 21) such that $J \parallel \overline{c} \xrightarrow{\tau}_{\ell}$.*

Lemma 3 (Junk Observables) *For every junk process J and every $\mathcal{D}_\pi\mathcal{C}$-context $C[-]$, we have that: (1) $\mathcal{O}^{\mathcal{D}_\pi}(J) = \emptyset$ (and) (2) $\mathcal{O}^{\mathcal{D}_\pi}(C[J]) = \mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}])$.*

Lemma 4 (Junk Behavior) *For every junk J, every $\mathcal{D}_\pi\mathcal{C}$-context $C[-]$, and every process P, we have $C[P \parallel J] \approx^{\pi}_{\ell} C[P]$.*

The following corollary follows directly from Lem. 4 and Def. 18:

Corollary 2 *For every junk J (cf. Def. 28) and every* `lcc` *process P, we have $P \parallel J \cong^{\pi}_{\ell} P$.*

The following lemma says that non-trivial junk processes only appear as a byproduct of the translation of branching/selection processes and conditionals; other forms of synchronization do not generate junk.

Lemma 5 (Occurrences of Junk) *Let R be a redex (Def. 9).*

1. *If $R = x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$, with $j \in I$ then:*
   $[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}{}^{3}_{\ell} \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!] \parallel J\big)$, *where*
   $J = \prod_{i\in I'} \forall\epsilon(l_j = l_i \to [\![Q_i]\!])$, *with $I' = I \setminus \{j\}$, and*
   $\exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!] \parallel J\big) \cong^{\pi}_{\ell} \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!]\big)$.
2. *If $R = b?\, P_1 : P_2$, $b \in \{\mathtt{tt}, \mathtt{ff}\}$, then:*
   $[\![R]\!] \xrightarrow{\tau}_{\ell} [\![P_i]\!] \parallel J$, *$i \in \{1, 2\}$ with $J = \forall\epsilon(b = \neg b \to [\![P_j]\!])$, $j \neq i$, and*
   $[\![P_i]\!] \parallel J \cong^{\pi}_{\ell} [\![P_i]\!]$.
3. *If $R = x\langle v\rangle.P \mid y(z).Q$ then*
   $[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}{}^{2}_{\ell} \cong^{\pi}_{\ell} \exists x, y.\big([\![P]\!] \parallel [\![Q\{v/z\}]\!] \parallel J\big)$ *with $J = \overline{\mathtt{tt}}$.*
4. *If $R = x\langle v\rangle.P \mid *\, y(z).Q$ then:*

$$[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}{}^{2}_{\ell} \cong^{\pi}_{\ell} \exists x, y.\big([\![P]\!] \parallel [\![Q\{v/z\}]\!] \parallel [\![*\, y(z).P]\!] \parallel \overline{\mathtt{tt}}\big)$$

Proof Each item follows from the definition of $[\![\cdot]\!]$ (cf. Def. 25 and Fig. 8). Items (1) and (2) concern reductions that induce junk (no junk is generated in Items (3) and (4)); those cases rely on the definition of $\cong^{\pi}_{\ell}$ (cf. Def. 23) and Cor. 2. For details see App. C.1. □

Invariants for Translated Pre-Redexes and Redexes Intuitively, the set of immediate observables of an `lcc` process denotes the current store of a process (i.e., all the constraints that can be consumed in a single transition).

Definition 29 (Immediate Observables of an `lcc` Process) Let P be an `lcc` process and $\mathcal{C}$ be a set of constraints. The set of *immediate observables* of P up to $\mathcal{C}$, denoted $\mathcal{I}^{\mathcal{C}}(P)$, is defined in Fig. 10.

$$\mathcal{I}^{\mathcal{C}}(\overline{c}) \stackrel{\text{def}}{=} \{c\}, \text{ if } c \in \mathcal{C} \qquad \mathcal{I}^{\mathcal{C}}(!\,P) \stackrel{\text{def}}{=} \begin{cases} \emptyset & \text{if } P = \forall \widetilde{x}(c \to P) \\ \{c\} & \text{if } P = \overline{c} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathcal{I}^{\mathcal{C}}(\forall \widetilde{x}(c \to P)) \stackrel{\text{def}}{=} \emptyset \qquad \mathcal{I}^{\mathcal{C}}(\exists \widetilde{z}.(P)) \stackrel{\text{def}}{=} \{\exists \widetilde{z}.c \mid c \in \mathcal{I}^{\mathcal{C}}(P)\}$$

$$\mathcal{I}^{\mathcal{C}}(P + Q) \stackrel{\text{def}}{=} \mathcal{I}^{\mathcal{C}}(P) \cup \mathcal{I}^{\mathcal{C}}(Q) \qquad \mathcal{I}^{\mathcal{C}}(P \parallel Q) \stackrel{\text{def}}{=} \mathcal{I}^{\mathcal{C}}(P) \cup \mathcal{I}^{\mathcal{C}}(Q)$$

Fig. 10: Immediate observables (cf. Def. 29).

It suffices to define immediate observables over a subset of `lcc` processes. We leave out processes that are not induced by the translation, such as $!(P \parallel P)$ or $!(P + P)$. The definition is parametric in $\mathcal{C}$, which we instantiate with the set $\mathcal{D}^{\star}_{\pi}$ of complete observables (cf. Def. 22).

We now introduce so-called *invariants* for the translation, i.e., properties that hold for every target term. Based on source π processes, we will define these invariants bottom-up, starting from translations of pre-redexes (i.e., a prefixed process that does not contain parallel composition at the top-level, cf. Def. 9), redexes, and translated programs.

The following invariant clarifies how the immediate observables of the translation of some pre-redex P give information about the nature of P itself (cf. App. C.3).

Lemma 6 (Invariants of $[\![\cdot]\!]$ for Pre-Redexes and the Inaction) *Let P be a pre-redex or the inactive process in π. Then the following properties hold:*

1. *If $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}([\![P]\!]) = \{\mathsf{snd}(x, v)\}$ then $P = x\langle v\rangle.P_1$, for some P_1.*
2. *If $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}([\![P]\!]) = \{\mathsf{sel}(x, l)\}$ then $P = x \triangleleft l.P_1$, for some P_1.*
3. *If $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}([\![P]\!]) = \{\mathtt{tt}\}$ then $P = \mathbf{0}$.*
4. *If $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}([\![P]\!]) = \emptyset$ then $P = \diamond\, y(z).P_1$ (cf. Not. 2) or $P = x \triangleright \{l_1 : P_i\}_{i \in I}$, for some P_i. Moreover, $[\![P]\!] \not\xrightarrow{\tau}_{\ell}$.*

When the immediate observables do not provide enough information on the shape of a pre-redex (as in Lem. 6(4)), we can characterize the minimal parallel context that induces immediate observables (cf. App. C.3).

Lemma 7 (Invariants of $[\![\cdot]\!]$ for Input-Like Pre-Redexes) *Let P be a pre-redex such that $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}([\![P]\!]) = \emptyset$. Then one of the following holds:*

1. *If $[\![P]\!] \parallel \overline{\mathsf{sel}(x, l_j) \otimes \{y{:}x\}} \xrightarrow{\tau}_{\ell} S$ then $\mathsf{bra}(y, l_j) \in \mathcal{I}^{\mathcal{D}^{\star}_{\pi}}(S)$ and $P = y \triangleright \{l_i : P_i\}_{i \in I}$, with $j \in I$.*
2. *If $[\![P]\!] \parallel \overline{\mathsf{snd}(x, v) \otimes \{y{:}x\}} \xrightarrow{\tau}_{\ell} S$ then $\mathsf{rcv}(y, v) \in \mathcal{I}^{\mathcal{D}^{\star}_{\pi}}(S)$ and $P = \diamond\, y(z).P_1$.*

The *intermediate* `lcc` *redexes* of a communicating redex (cf. Def. 9) are processes obtained through the transitions of a target term:

$$(\!|R|\!)^1_{\widetilde{x}\widetilde{y}} \stackrel{\text{def}}{=} \begin{cases} \overline{\mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{z{:}x\} \to \\ [\![P]\!]) \parallel [\![Q\{v/x\}]\!] & \text{if } R = x\langle v\rangle.P \mid y(z).Q \\ \overline{\mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{z{:}x\} \to \\ [\![P]\!]) \parallel [\![Q\{v/x\}]\!] \parallel [\![*y(w).Q]\!] & \text{if } R = x\langle v\rangle.P \mid *y(z).Q \\ \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to \\ [\![P]\!]) \parallel \forall \epsilon(l_j = l_j \to [\![Q_j]\!]) \parallel J & \text{if } R = x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I} \end{cases}$$

$$(\!|R|\!)^2_{\widetilde{x}\widetilde{y}} \stackrel{\text{def}}{=} \begin{cases} [\![P]\!] \parallel \forall \epsilon(l_j = l_j \to [\![Q_j]\!]) \parallel J & \text{if } R = x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I} \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$(\!|R|\!)^3_{\widetilde{x}\widetilde{y}} \stackrel{\text{def}}{=} \begin{cases} \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to \\ [\![P]\!]) \parallel [\![Q_j]\!] \parallel J & \text{if } R = x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I} \\ \text{undefined} & \text{otherwise} \end{cases}$$

Fig. 11: Elements in the set of intermediate redexes (cf. Not. 14)

Definition 30 (Intermediate Redexes) Let R be a communicating redex in π enabled by $\widetilde{x}, \widetilde{y}$. The *set of intermediate* `lcc` *redexes* of R, denoted $\{\!|[\![R]\!]|\!\}$, is defined as follows:

$$\begin{aligned} \{\!|[\![x\langle v\rangle.P \mid y(z).Q]\!]|\!\} &\stackrel{\text{def}}{=} \{\overline{\mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{z{:}x\} \to [\![P]\!]) \parallel [\![Q\{v/z\}]\!]\} \\ \{\!|[\![x\langle v\rangle.P \mid *y(z).Q]\!]|\!\} &\stackrel{\text{def}}{=} \{\overline{\mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{z{:}x\} \to [\![P]\!]) \parallel [\![Q\{v/z\}]\!] \parallel \\ &\qquad [\![*y(w).Q]\!]\} \\ \{\!|[\![x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}]\!]|\!\} &\stackrel{\text{def}}{=} \{\overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to [\![P]\!]) \parallel \\ &\qquad \forall \epsilon(l_j = l_j \to [\![Q_j]\!]) \parallel J, [\![P]\!] \parallel \forall \epsilon(l_j = l_j \to [\![Q_j]\!]) \parallel J, \\ &\qquad \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to [\![P]\!]) \parallel \\ &\qquad [\![Q_j]\!] \parallel J \,\big|\, J \text{ as in Def. 28}\} \end{aligned}$$

Thus, the set of intermediate redexes is a singleton, except for the translation of selection and branching. We introduce a convenient notation for these intermediate redexes:

Notation 14 *We will denote the elements of* $\{\!|[\![R]\!]|\!\}$ *as* $(\!|R|\!)^k_{\widetilde{x}\widetilde{y}}$*, with* $k \in \{1,2,3\}$ *as in Fig. 11.*

This notation aims to clarify the behavior of intermediate redexes, particularly in the case of the selection and branching. Their use will become much more apparent in the following invariant, which describes how translated redexes interact (see App. C.3 for proof details).

Lemma 8 (Invariants for Redexes and Intermediate Redexes) *Let R be a redex enabled by $\widetilde{x}, \widetilde{y}$, such that $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})R'$. Then one of the following holds:*

1. *If $R \equiv_\pi v?\, P_1 : P_2$ and $v \in \{\mathtt{tt}, \mathtt{ff}\}$, then $[\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R]\!] \xrightarrow{\tau}_\ell \cong^\pi_\ell (\boldsymbol{\nu}\widetilde{x}\widetilde{y})[\![P_i]\!]$, with $i \in \{1,2\}$.*
2. *If $R \equiv_\pi x\langle v\rangle.P \mid \diamond\, y(w).Q$, then $[\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R]\!] \longrightarrow_\ell \equiv C_{\widetilde{x}\widetilde{y}}[(\!|R|\!)^1_{\widetilde{x}\widetilde{y}}] \longrightarrow_\ell \cong^\pi_\ell [\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R']\!]$.*
3. *If $R \equiv_\pi x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$, with $j \in I$, then we have the reductions in Fig. 12.*

A corollary of Lem. 8 and Def. 30 is that every intermediate redex reduces to some target term:

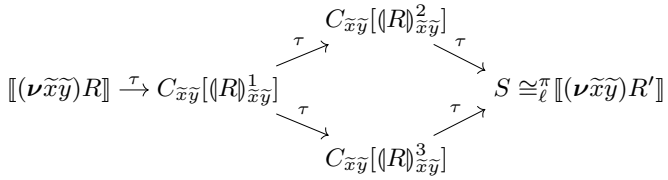


Fig. 12: Lem. 8(3).

Corollary 3 *For every intermediate redex $S \in \{\!|[R]|\!\}$ (cf. Def. 30), there exist some π process R' and some $k \in \{1,2\}$ such that $S \longrightarrow^{k}_{\ell} [\![R']\!]$ and $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})R'$.*

We introduce some useful notation for the set of immediate observables of a target term:

Notation 15 *We define the following conventions:*

- *$\mathcal{I}_S$ will be a short-hand notation for the set $\mathcal{I}^{\mathcal{D}^{\star}_{\pi}}(S)$ (cf. Def. 29).*
- *By a slight abuse of notation, we will write $c_{\widetilde{z}} \in \mathcal{I}_S$ instead of $\exists \widetilde{z}.c \in \mathcal{I}_S$.*

This notation conveniently captures the constraints that are consumed as a result of a τ-transition. In turn, such consumed constraints will allow us to recognize which π process is simulated by the translation. In particular, every τ-transition of a target term modifies the store (and the immediate observables) either (i) by adding new constraints (if the transition is induced by the translation of conditionals and labeled choices) or (ii) by consuming some existing constraints (if the transition is induced by other kinds of source synchronizations). Case (i) is formalized by Lem. 11 and case (ii) is covered by Lem. 12.

Invariants for Translated Well-Typed Programs Given a program P, we say that R_k is a (pre)redex *reachable* from P if $P \longrightarrow^{*} (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(R_k \mid R)$, for some R.

The following lemma will allow us to determine the structure of a given target term. It states that any target term corresponds to the parallel composition of the translation of processes, intermediate redexes and junk, all enclosed within a context that provides the required co-variable constraints. The proof is by induction on the length of the transition of the encoded program (cf. App. C.4).

Lemma 9 *Let P be a well-typed program. If $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S$ then*

$$S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n \parallel J]$$

where $n \geq 1$, J is some junk, and for all $i \in \{1,\ldots,n\}$ we have $U_i = \overline{\texttt{tt}}$ or one the following:

1. *$U_i = [\![R_k]\!]$, where R_k is a conditional redex (cf. Def. 9) reachable from P;*
2. *$U_i = [\![R_k]\!]$, where R_k is a pre-redex reachable from P;*
3. *$U_i \in \{\!|[R_k \mid R_j]|\!\}$ (cf. Def. 30), where redex $R_k \mid R_j$ is reachable from P.*

Observe that this lemma is not enough to prove completeness because we have not yet analyzed the interactions between intermediate processes and the translations of pre-redexes in target terms.

The next lemma provides two insights: first, it gives a precise characterization of a target term whenever constraints are being added to the store and there is no constraint consumption. Second, it captures the fact that τ-transitions consume one constraint at a time.

Lemma 10 *Let P be a well-typed π program. Then, for every S, S' such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \stackrel{\tau}{\longrightarrow}_\ell S'$ one of the following holds:*

(a) $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$ (cf. Not. 15) and one of the following holds:
  (1) $S \equiv C_{\widetilde{xy}}[[\![b?\, P_1 : P_2]\!] \parallel U]$ and $S' = C_{\widetilde{xy}}[[\![P_i]\!] \parallel U]$, with $i \in \{1,2\}$;
  (2) $S \equiv C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{xy}} \parallel U]$ and $S' = C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{xy}} \parallel U]$;
  (3) $S \equiv C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{xy}} \parallel U]$ and $S' = C_{\widetilde{xy}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U]$.
(b) $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$ and $|\mathcal{I}_S \setminus \mathcal{I}_{S'}| = 1$.

Proof We first use Lem. 9 to characterize every parallel sub-process U_i of S; then, by a case analysis on the shape of the U_i that originated the transition $S \stackrel{\tau}{\longrightarrow}_\ell S'$ we show how each case falls under either (a) or (b). For details see App. C.4. □

Let $\gamma \in \{\mathsf{rcv}, \mathsf{snd}, \mathsf{sel}, \mathsf{bra}\}$ denote a predicate in Fig. 7. The next lemma formalizes the following fact: for any variable x, a target term S will never contain a sub-process such as $\overline{\gamma_1(x,v)} \parallel \overline{\gamma_2(x,v')}$. That is, the constraints added to the store at any point during the execution of a target term are *unique* with respect to x. This is where the absence of output races in source processes, ensured by our type system, plays a key rôle.

Proposition 1 *Suppose S is a target term (cf. Def. 26).*

1. *$S \equiv C_{\widetilde{xy}}[\overline{c_1} \parallel \cdots \parallel \overline{c_n} \parallel Q_1 \parallel \cdots \parallel Q_k]$ with $n, k \geq 1$, where every $c_j = \gamma_j(x_j, m_j)$ (with $1 \leq j \leq n$), for some value or label m_j, and every Q_i (with $1 \leq i \leq k$) is an abstraction (possibly replicated).*
2. *For every $i, j \in \{1, \ldots, n\}$, $i \neq j$ implies $c_i = \gamma_i(x_i, m_i)$, $c_j = \gamma_j(x_j, m_j)$, and $x_i \neq x_j$.*

Proof Part (1) follows immediately by Def. 25 and Lem. 9. Part (2) is proven by contradiction, exploiting that well-typed programs do not contain output races (cf. Thm. 4), compositionality of $[\![\cdot]\!]$, and that by construction $[\![\cdot]\!]$ ensures that target terms add input-like constraints to the store only once a corresponding output-like constraint has been consumed; see App. C.4 for details. □

As already discussed, in mimicking the behavior of a π process, the store of its corresponding target process in `lcc` may either add or consume constraints:

- Lem. 11, given below, covers the case where a transition adds information to the store: by Lem. 10(a) the target term must then correspond to either (i) the translation of a conditional redex or (ii) an intermediate redex of a branching/selection interaction.
- Lem. 12 covers the case where the transition consumes information in the store (cf. by Lem. 10(b)).

As such, Lem. 11 and Lem. 12 cover the complete spectrum of possibilities for target terms. The first invariant is proven by induction on the length of the reduction (cf. App. C.4).

Lemma 11 (Invariants of Target Terms (I): Adding Information) *Let P be a well-typed π program. For any S, S' such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \stackrel{\tau}{\longrightarrow}_\ell S'$ and $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$ (cf. Not. 15) one of the following holds, for some U:*

1. *$S \equiv C_{\widetilde{z}}[[\![b?\, P_1 : P_2]\!] \parallel U \parallel J_1]$ and $S' = C_{\widetilde{z}}[[\![P_i]\!] \parallel \forall\epsilon(b = \neg b \to P_j) \parallel U \parallel J_1]$ with $i, j \in \{1,2\}, i \neq j$;*

2. $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![x \triangleleft l_j.P' \mid y \triangleright \{l_i\ Q_i\}_{i\in I}]\!] \parallel U \parallel J_1]$ *and either:*
   (a) *All of the following hold:*
      (i) $S_0 \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1] \stackrel{\tau}{\longrightarrow}_\ell S$,
      (ii) $S = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$ *(and)*
      (iii) $S' = C_{\widetilde{x}\widetilde{y}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U \parallel J_1 \parallel J_2]$.
   (b) *All of the following hold:*
      (i) $S_0 \stackrel{\tau}{\longrightarrow}_\ell S = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$,
      (ii) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$ *(and)*
      (iii) $S' \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U \parallel J_1 \parallel J_2]$.
   where $J_2 = \prod_{k\in I\setminus\{j\}} \forall\epsilon(l_j = l_k \to [\![P_k]\!])$.

We state our next invariant. Notice that Lem. 10(b) clarifies the behavior of the immediate observables (cf. Def. 29) in a single transition whenever a constraint has been consumed. The proof is by induction on the length of the `lcc` transition (cf. App. C.4).

Lemma 12 (Invariants of Target Terms (II): Consuming Information) *Let P be a well-typed π program. For any S, S' such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \stackrel{\tau}{\longrightarrow}_\ell S'$ and $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$ the following holds, for some U:*

(1) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{snd}(x_1, v)\}$ then all the following hold:
   (a) $S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![x_1\langle v\rangle.P_1 \mid \diamond\, y_1(z).P_2]\!] \parallel U]$;
   (b) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid \diamond\, y_1(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;
   (c) $S' \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_2\{v/z\}]\!] \parallel S'' \parallel U]$, where $S'' = *\, [\![y(z).P_2]\!]$ or $S'' = \overline{\mathtt{tt}}$.

(2) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{rcv}(x_1, v)\}$ then there exists S_0 such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_0 \stackrel{\tau}{\longrightarrow}_\ell S$ and all of the following hold:
   (a) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![y_1\langle v\rangle.P_1 \mid \diamond\ x_1(z).P_2]\!] \parallel U]$;
   (b) $S = C_{\widetilde{x}\widetilde{y}}[(\!|y_1\langle v\rangle.P_1 \mid \diamond\, x_1(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;
   (c) $S' = C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_2\{v/z\}]\!] \parallel S'_1 \parallel U]$, where $S'_1 = *\, [\![y(z).P_2]\!]$ or $S'_1 = \overline{\mathtt{tt}}$.

(3) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{sel}(x_1, l_j)\}$ then all of the following hold:
   (a) $S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![x_1 \triangleleft l.P_1 \mid y_1 \triangleright \{l_i : P_i\}_{i\in I}]\!]U]$;
   (b) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x_1 \triangleleft l.P_1 \mid y_1 \triangleright \{l_i : P_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;
   (c) $S_1 \stackrel{\tau}{\longrightarrow}^2_\ell \cong^\pi_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_j]\!] \parallel U']$, with $U' \equiv U \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_h]\!])$.

(4) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{bra}(x, l_j)\}$, then there exists $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![y \triangleleft l_j.Q \mid x \triangleright \{l_i\ Q_i\}_{i\in I}]\!] \parallel U]$ such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_0$ and either:
   (a) All of the following hold:
      (i) $S_0 \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.Q \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U] \stackrel{\tau}{\longrightarrow}_\ell S$,
      (ii) $S = C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.Q \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel U]$ (and)
      (iii) $S' = C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![Q \mid Q_j]\!] \parallel U']$.
   (b) All of the following hold:
      (i) $S_0 \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.P \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U] \equiv S$,
      (ii) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.P \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel U]$ (and)
      (iii) $S' \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![P \mid Q_j]\!] \parallel U']$.
   with $U' \equiv U \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_h]\!])$.

By combining previous results we obtain the following corollary, which allows us to remove junk processes from any target term.

Corollary 4 *Let P be a well-typed π program. If $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S$ then there exist S' and J such that $S = C_{\widetilde{x}\widetilde{y}}[S' \parallel J] \cong^\pi_\ell C_{\widetilde{x}\widetilde{y}}[S']$.*

Proof Since P is well-typed, by Lem. 1, $[\![P]\!] = C_{\widetilde{x}\widetilde{y}}[[\![P']\!]]$. By applying Lem. 11 and Lem. 12, we know that for every S, such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S$ it holds that $S = C_{\widetilde{x}\widetilde{y}}[S' \parallel J]$, where $S' = U_1 \parallel \cdots \parallel U_n$, $n \geq 1$, with $U_i = [\![R_i]\!]$ for some π pre-redex R_i or $U_i = (\!|R_i|\!)^k_{\widetilde{x}\widetilde{y}}$, for some π pre-redex R_i and $k \in \{1,2,3\}$. Finally, by Cor. 2, we can conclude that $C_{\widetilde{x}\widetilde{y}}[S' \parallel J] \cong^{\pi}_{\ell} C_{\widetilde{x}y}[S']$. □

A Diamond Property for Target Terms We now move to establish a *diamond property* over target terms. This property, given by Lem. 15, concerns τ-transitions originating from the intermediate processes of the same target term (cf. Def. 26). Informally speaking, it says that τ-transitions originated from intermediate processes that reach the translation of some π process do not preclude the execution of translated terms. First, we illustrate how our translation captures the non-determinism allowed by typing in π.

Example 5 Let us recall process P_2 from Ex. 3, which is well-typed:

$$P_2 = (\boldsymbol{\nu} xy)(x\langle v_1\rangle.Q_1 \mid *\, y(z_1).Q_2 \mid *\, y(z_2).Q_3)$$

Process P_2 is not confluent if $Q_2 \neq Q_3$ since

$$P_2 \longrightarrow (\boldsymbol{\nu} xy)(Q_1 \mid Q_2\{^{v_1}/_{z_1}\} \mid *\, y(z_2).Q_3 \mid *\, y(z_1).Q_2)$$

but also $P_2 \longrightarrow (\boldsymbol{\nu} xy)(Q_1 \mid *\, y(z_1).Q_2 \mid Q_3\{^{v_1}/_{z_2}\} \mid *\, y(z_2).Q_3)$. We have:

$$\begin{aligned}[\![P_2]\!] = C_{xy}[\overline{\mathsf{snd}(x, v_1)} \parallel \forall z'\big(\mathsf{rcv}(z', v_1) \otimes \{x{:}z'\} \to [\![Q_1]\!]\big) \parallel \\ !\,\forall z_1, w\big(\mathsf{snd}(w, z_1) \otimes \{y{:}w\} \to \overline{\mathsf{rcv}(y, z_1)} \parallel [\![Q_2]\!]\big) \parallel \\ !\,\forall z_2, w'\big(\mathsf{snd}(w', z_2) \otimes \{y{:}w'\} \to \overline{\mathsf{rcv}(y, z_2)} \parallel [\![Q_3]\!]\big)]\end{aligned}$$

and the following transitions are possible:

$$\begin{aligned}[\![P_2]\!] \longrightarrow_\ell C_{xy}[\forall z'\big(\mathsf{rcv}(z', v_1) \otimes \{x{:}z'\} \to [\![Q_1]\!]\big) \parallel \\ \overline{\mathsf{rcv}(y, v_1)} \parallel [\![Q_2]\!]\{^{v_1}/_{z_1}\} \parallel !\,\forall z_2, w'\big(\mathsf{snd}(w', z_2) \otimes \{y{:}w'\} \to [\![Q_3]\!]\big) \parallel \\ !\,\forall z_1, w\big(\mathsf{snd}(w, z_1) \otimes \{y{:}w\} \to [\![Q_2]\!]\big)] \\ [\![P_2]\!] \longrightarrow_\ell C_{xy}[\forall z'\big(\mathsf{rcv}(z', v_1) \otimes \{x{:}z'\} \to [\![Q_1]\!]\big) \parallel \\ \overline{\mathsf{rcv}(y, v_1)} \parallel [\![Q_3]\!]\{^{v_1}/_{z_2}\} \parallel !\,\forall z_1, w\big(\mathsf{snd}(w, z_1) \otimes \{y{:}w\} \to [\![Q_2]\!]\big) \parallel \\ !\,\forall z_2, w'\big(\mathsf{snd}(w', z_2) \otimes \{y{:}w'\} \to [\![Q_3]\!]\big)]\end{aligned}$$

The resulting processes unequivocally correspond to the following intermediate processes, respectively:

$$\begin{aligned}\exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel (\!|x\langle v_1\rangle.Q_1 \mid *\, y(z_1).Q_2|\!)^1_{xy} \parallel [\![*\, y(z_2).Q_3]\!]\big) = S_1 \\ \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel (\!|x\langle v_1\rangle.Q_1 \mid *\, y(z_2).Q_3|\!)^1_{xy} \parallel [\![*\, y(z_1).Q_2]\!]\big) = S_1'\end{aligned}$$

S_1 and S_1' specify a 'committed' state in which only one process can consume constraint $\mathsf{rcv}(y, v_1)$, which forces the translation to finish the synchronization in the translation of the correct source process.

The diamond property exploits the following notation, which distinguishes τ-transitions depending on the action that originates it. For example, a transition that simulates the first part of a synchronization between endpoints x, y will be denoted $\xrightarrow{\mathtt{IO}(x,y)}_\ell$; the completion of such synchronization is represented by transition $\xrightarrow{\mathtt{IO}_1(x,y)}_\ell$. Formally, we have:

$$\lfloor \text{IO} \rfloor\ C_{\widetilde{xy}}[[\![x\langle v\rangle.P_1]\!] \parallel [\![y(z).P_2]\!] \parallel U] \xrightarrow{\texttt{IO}(x,y)}_\ell C_{\widetilde{xy}}[(\!|x\langle v\rangle.P_1 \mid y(z).P_2|\!)^1_{xy} \parallel U]$$

$$\lfloor \text{RP} \rfloor\ C_{\widetilde{xy}}[[\![x\langle v\rangle.P_1]\!] \parallel [\![* y(z).P_2]\!] \parallel U] \xrightarrow{\texttt{RP}(x,y)}_\ell C_{\widetilde{xy}}[(\!|x\langle v\rangle.P_1 \mid * y(z).P_2|\!)^1_{xy} \parallel U]$$

$$\lfloor \text{SL} \rfloor\ C_{\widetilde{xy}}[[\![x \lhd l_j.P_1]\!] \parallel [\![y \rhd \{l_i : P_i\}_{i\in I}]\!] \parallel U] \xrightarrow{\texttt{SL}(x,y)}_\ell C_{\widetilde{xy}}[(\!|x \lhd l_j.P_1 \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^1_{xy} \parallel U]$$

$$\lfloor \text{CDT} \rfloor\ C_{\widetilde{xy}}[[\![\texttt{tt}?\, P_1 : P_2]\!] \parallel U] \xrightarrow{\texttt{CD}(-)}_\ell C_{\widetilde{xy}}[[\![P_1]\!] \parallel \forall\epsilon(\texttt{tt} = \texttt{ff} \to [\![P_2]\!]) \parallel U]$$

$$\lfloor \text{CDF} \rfloor\ C_{\widetilde{xy}}[[\![\texttt{ff}?\, P_1 : P_2]\!] \parallel U] \xrightarrow{\texttt{CD}(-)}_\ell C_{\widetilde{xy}}[[\![P_2]\!] \parallel \forall\epsilon(\texttt{ff} = \texttt{tt} \to [\![P_1]\!]) \parallel U]$$

$$\lfloor \text{IO1} \rfloor\ C_{\widetilde{xy}}[(\!|x\langle v\rangle.P_1 \mid y(z).P_2|\!)^1_{xy} \parallel U] \xrightarrow{\texttt{IO}_1(x,y)}_\ell C_{\widetilde{xy}}[[\![P_1]\!] \parallel [\![P_2]\!]\{^v/_z\} \parallel U]$$

$$\lfloor \text{RP1} \rfloor\ C_{\widetilde{xy}}[(\!|x\langle v\rangle.P_1 \mid * y(z).P_2|\!)^1_{xy} \parallel U] \xrightarrow{\texttt{RP}_1(x,y)}_\ell C_{\widetilde{xy}}[[\![P_1]\!] \parallel [\![P_2]\!]\{^v/_z\} \parallel [\![* y(z).P_2]\!] \parallel U]$$

$$\lfloor \text{SL1} \rfloor\ C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^1_{xy} \parallel U] \xrightarrow{\texttt{SL}_1(x,y)}_\ell C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^2_{xy} \parallel U]$$

$$\lfloor \text{SL2} \rfloor\ C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^1_{xy} \parallel U] \xrightarrow{\texttt{SL}_1(x,y)}_\ell C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^3_{xy} \parallel U]$$

$$\lfloor \text{SL3} \rfloor\ \frac{J = \prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to \overline{\mathsf{bra}(y, l_j)} \parallel [\![P_i]\!])}{C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^2_{xy} \parallel U] \xrightarrow{\texttt{SL}_2(x,y)}_\ell C_{\widetilde{xy}}[[\![P]\!] \parallel [\![P_j]\!] \parallel J \parallel U]}$$

$$\lfloor \text{SL4} \rfloor\ \frac{J = \prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to \overline{\mathsf{bra}(y, l_j)} \parallel [\![P_i]\!])}{C_{\widetilde{xy}}[(\!|x \lhd l_j.P \mid y \rhd \{l_i : P_i\}_{i\in I}|\!)^3_{xy} \parallel U] \xrightarrow{\texttt{SL}_3(x,y)}_\ell C_{\widetilde{xy}}[[\![P]\!] \parallel [\![P_j]\!] \parallel J \parallel U]}$$

Fig. 13: Labeled Transitions for $[\![\cdot]\!]$ (cf. Def. 31).

Definition 31 (Labeled τ-Transitions for Target Terms) Let S be a target term as in Def. 26). Also, let $\mathfrak{L} = \{\texttt{IO}, \texttt{SL}, \texttt{RP}, \texttt{CD}, \texttt{IO}_1, \texttt{RP}_1, \texttt{SL}_1, \texttt{SL}_2, \texttt{SL}_3\}$ be a set of labels ranged over by $\alpha, \alpha_1, \alpha_2, \alpha', \ldots$ and let $\eta \in \{\alpha(x,y) \mid \alpha \in \mathfrak{L} \setminus \{\texttt{CD}\} \wedge x, y \in \mathcal{V}_\pi\} \cup \{\texttt{CD}(-)\}$. We define the labeled transition relation $\xrightarrow{\eta}_\ell$ by using the rules in Fig. 13, where we assume that $U = U_1 \parallel \cdots \parallel U_n$ with $n \geq 0$.

The following lemma ensures that labeled transitions and τ-transitions coincide.

Lemma 13 *Let S be a target term (cf. Def. 26) and x, y be endpoints. Then, $S \xrightarrow{\tau}_\ell S'$ if and only if $S \xrightarrow{\eta}_\ell S'$ where $\eta \in \{\alpha(x,y) \mid \alpha \in \{\texttt{IO}, \texttt{SL}, \texttt{RP}, \texttt{IO}_1, \texttt{RP}_1, \texttt{SL}_1, \texttt{SL}_2, \texttt{SL}_3\} \wedge x, y \in \mathcal{V}_\pi\} \cup \{\texttt{CD}(-)\}$.*

Proof The $\Rightarrow$ direction proceeds by a case analysis on the structure of target term S; the $\Leftarrow$ direction proceeds by a case analysis on the label η. For details see App. C.5. □

We further categorize labels in Def. 31 as *opening* or *closing*:

Definition 32 (Opening and Closing Labels) Consider the set of labels defined in Def. 31. We will say that $O = \{\texttt{IO}, \texttt{SL}, \texttt{RP}, \texttt{SL}_1\}$ is the set of *opening labels* and write ω to refer to its elements. Similarly, we will call $C = \{\texttt{IO}_1, \texttt{RP}_1, \texttt{CD}, \texttt{SL}_2, \texttt{SL}_3\}$ the set of *closing labels* and write κ to refer to its elements.

The idea is that a transition with an opening label always evolves into an intermediate process, whereas one with a closing label leads to the translation of a π process. Label $\texttt{CD}$ is closing because it does not have intermediate processes, but goes directly into the translation

of the continuation. Also, label $\mathtt{SL}_1$ is opening because it reaches an intermediate process, rather than the translation of a π process.

Now we introduce some notation for dealing with sequences of labels:

Notation 16

- *We write $\gamma(\widetilde{xy})$ to denote finite sequences $\alpha_1(x_1,y_1),\ldots,\alpha_m(x_n,y_n)$, with $n,m \geq 1$.*
- *We shall write $S \stackrel{\gamma(\widetilde{xy})}{\Longrightarrow}_{\ell} S'$ only if there exist target terms $S_1,\ldots,S_m$ such that $S \xrightarrow{\alpha_1(x_1,y_1)}_{\ell} S_1 \xrightarrow{\alpha_2(x_2,y_2)}_{\ell} \cdots S_{m-1} \xrightarrow{\alpha_m(x_n,y_n)}_{\ell} S_m = S'$ and $\gamma(\widetilde{xy}) = \alpha_1(x_1,y_2),\ldots,\alpha_m(x_n,y_n)$.*
- *Given $\gamma(\widetilde{xy})$, we write $\alpha(x,y) \in \gamma(\widetilde{xy})$ to denote that $\alpha(x,y)$ is in the sequence $\gamma(\widetilde{xy})$. Moreover, when x and y are unimportant, we write $\alpha \in \gamma(\widetilde{xy})$.*
- *Given $\gamma(\widetilde{xy})$, we write $\gamma(\widetilde{xy}) \setminus \alpha_i(x_j,y_j)$ to denote the sequence obtained from $\gamma(\widetilde{xy})$ by removing $\alpha_i(x_j,y_j)$.*
- *Given $\gamma(\widetilde{xy})$, we say that $\gamma(\widetilde{xy})$ is an opening (resp. closing) sequence if every $\alpha \in \gamma(\widetilde{xy})$ is an opening (resp. closing) label (cf. Def. 32).*

Opening and closing labels represent our two-step approach to simulate synchronizations in π: an opening label signals the beginning of a synchronization (i.e., consuming a constraint snd or sel), while a closing label signals its completion (i.e., consuming a constraint rcv or bra). Whenever a synchronization opens and closes, it can be shown that the translation reaches some π program. The following definition captures these *complete synchronizations*:

Definition 33 (Complete Synchronizations) Let S_0 be a target term (cf. Def. 26) such that $S_0 \stackrel{\gamma(\widetilde{xy})}{\Longrightarrow} S_1$.

1. If there exist $\gamma_1(\widetilde{xy})$ and $\gamma_2(\widetilde{xy})$ such that either:
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{IO}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{IO}_1(x,y)$ or
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{IO}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{RP}_1(x,y)$

   then we say that $\gamma(\widetilde{xy})$ is a *complete synchronization with respect to* $\mathtt{IO}(x,y)$.
2. If there exist $\gamma_1(\widetilde{xy})$ and $\gamma_2(\widetilde{xy})$ such that either:
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{RP}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{RP}_1(x,y)$
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{RP}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{IO}_1(x,y)$

   then we say that $\gamma(\widetilde{xy})$ is a *complete synchronization with respect to* $\mathtt{RP}(x,y)$.
3. If there exist $\gamma_1(\widetilde{xy})$, $\gamma_2(\widetilde{xy})$ and $\gamma_3(\widetilde{xy})$ such that either:
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{SL}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{SL}_1(x,y)\ \gamma_3(\widetilde{xy})\ \mathtt{SL}_2(x,y)$
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{SL}(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{SL}_1(x,y)\ \gamma_3(\widetilde{xy})\ \mathtt{SL}_3(x,y)$

   then we say that $\gamma(\widetilde{xy})$ is a *complete synchronization with respect to* $\mathtt{SL}(x,y)$.
4. If there exist $\gamma_1(\widetilde{xy})$ and $\gamma_2(\widetilde{xy})$ such that either:
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{SL}_1(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{SL}_2(x,y)$ or
   - $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{SL}_1(x,y)\ \gamma_2(\widetilde{xy})\ \mathtt{SL}_3(x,y)$

   then we say that $\gamma(\widetilde{xy})$ is a *complete synchronization with respect to* $\mathtt{SL}_1(x,y)$.
5. If there exists $\gamma_1(\widetilde{xy})$ such that $\gamma(\widetilde{xy}) = \gamma_1(\widetilde{xy})\ \mathtt{CD}(-)$ then we say that $\gamma(\widetilde{xy})$ is a *complete synchronization* with respect to $\mathtt{CD}(-)$.

Case 5 is the only whose translation needs a single `lcc` step to reach the translation of its continuation. Therefore, every conditional transition is a complete synchronization.

Example 6 (Complete Synchonizations) Consider the following target terms:

$$S_1 = C_{\widetilde{x}\widetilde{y}}[[\![x_1\langle v\rangle.P_1]\!] \parallel [\![y_1(z).y_2 \triangleright \{l_i : Q_i\}_{i\in I}]\!] \parallel [\![x_2 \triangleleft l.Q]\!]]$$
$$S_2 = C_{\widetilde{x}\widetilde{y}}[[\![x_1\langle v\rangle.P_1]\!] \parallel [\![y_1(z).P_2]\!] \parallel [\![y_1(z').P_3]\!]]$$
$$S_3 = C_{xy}[(\!|x \triangleleft l_j.P_1 \mid y \triangleright \{l_i : P_i\}_{i\in I}|\!)^1_{xy}]$$

The following transitions are complete synchronizations with respect to the first label in the sequence for processes S_1, S_2, and S_3:

$$\begin{aligned}
S_1 &\xrightarrow{\mathtt{IO}(xy)}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid y_1(z).x_2 \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{xy} \parallel [\![x_2 \triangleleft l.Q]\!]] \\
&\xrightarrow{\mathtt{IO}_1(xy)}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![x_2 \triangleright \{l_i : Q_i\}_{i\in I}]\!]\{v/z\} \parallel [\![x_2 \triangleleft l.Q]\!]] \\
S_2 &\xrightarrow{\mathtt{IO}(xy)}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid y_1(z).P_2|\!)^1_{xy} \parallel [\![y_1(z').P_3]\!]] \\
&\xrightarrow{\mathtt{IO}_1(xy)}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel [\![y_1(z').P_3]\!]] \\
S_3 &\xrightarrow{\mathtt{SL}_1(xy)}_\ell C_{xy}[(\!|x \triangleleft l_j.P_1 \mid y \triangleright \{l_i : P_i\}_{i\in I}|\!)^3_{xy}] \\
&\xrightarrow{\mathtt{SL}_3(xy)}_\ell C_{xy}[[\![P_1]\!] \parallel [\![Q_j]\!]]
\end{aligned}$$

Using complete synchronizations, we can describe the *open labels* of a sequence of transitions:

Definition 34 (Open Labels of a Sequence of Transitions) Let P be a well-typed π program such that $[\![P]\!] = S_0 \xRightarrow[\ell]{\gamma(\widetilde{x}\widetilde{y})} S_n$, with $n = |\gamma(\widetilde{x}\widetilde{y})|$. We define the *open labels* of $\gamma(\widetilde{x}\widetilde{y})$, written $open(\gamma(\widetilde{x}\widetilde{y}))$, as the longest sequence $\beta_1 \ldots \beta_m$ (with $m \leq n$) that preserves the order in $\gamma(\widetilde{x}\widetilde{y})$ and such that for every β_i (with $1 \leq i \leq m$):

(1) $\beta_i = \alpha_j$, for some opening label $\alpha_j \in \gamma(\widetilde{x}\widetilde{y})$;
(2) there is not a subsequence $\gamma(\widetilde{x}\widetilde{y})$ that is a complete synchronization with respect to β_i (cf. Def. 33).

The *complementary execution sequence* of an opening label intuitively contains the transition labels required to complete a synchronization:

Definition 35 (Complementary Execution Sequence) Let ω be any opening label. We say that the *complementary execution sequence* of ω, written $\omega\!\downarrow$, is defined as follows:

$$\begin{aligned}
\mathtt{IO}(xy)\!\downarrow &= \mathtt{IO}_1(xy) & \mathtt{RP}(xy)\!\downarrow &= \mathtt{RP}_1(xy) \\
\mathtt{SL}(xy)\!\downarrow &= \mathtt{SL}_1(xy)\mathtt{SL}_2(xy) & \mathtt{SL}_1(xy)\!\downarrow &= \mathtt{SL}_2(xy)
\end{aligned}$$

Furthermore, let $S_1 \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_2$ be transition sequence such that $open(\gamma(\widetilde{x}\widetilde{y})) = \omega_1 \ldots \omega_n$, with $n \geq 1$. We define $\gamma(\widetilde{x}\widetilde{y})\!\downarrow$ as $\omega_1\!\downarrow \ldots \omega_n\!\downarrow$.

The following lemma provides a diamond property for opening and closing transitions. It states that closing actions do not interfere with opening transitions. The proof follows by induction on the size of the sequence of labels (see App. C.5 for details).

Lemma 14 *Let S be a target term such that $S \xrightarrow{\omega}_\ell S_1$ and $S \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_2$, where $\gamma(\widetilde{x}\widetilde{y})$ is a closing sequence (cf. Not. 16). Then, there exists S_3 such that $S_1 \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_3$ and $S_2 \xrightarrow{\omega}_\ell S_3$.*

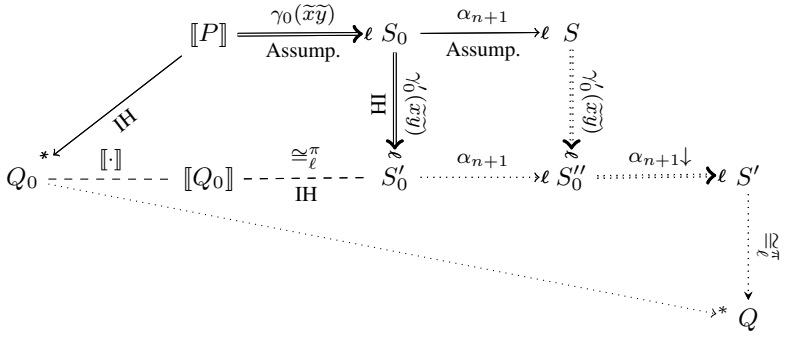


Fig. 14: Diagram of the proof of Lem. 15. The dotted arrows represent the reductions and equivalences that must be proven.

The next lemma shows that every target term can reach the translation of a π program by closing all its remaining open synchronizations, if any.

Lemma 15 *Suppose a well-typed π program P. For every sequence of labels $\gamma(\widetilde{xy})$ such that $[\![P]\!] \xRightarrow{\gamma(\widetilde{xy})} S$, there exist Q, S', and $\gamma'(\widetilde{xy})$ such that $P \longrightarrow^* Q$ and $S \xRightarrow{\gamma'(\widetilde{xy})} S'$, with $\gamma'(\widetilde{xy}) = \gamma(\widetilde{xy})\downarrow$ (cf. Def. 35). Moreover, $[\![Q]\!] \cong^\pi_\ell S'$.*

Proof By induction on $|\gamma(\widetilde{xy})|$ and a case analysis on the last label of the sequence. The base case is immediate since $[\![P]\!] \xRightarrow{\gamma(\widetilde{xy})} [\![P]\!]$ and $P \longrightarrow^* P$. The proof for the inductive hypothesis can be seen in Fig. 14. The dotted arrows are the reductions that must be proven to exist. For details see App. C.5. □

4.3.3 Proof of Operational Soundness

Having detailed all the ingredients required in our proof, we restate Thm. 12 (cf. Page 25) and develop the sketch discussed in § 4.3.1:

Theorem 12 (Soundness for $[\![\cdot]\!]$) *Let $[\![\cdot]\!]$ be the translation in Def. 25. Also, let P be a well-typed π program. For every S such that $[\![P]\!] \xRightarrow{\tau} S$ there are Q, S' such that $P \longrightarrow^* Q$ and $S \xRightarrow{\tau} S' \cong^\pi_\ell [\![Q]\!]$.*

Proof By induction on k, the length of the reduction $[\![P]\!] \xRightarrow{\tau} S$, followed by a case analysis on the constraints that may have been consumed in the very last reduction.

Base Case: Then $[\![P]\!] \xRightarrow{\tau} [\![P]\!]$. The thesis follows from reflexivity of $\cong^\pi_\ell$: $[\![P]\!] \cong^\pi_\ell [\![P]\!]$.

Inductive Step: Assume $[\![P]\!] \xRightarrow{\tau} S_0 \longrightarrow_\ell S$ (with $k-1$ steps between $[\![P]\!]$ and S_0). By IH, there exist Q_0 and S'_0 such that $P \longrightarrow^* Q_0$ and $S_0 \xRightarrow{\tau} S'_0 \cong^\pi_\ell [\![Q_0]\!]$. Observe that by combining the IH and Lem. 15, we have that the sequence $S_0 \xRightarrow{\tau} S'_0$ contains only closing labels. We must prove that there exist Q and S' such that $P \longrightarrow^* Q$ and

$S \stackrel{\tau}{\Longrightarrow} S' \cong_\ell^\pi [\![Q]\!]$. We analyze $\mathcal{I}_{S_0}$ and $\mathcal{I}_S$ (cf. Def. 29) according to two cases: $\mathcal{I}_{S_0} \subseteq \mathcal{I}_S$ and $\mathcal{I}_{S_0} \not\subseteq \mathcal{I}_S$, which use Lem. 11 and Lem. 12, respectively:

Case $\mathcal{I}_{S_0} \subseteq \mathcal{I}_S$: By Lem. 11 there are two sub-cases depending on the shape of S_0:

1. $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[[\![b?\, Q_1 : Q_2]\!] \parallel U]$, $b \in \{\mathtt{tt}, \mathtt{ff}\}$, for some U. By Lem. 11 and inspection on the translation definition (Fig. 8), it must be the case that $Q_0 \equiv_\pi (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(b?\, Q_1 : Q_2 \mid R)$, for some R. We distinguish two sub-subcases, depending on b:

   1.1 $b = \mathtt{tt}$**:** We proceed as follows:
   (1) $S_0 \stackrel{\tau}{\Longrightarrow} S_0' \equiv C_{\widetilde{x}\widetilde{y}}[[\![\mathtt{tt}?\, Q_1 : Q_2]\!] \parallel U']$, for some U' (IH).
   (2) $S_0' \cong_\ell^\pi [\![Q_0]\!] = C_{\widetilde{x}\widetilde{y}}[[\![\mathtt{tt}?\, Q_1 : Q_2]\!] \parallel [\![R]\!]]$ (IH).
   (3) $S_0 \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q_2]\!]) \parallel U] = S$ (Lem. 11).
   (4) $S_0 \longrightarrow_\ell S \stackrel{\tau}{\Longrightarrow} C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q_2]\!]) \parallel U'] = S'$ ((3),(1)).
   (5) $S_0' \longrightarrow_\ell \cong_\ell^\pi C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel U'] \cong_\ell^\pi S'$ (Fig. 6, Cor. 4, (4)).
   (6) $[\![Q_0]\!] \longrightarrow_\ell \cong_\ell^\pi C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel [\![R]\!]] = W$ ((2), Fig. 6, Cor. 4).
   (7) $S' \cong_\ell^\pi W = C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel [\![R]\!]]$ ((2),(5), Lem. 14 with $S = S_0$, $S_1 = S$, $S_2 = [\![Q_0]\!]$).
   (8) $Q_0 \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q_1 \mid R) = Q$ (Fig. 1 - Rule $\lfloor$IFT$\rfloor$)
   (9) $[\![Q]\!] = C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel [\![R]\!]] = W$ (Fig. 8, (8), (6)).
   (10) $S' \cong_\ell^\pi [\![Q]\!]$ ((7),(9)).

   1.2 $b = \mathtt{ff}$**:** This case is analogous to the one above.

2. By Lem. 11, there is a W such that $W \longrightarrow_\ell^h S_0$ (with $h \in \{1, 2\}$) where

$$W = C_{\widetilde{x}\widetilde{y}}[[\![x \triangleleft l_k.Q' \mid x \triangleright \{l_h : Q_h\}_{h\in I}]\!] \parallel U]$$

   for some U. We distinguish cases according to h:

   2.1 $h = 2$**:**
   (1) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_k.Q' \mid x \triangleright \{l_h : Q_h\}_{h\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel U]$ (Lem. 11).
   (2) $Q_0 = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q' \mid Q_k \mid R)$ ((1), IH).
   (3) $S_0 \stackrel{\tau}{\Longrightarrow} \cong_\ell^\pi C_{\widetilde{x}\widetilde{y}}[[\![Q' \mid Q_k]\!] \parallel U'] = S_0'$ (IH, Fig. 8, Cor. 4).
   (4) $S_0' \cong_\ell^\pi [\![Q_0]\!] = C_{\widetilde{x}\widetilde{y}}[[\![Q']\!] \parallel [\![Q_k]\!] \parallel [\![R]\!]]$ (IH, (3), (2)).
   (5) $S_0 \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q' \mid Q_k]\!] \parallel \prod_{h\in I\setminus\{k\}} \forall\epsilon(l_k = l_h \to [\![Q_h]\!]) \parallel U] = S$ (Fig. 6, (1)).
   (6) $S_0 \longrightarrow_\ell S \stackrel{\tau}{\Longrightarrow} \cong_\ell^\pi S_0' = S'$ ((5), (3), Lem. 14, with $S = S_0$, $S_1 = S$, $S_2 = [\![Q_0]\!]$).
   (7) $Q_0 \longrightarrow^* Q_0 = Q$ (Fig. 1).
   (8) $S' \cong_\ell^\pi [\![Q]\!]$ ((6),(7),(4)).

   2.2 $h = 1$**:**
   (1) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_k.Q' \mid x \triangleright \{l_h : Q_h\}_{h\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$ (Lem. 11).
   (2) $Q_0 = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q' \mid Q_k \mid R)$ ((1), Assumption)
   (3) $S_0 \stackrel{\tau}{\Longrightarrow} \cong_\ell^\pi C_{\widetilde{x}\widetilde{y}}[[\![Q' \mid Q_k]\!] \parallel U'] = S_0'$ (IH, Fig. 8, Cor. 4).
   (4) $S_0' \cong_\ell^\pi [\![Q_0]\!] = C_{\widetilde{x}\widetilde{y}}[[\![Q']\!] \parallel [\![Q_k]\!] \parallel [\![R]\!]]$ (IH,(3),(2)).
   (5) $S_0 \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_k.Q' \mid x \triangleright \{l_h : Q_h\}_{h\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel U] = S$ (Fig. 6).
   (6) $S \longrightarrow_\ell \cong_\ell^\pi C_{\widetilde{x}\widetilde{y}}[[\![Q' \mid Q_k]\!] \parallel U] \stackrel{\tau}{\Longrightarrow} C_{\widetilde{x}\widetilde{y}}[[\![Q' \mid Q_k]\!] \parallel U'] = S'$ (Fig. 6, Cor. 4, (3)).

(7) $S_0 \longrightarrow_\ell S \stackrel{\tau}{\Longrightarrow} \cong_\ell^\pi S_0' = S'$ ((5),(6), Lem. 14, (3), with $S = S_0$, $S_1 = S$, $S_2 = [\![Q_0]\!]$).
(8) $Q_0 \longrightarrow^* Q_0 = Q$ (Fig. 1).
(9) $S' \cong_\ell^\pi [\![Q]\!]$ ((7), (8), (4)).

Case $\mathcal{I}_{S_0} \not\subseteq \mathcal{I}_S$: By Lem. 12 we distinguish sub-cases depending on the constraints in $\mathcal{I}_{S_0} \setminus \mathcal{I}_S$. By Prop. 1, constraints are unique and therefore, $\mathcal{I}_{S_0} \setminus \mathcal{I}_S$ correctly accounts for the specific consumed constraint. There are four cases, as indicated by Lem. 12:

1. $\mathsf{snd}(x, v) \in \mathcal{I}_{S_0} \setminus \mathcal{I}_S$: By Lem. 12 we have, for some U:
(a) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![x\langle v\rangle.Q_1 \mid \diamond\ y(z).Q_2]\!] \parallel U]$.
(b) $S \equiv C_{\widetilde{x}\widetilde{y}}[(\!|x\langle v\rangle.Q_1 \mid \diamond\ y(z).Q_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$.
We distinguish cases depending on $\diamond\, y(z).Q_2$ (cf. Not. 2):

1.1 $\diamond\, y(z).Q_2 = y(z).Q_2$**:** We proceed as follows:
(1) $S_0 \stackrel{\tau}{\Longrightarrow} C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![x\langle v\rangle.Q_1 \mid y(z).Q_2]\!] \parallel U'] = S_0'$ (IH).
(2) $S_0' \cong_\ell^\pi [\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})(x\langle v\rangle.Q_1 \mid y(z).Q_2 \mid R)]\!] = [\![Q_0]\!]$ (IH).
(3) $S_0 \longrightarrow_\ell S \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q_1 \mid Q_2\{v/z\}]\!] \parallel U]$ ((a),(b), Fig. 6).
(4) $S \stackrel{\tau}{\Longrightarrow} C_{\widetilde{x}\widetilde{y}}[[\![Q_1 \mid Q_2\{v/z\}]\!] \parallel U'] = S'$ ((3),(1)).
(5) $S_0' \longrightarrow_\ell^2 C_{\widetilde{x}\widetilde{y}}[[\![Q_1 \mid Q_2\{v/z\}]\!] \parallel U'] = S'$ ((1), Fig. 6, (4)).
(6) $[\![Q_0]\!] \longrightarrow_\ell^2 [\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q_1 \mid Q_2\{v/z\} \mid R)]\!] = W$ ((2), Fig. 6).
(7) $S' \cong_\ell^\pi W$ ((2), (5), Lem. 14 with $S = S_0, S_1 = S, S_2 = [\![Q_0]\!]$).
(8) $Q_0 \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q_1 \mid Q_2\{v/z\} \mid R) = Q$ (Fig. 1 - Rule $\lfloor$Com$\rfloor$).
(9) $W = [\![Q]\!]$ ((8), (6)).
(10) $S' \cong_\ell^\pi [\![Q]\!]$ ((7),(9)).

1.2 $\diamond\, y(z).Q_2 = *\, y(z).Q_2$**:** Similar to the case above, using Rule $\lfloor$Repl$\rfloor$ instead of Rule $\lfloor$Com$\rfloor$.

2. $\mathsf{rcv}(x, v) \in \mathcal{I}_{S_0} \setminus \mathcal{I}_S$: By Lem. 12, there exists

$$W = C_{\widetilde{x}\widetilde{y}}[[\![x\langle v\rangle.Q_1 \mid \diamond\ y(z).Q_2]\!] \parallel U]$$

such that $W \longrightarrow_\ell S_0$. We distinguish two cases for $\diamond\, y(z).Q_2$ (cf. Not. 2):

2.1 $\diamond\, y(z).Q_2 = y(z).Q_2$**:** We proceed as follows:
(1) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[(\!|x\langle v\rangle.Q_1 \mid \diamond\ y(z).Q_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$ (Lem. 12).
(2) $Q_0 = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(Q_1 \mid Q_2\{v/z\} \mid R)$ ((1)).
(3) $S_0 \stackrel{\tau}{\Longrightarrow} C_{\widetilde{x}\widetilde{y}}[[\![Q_1 \mid Q_2\{v/z\} \mid R]\!] \parallel U'] = S_0'$ ((1), Fig. 6, IH).
(4) $S_0' \cong_\ell^\pi [\![Q_0]\!]$ (IH).
(5) $S_0 \longrightarrow_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q_1 \mid Q_2\{v/z\} \mid R]\!] \parallel U] = S$ (Fig. 6).
(6) $S_0 \longrightarrow_\ell S \stackrel{\tau}{\Longrightarrow} S_0' = S'$ ((5), (3), Lem. 14, with $S = S_0$, $S_1 = S$, $S_2 = [\![Q_0]\!]$).
(7) $Q_0 \longrightarrow^* Q_0 = Q$ (Fig. 1).
(8) $S' \cong_\ell^\pi [\![Q]\!]$ ((6), (7), (4)).

2.2 $\diamond\, y(z).Q_2 = *\, y(z).Q_2$**:** Similar as above.

3. $\mathsf{sel}(x, l) \in \mathcal{I}_{S_0} \setminus \mathcal{I}_S$: As above.
4. $\mathsf{bra}(x, l) \in \mathcal{I}_{S_0} \setminus \mathcal{I}_S$: As above. □

Besides the criteria considered in our definition of valid encoding (Def. 20), Gorla [26] advocates for *divergence reflection*, a correctness criterion that we informally discuss here,

as it is related to operational correspondence. Divergence reflection ensures that every infinite sequence of reductions in a target term corresponds to some infinite sequence of reductions in its associated source term. Let us write $S \longrightarrow_s^\omega$ (resp. $T \longrightarrow_t^\omega$) whenever the source term S (resp. target term T) has such an infinite sequence of reductions. A translation $[\![\cdot]\!] : \mathcal{L}_s \to \mathcal{L}_t$ then reflects divergence if for every S such that $[\![S]\!] \longrightarrow_t^\omega$ then $S \longrightarrow_s^\omega$.

Our translation $[\![\cdot]\!]$ (Fig. 8) reflects divergence. The only sources of infinite behavior it induces concern the translation of restriction, which includes the persistent tell process $!\overline{\{x{:}y\}}$, and the translation of input-guarded replication in π. The persistent tell cannot reduce by itself; by providing copies of constraint $\{x{:}y\}$, it partakes in target reductions (inferred using Rule $\lfloor$C:SYNC$\rfloor$). Importantly, such reductions occur only when an auxiliary predicate (such as $\mathsf{rcv}(x, y)$, added by the translation of a source process) is also in the store. The translation of input-guarded replication does not reduce on its own either: associated reductions depend on synchronizations with (the translation of) corresponding outputs. Therefore, as $[\![\cdot]\!]$ does not induce other forms of infinite behavior, every infinite sequence of reductions emanating from $[\![P]\!]$ corresponds exclusively to infinite reductions present in P.

4.4 Success Sensitiveness

Here we consider success sensitiveness, the last criterion in our definition of valid encoding (Def. 20). For the purpose of proving that our translation satisfies this criterion, we adopt some extensions, following [25,26]. First, we extend the syntax of π (cf. Def. 1) and `lcc` (Def. 11) with a *success process*, denoted ✓ in both languages. In π, the operational semantics is kept unchanged, assuming that ✓ is preserved by reduction. That is, if $P \mid \checkmark$ and $P \longrightarrow^* Q$ then $Q = Q' \mid \checkmark$, for some Q'. The new process ✓ is assumed to be well-typed. In `lcc`, we define $\checkmark \stackrel{\text{def}}{=} !\overline{check}$, where $check$ denotes a constraint that is *fresh*, i.e., it does not occur anywhere else. Thus, in `lcc`, the success process ✓ defines $check$ as a persistent constraint; in particular, notice that $[\![\cdot]\!]$ does not introduce abstractions that consume $check$.

With process ✓ in place, we define success predicates for π and `lcc` as the potential that a process has of reducing to a process with a top-level unguarded occurrence of ✓:

$$P \Downarrow_\pi \text{ iff } P \longrightarrow^* P' \text{ and } P' \equiv_\pi P'' \mid \checkmark$$
$$P \Downarrow_{\texttt{lcc}} \text{ iff } P \stackrel{\tau}{\Longrightarrow} P' \text{ and } P' \equiv P'' \parallel \checkmark = P'' \parallel !\overline{check}$$

The equivalences in each language (cf. Def. 24) are *sensitive to success*. That is, it is never the case that $P \equiv_\pi Q$ if $P \Downarrow_\pi$ but $Q \not\Downarrow_\pi$ (cf. Def. 2). Similarly, by adding the (fresh) constraint $check$ to the set of output observables used as parameter to $\cong_\ell^\pi$ (Def. 22), it is never the case that $P \cong_\ell^\pi Q$ if $P \Downarrow_{\texttt{lcc}}$ but $Q \not\Downarrow_{\texttt{lcc}}$ (cf. Def. 23).

Under these assumptions, success sensitiveness is a property of the translation $[\![\cdot]\!]$ in Fig. 8, trivially extended with the case $[\![\checkmark]\!] = !\overline{check}$. For this (extended) translation, the proof of success sensitiveness, given next, relies on operational completeness and soundness (Thm. 11 and Thm. 12), which we assume extended to the source and target languages with success processes.

Theorem 17 (Success Sensitiveness for $[\![\cdot]\!]$) *Let $[\![\cdot]\!]$ be the translation in Def. 25. Also, let P be a well-typed π program. Then, $P \Downarrow_\pi$ if and only if $[\![P]\!] \Downarrow_{\texttt{lcc}}$.*

Proof We consider both directions. Recall that Fig. 8 defines $[\![P \mid Q]\!] = [\![P]\!] \parallel [\![Q]\!]$.

1. If $P \Downarrow_\pi$ then $P \longrightarrow^* P'$ and $P' \equiv_\pi P'' \mid \checkmark$. By operational completeness (Thm. 11), there exists an S such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \cong_\ell^\pi [\![P'' \mid \checkmark]\!] = [\![P'']\!] \parallel !\overline{check}$. Because $\cong_\ell^\pi$ is sensitive to success, we have $S \Downarrow_{\texttt{lcc}}$. Therefore, $[\![P]\!] \Downarrow_{\texttt{lcc}}$.
2. If $[\![P]\!] \Downarrow_{\texttt{lcc}}$ then $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S$ and $S \equiv S_1 \parallel !\overline{check}$. By operational soundness (Thm. 12), there exist Q, S' such that $P \longrightarrow^* Q$ and $S_1 \parallel !\overline{check} \stackrel{\tau}{\Longrightarrow} S' \cong_\ell^\pi [\![Q]\!]$. Now, because process $!\overline{check}$ is persistent, we have that $S' = S'' \parallel !\overline{check}$. Because success is captured by $\cong_\ell^\pi$, we infer that $[\![Q]\!] = [\![Q']\!] \parallel !\overline{check} = [\![Q']\!] \mid \checkmark$, for some Q'. Therefore, $P \longrightarrow^* Q' \mid \checkmark$ and we conclude that $P \Downarrow_\pi$.

□

We have proven that the translation $[\![\cdot]\!]$ is name invariant (Thm. 8), compositional (Thm. 9), operationally complete (Thm. 11), operationally sound (Thm. 12), and also success sensitive (Thm. 17). Therefore, we may now state that $[\![\cdot]\!]$ is a valid encoding:

Corollary 5 *The translation $\langle [\![\cdot]\!], \varphi_{[\![\cdot]\!]} \rangle$ (cf. Def. 25) is a valid encoding (cf. Def. 20).*

As an application of our encoding and its correctness properties, in the following section we devise a methodology to use encoded terms as macros in `lcc` contexts; we show how it can be used to enhance π specifications with time constraints.

5 The Encoding at Work

The correctness properties of $[\![\cdot]\!]$ (in particular, compositionality and operational correspondence) can be used to enhance operational specifications written in π with declarative features that can be expressed in `lcc`. We focus on expressing two of the patterns presented by Neykova et al. in [36], where a series of realistic communication protocols with time were analyzed using the Scribble specification language [46].

Next, we overview our approach. Then, in § 5.2 and § 5.3 we develop two of the patterns presented in [36] and show how our encoding can help in specifying them in `lcc`.

5.1 Overview: Exploiting Compositionality via Decompositions

As shown before, $[\![\cdot]\!]$ satisfies correctness properties that ensure that source specifications in π can be represented in `lcc` and that their behavior is preserved (cf. Cor. 5). We are interested in using $[\![\cdot]\!]$ to represent requirements that may appear in message-passing components but are not explicitly representable in π. An example of such requirements is: "*an acknowledgment message* `ACK` *should be sent (by the server) no later than three time units after receiving the request message* `REQ`". As already mentioned, this kind of behavior is not straightforwardly representable in π. In fact, using π we could only represent the interaction in the previous requirement as a *request-acknowledgment handshake*. Consider the process

$$P_h = (\boldsymbol{\nu} xy)(x\langle \texttt{REQ} \rangle.x(z).\mathbf{0} \mid y(z').y\langle \texttt{ACK} \rangle.\mathbf{0}) \tag{6}$$

where x represents the endpoint used by the client, while y represents the endpoint of the server. The client sends `REQ` and then awaits for `ACK`, which is sent by the server.

To represent this kind of requirements, the key idea is to consider encoded terms as macros to be used inside a larger `lcc` specification. Then, because of the correctness properties we have established for $[\![\cdot]\!]$, these snippets represent `lcc` processes that will execute correct communication behavior. For example, using $[\![P_h]\!]$ we could specify the `lcc` process

$$Q = \forall\epsilon\big(\mathit{month}(m) = \mathit{january} \rightarrow [\![P_h]\!]\big)$$

which checks that the current month (stored in variable m) is $\mathit{january}$; only when this constraint is true, the behavior in $[\![P_h]\!]$ is executed. Our encoding's operational correspondence property ensures that the behavior of Q corresponds to that of P_h, provided that the guard is satisfied (cf. Thm. 11 and Thm. 12). This is useful to represent communication behavior that depends on information that is *contextual*, i.e., external to the program.

In a way, Q defines a constraint over the whole process P_h. It can be desirable to constrain only some of the actions of P_h. For the sake of illustration, let us imagine a variant of π in which prefixes are annotated:

$$P'_h = (\boldsymbol{\nu} xy)(x\langle \mathtt{REQ}\rangle.x^3(z).\mathbf{0} \mid y(z').y\langle \mathtt{ACK}\rangle.\mathbf{0}) \tag{7}$$

Above, prefix $x^3(z).\mathbf{0}$ says that the input action should take place within three time units after the preceding action. This defines a *timed requirement* connecting two separate actions.

To specify processes such as P'_h using our encoding, we consider π processes that have been *decomposed* in such a way that every prefix appears in an independent parallel process. Such a decomposition should preserve the order in which actions are executed (as dictated by session types). In this way, the compositionality property of our encoding gives us control over specific parts of the translated π process, allowing us to define constraints on the translations of some specific prefixes (cf. Thm. 10).

Decompositions of (un)typed π-calculus processes have been studied as *trios processes* and *minimal session types* [38, 1]. The idea in [38] is to transform a process P into an equivalent process formed only by *trios*, sequential processes with at most three prefixes. Each trio emulates a specific prefix of P; trios interact in such a way that every sub-term is executed at the right time, to ensure that the *causal order* of interactions in P is preserved. Building upon this idea, the work of Arslanagic et al. on minimal session types [1] considers decompositions of processes and of their associated session types.

We shall build upon the decomposition strategy in [38, 1] and leave an in-depth study of decompositions for π for follow up work. We generate π processes in which each prefix is represented by a parallel subprocess, while preserving the causal order of the given process. Then, using the compositionality of $[\![\cdot]\!]$ (Thm. 9), we obtain an appropriate granularity level for the analysis of code snippets (obtained from trios) in `lcc` specifications.

For the sake of illustration, we follow [1] and consider the decomposition of well-typed programs from the finite fragment of π without selection and branching (i.e., output, input, restriction, parallel composition, and inaction). Since decompositions based on trios processes require polyadic communication, we shall assume the following shorthand notations:

$$\begin{aligned} x\langle\widetilde{v}\rangle.P &= x\langle v_1\rangle.\ldots.x\langle v_n\rangle.P \quad (|\widetilde{v}| = n \wedge n \geq 1)\\ x(\widetilde{y}).P &= x(y_1).\ldots.x(y_n).P \quad (|\widetilde{y}| = n \wedge n \geq 1) \end{aligned}$$

We use $\mathit{size}(\cdot)$ and $\mathsf{size}(\cdot)$ to denote the size of process P and type T, respectively (cf. Fig. 15). With these auxiliary functions, we define the following decomposition function:

$$size(P) = \begin{cases} 1 + size(Q) & \text{if } P = x\langle v\rangle.Q \text{ or } P = x(y).Q \\ 1 + size(Q_1) + size(Q_2) & \text{if } P = Q_1 \mid Q_2 \\ size(Q) & \text{if } P = (\boldsymbol{\nu} xy)Q \\ 1 & \text{if } P = \mathbf{0} \end{cases}$$

$$\mathsf{size}(T) = \begin{cases} 1 + \mathsf{size}(T') & \text{if } T = q!T.T' \text{ or } T = q?T.T' \\ 1 & \text{if } T = \texttt{bool} \\ 0 & \text{if } T = \texttt{end} \end{cases}$$

Fig. 15: Size of a process (resp. type) in the finite fragment of π.

P	$\mathfrak{B}^k_{\widetilde{u}}(P)$	Side Conditions
$x_{i,j}\langle v\rangle.Q$	$c_k(\widetilde{u_1}x_{i,j}\widetilde{u_2}).x_{i,j}\langle\widetilde{v}\rangle.d_{k+1}\langle\widetilde{u_1}\widetilde{u_2}\rangle.\mathbf{0} \mid \mathfrak{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}\setminus\widetilde{v}}(Q\{^{x_{i,j+1}}/_{x_{i,j}}\})$	$\widetilde{u} = \widetilde{u_1}x_{i,j}\widetilde{u_2}$ $i \in \{1,\dots,n\}$ $j \in \{1,\dots,m\}$
$x_{i,j}(y).Q$	$c_k(\widetilde{u_1}x_{i,j}\widetilde{u_2}).x_{i,j}(y\widetilde{z}).d_{k+1}\langle\widetilde{u_1}\widetilde{u_2}y\widetilde{z}\rangle.\mathbf{0} \mid \mathfrak{B}^{k+1}_{\widetilde{u_1}\widetilde{u_2}y\widetilde{z}}(Q\{^{x_{i,j+1}}/_{x_{i,j}}\})$	$\widetilde{u} = \widetilde{u_1}x_{i,j}\widetilde{u_2}$ $i \in \{1,\dots,n\}$ $j \in \{1,\dots,m\}$
$P \mid Q$	$c_k(\widetilde{u_1}\widetilde{u_2}).d_{k+1}\langle\widetilde{u_1}\rangle.d_{l+1}\langle\widetilde{u_2}\rangle.\mathbf{0} \mid \mathfrak{B}^{k+1}_{\widetilde{u_1}}(Q_1) \mid \mathfrak{B}^{l+1}_{\widetilde{u_2}}(Q_2)$	$l = k + size(Q_1)$ $\mathsf{fv}_\pi(Q_1) \in \widetilde{u_1}$ $\mathsf{fv}_\pi(Q_2) \in \widetilde{u_2}$
$\mathbf{0}$	$c_k(\widetilde{u}).\mathbf{0}$	

Fig. 16: Breakdown function for the processes in the finite fragment of π.

Definition 36 (Decomposition) Let $P = (\boldsymbol{\nu} x_1y_1)\dots(\boldsymbol{\nu} x_ny_n)Q$ be a well-typed program in the finite fragment of π and $\Gamma = \{x_1 : T_1, y_1 : \overline{T_1}, \dots, x_n : T_n, y_n : \overline{T_n}\}$ be a context such that $\Gamma \vdash Q$. The decomposition of P, denoted $\mathfrak{D}(P)$, is defined as

$$\mathfrak{D}(P) = (\boldsymbol{\nu}\widetilde{c}\widetilde{d})(\boldsymbol{\nu}\widetilde{u})(d_1\langle\widetilde{u}\rangle.\mathbf{0} \mid \mathfrak{B}^1_{\widetilde{u}}(Q\{^{\widetilde{w'}}/_{\widetilde{w}}\}))$$

where:

(1) $\widetilde{u} = \widetilde{x_1}\widetilde{y_1}\dots\widetilde{x_n}\widetilde{y_n}$, $\widetilde{w} = x_1y_1\dots x_ny_n$, and $\widetilde{w'} = x_{1,1}y_{1,1}\dots x_{n,1}y_{n,1}$.
(2) $\widetilde{x_i} = x_{i,1}\dots x_{i,m}$ and $\widetilde{y_i} = y_{i,1}\dots y_{i,m}$ with $m = \mathsf{size}(T_i)$ for every $i \in \{1,\dots,m\}$.
(3) $\widetilde{c} = c_1\dots c_r$ and $\widetilde{d} = d_1\dots d_r$ with $r = size(P)$.
(4) $\mathfrak{B}^k_{\widetilde{u}}(P)$ is defined inductively over the finite fragment of π as in Fig. 16.

Remark 3 We decompose only well-typed programs (cf. Not. 3). Thm. 4 ensures that for every program P to be decomposed, it holds that $P = (\boldsymbol{\nu} x_1y_1)\dots(\boldsymbol{\nu} x_ny_n)Q$, with $n \geq 0$. (For simplicity, we shall also assume no $(\boldsymbol{\nu} x'y')$ occurs in Q.) Moreover, typability ensures there exists a context $\Gamma = \{x_1 : T_1, y_1 : \overline{T_1}, \dots, x_n : T_n, y_n : \overline{T_n}\}$ such that $\Gamma \vdash Q$.

Fig. 17 illustrates $\mathfrak{D}(P_h)$, the process decomposition of P_h obtained from Def. 36. We shall use this decomposition to represent the timed behavior required by P'_h. Let us first analyze how parallel sub-processes in $\mathfrak{D}(P_h)$ implement individual prefixes of P_h:

– process $c_2(\widetilde{u_x}).x_{1,1}\langle\texttt{REQ}\rangle.d_3\langle x_{1,2}\rangle.\mathbf{0}$ implements prefix $x\langle\texttt{REQ}\rangle$.

$$\begin{aligned}\mathfrak{D}(P_h) = {} & (\boldsymbol{\nu} c_1 d_1)(\boldsymbol{\nu} c_2 d_2)(\boldsymbol{\nu} c_3 d_3)(\boldsymbol{\nu} c_4 d_4)(\boldsymbol{\nu} c_5 d_5)(\boldsymbol{\nu} c_6 d_6)(\boldsymbol{\nu} c_7 d_7) \\ & (\boldsymbol{\nu} x_{1,1} y_{1,1})(\boldsymbol{\nu} x_{1,2} y_{1,2}) \\ & (d_1\langle \widetilde{u} \rangle.\mathbf{0} \mid c_1(\widetilde{u}).d_2\langle \widetilde{u_x} \rangle.d_5\langle \widetilde{u_y} \rangle.\mathbf{0} \mid c_2(\widetilde{u_x}).x_{1,1}\langle \mathtt{REQ} \rangle.d_3\langle x_{1,2} \rangle.\mathbf{0} \mid \\ & c_3(x_{1,2}).x_{1,2}(z).d_4\langle z \rangle.\mathbf{0} \mid c_4(w).\mathbf{0} \mid c_5(\widetilde{u_y}).y_{1,1}(z').d_6\langle z' y_{1,2} \rangle.\mathbf{0} \mid \\ & c_6(z' y_{1,2}).y_{1,2}\langle \mathtt{ACK} \rangle.d_7\langle z' \rangle.\mathbf{0} \mid c_7(w).\mathbf{0})\end{aligned}$$

Fig. 17: A process decomposition. We let $\widetilde{u_x} = x_{1,1}x_{1,2}$, $\widetilde{u_y} = y_{1,1}y_{1,2}$, and $\widetilde{u} = \widetilde{u_x}, \widetilde{u_y}$.

- process $c_3(x_{1,2}).x_{1,2}(z).d_4\langle z \rangle.\mathbf{0}$ implements prefix $x(z)$.
- process $c_4(w).\mathbf{0}$ implements $\mathbf{0}$.

The sub-processes of $\mathfrak{D}(P_h)$ that do not correspond to a prefix in P_h are auxiliary processes that trigger the prefix representations. Process $\mathfrak{D}(P_h)$ is typable under the appropriate contexts. This is because, by definition, there are no shared variables and each pair of co-variables implement complementary behaviors. Also, the decomposition can be shown to preserve the causal order of the source process [38,1]. In our example, the order implemented by P_h to send messages `REQ` and `ACK` is preserved by $\mathfrak{D}(P_h)$—see App. A for the associated reduction steps. We may then argue that P_h and $\mathfrak{D}(P_h)$ execute the same protocol, up to the synchronizations added by the decomposition.

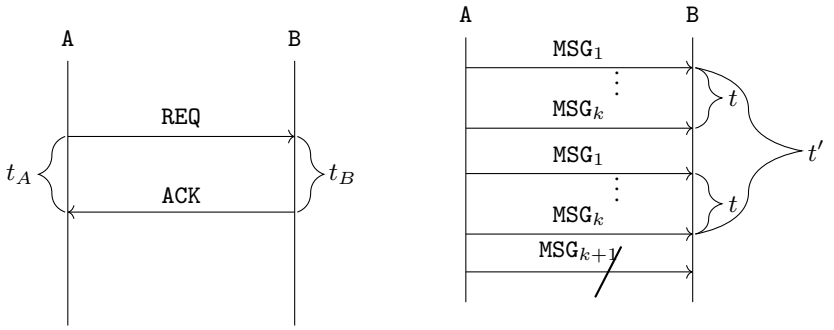
By using $\mathfrak{D}(P_h)$ instead of P_h as the source process for $[\![\cdot]\!]$, we can modularly treat translations for each prefix of P_h (i.e., its trios) as "black boxes" in `lcc`. Our operational correspondence results (Thm. 11 and Thm. 12) ensure that these black boxes correctly mimick the behavior of $\mathfrak{D}(P_h)$. This allows us to safely "plug" these black boxes into a larger `lcc` specification with additional declarative requirements, thus going beyond the merely operational specification given by P_h. For example, we can define an `lcc` process that specifies the timed conditions in P'_h in (7), which cannot be implemented easily in π.

We now use these ideas to develop `lcc` specifications of two of the communication patterns presented in [36]—they are graphically represented in Fig. 18a and Fig. 18b. These patterns will allow us to use `lcc` to specify processes such as P'_h.

5.2 Request-Response Timeout

This pattern is used to enforce requirements on the timing of a response, ensuring quality of service; it can be required both at server or client side (cf. Fig. 18a). Process P'_h can be seen as a specific implementation of this pattern. In [36] three use cases are identified:

(1) In [19], a service is requested to respond timely: "*an acknowledgment message* `ACK` *should be sent (by the server) no later than one second after receiving the request message* `REQ`".
(2) Similarly, also in [19], a Travel Agency web service specifies the pattern at the client side: "*A user should be given a response* `RES` *within one minute of a given request* `REQ`".
(3) Finally, extracted from the Simple Mail Transport Protocol specification [30], we have a requirement that exhibits a composition of request-response timeout patterns: "*a user should have a five minutes timeout for the* `MAIL` *command and a three minutes timeout for the* `DATABLOCK` *command*".



(a) Request-response timeout. (b) Messages in a time-frame.

Fig. 18: Timed patterns for communicating systems.

Requirement (1) above (i.e., the pattern at the server side) specifies that a reply should be sent within a fixed amount of time after the request have been received. In Requirement (2), which represents the client side, the server must be ready to receive the client's response within a fixed amount of time. In general, these patterns can be written as:

(a) *Server side:* After receiving a message `REQ` from `A`, `B` must send the acknowledgment `ACK` within $t_{\mathtt{A}}$ time units.
(b) *Client side:* After sending a message `REQ` to `B`, `A` must be able to receive the acknowledgment `ACK` from `B` within $t_{\mathtt{B}}$ time units.

A possible π specification for this pattern is shown below. We use a more general version of P'_h in (7), which is informally annotated to describe the intended timing requirements:

$$P_r = (\boldsymbol{\nu} xy)(\underbrace{x\langle \mathtt{REQ}\rangle.x(z).Q_1 \mid y(z).y\langle \mathtt{ACK}\rangle}_{t}.Q_2) \tag{8}$$

The intent is that the time elapsed between the reception of the request and the acknowledgment must not exceed t time units.

We now present a decomposition for P_r, following Def. 36:

$$\mathfrak{D}(P_r) = (\boldsymbol{\nu}\widetilde{u})(\mathfrak{D}_1 \mid \mathfrak{D}_2 \mid \mathfrak{D}_4 \mid \mathfrak{D}_3 \mid R) \tag{9}$$

where we assume that R contains the decompositions for processes Q_1 and Q_2 (not needed in this example), that $\widetilde{u}$ can be obtained from Def. 36, and that each of the parallel subprocesses $\mathfrak{D}_i$ ($i \in \{1,2,3,4\}$) represents a prefix in P_r:

$$\begin{aligned}
\mathfrak{D}_1 &= c_2(\widetilde{u_x}).x_{1,1}\langle \mathtt{REQ}\rangle.d_3\langle \widetilde{u_x} \setminus x_{1,1}\rangle.\mathbf{0}\\
\mathfrak{D}_2 &= c_3(\widetilde{u_x} \setminus x_{1,1}).x_{1,2}(z\widetilde{z_1}).d_4\langle \widetilde{u_x} z\widetilde{z_1} \setminus x_{1,1}x_{1,2}\rangle.\mathbf{0}\\
\mathfrak{D}_3 &= c_5(\widetilde{u_y}).y_{1,1}(z'\widetilde{z_2}).d_6\langle \widetilde{u_y} z'\widetilde{z_2} \setminus y_{1,1}\rangle.\mathbf{0}\\
\mathfrak{D}_4 &= c_6(\widetilde{u_y} z'\widetilde{z_2} \setminus y_{1,1}).y_{1,2}\langle \mathtt{ACK}\rangle.d_7\langle \widetilde{u_y} z'\widetilde{z_2} \setminus y_{1,1}y_{1,2}\rangle.\mathbf{0}
\end{aligned}$$

By applying $[\![\cdot]\!]$ and thanks to Thm. 9, we can use the translations of each $\mathfrak{D}_i$ ($i \in \{1,2,3,4\}$) as part of an `lcc` process that represents the request-response timeout pattern:

$$S_1 = C_{\widetilde{u}}[[\![\mathfrak{D}_1]\!] \parallel \forall\epsilon(clock(x) \leq t \rightarrow [\![\mathfrak{D}_2]\!]) \parallel [\![\mathfrak{D}_3]\!] \parallel [\![\mathfrak{D}_4]\!] \parallel [\![R]\!]]$$

where $\widetilde{u}$ can be obtained by following Def. 36. We can also exploit non-deterministic choices in `lcc` to signal that the process fails after a time-out:

$$S_2 = \exists \widetilde{x}\widetilde{y}.\big([\![\mathfrak{D}_1]\!] \parallel (\forall\epsilon(clock(u) \leq t \rightarrow [\![\mathfrak{D}_2]\!]) + \forall\epsilon(clock(u) > t \rightarrow Q_f)) \parallel [\![\mathfrak{D}_3]\!] \parallel [\![\mathfrak{D}_4]\!] \parallel [\![R]\!]\big)$$

Whenever $clock(u) > t$, process S_2 reduces to process Q_f which represents the actions that must be taken in case the timing constraint is not met. Interestingly, if Q_f is taken to represent an error process, then the behavior would be reminiscent of the input operators with deadlines presented in [9], which evolve into a failure whenever a deadline is not met.

The operational correspondence property of $[\![\cdot]\!]$ (cf. Thm. 11 and Thm. 12) ensures that S_2 preserves the behavior of the source π processes, whenever the timing constraint is met.

5.3 Messages in a Time-Frame

This pattern enforces a limit on the number of messages exchanged within a given time-frame (cf. Fig. 18b). In [36] two use cases from [19] were identified:

(1) Controlling *denial of service* attacks: "*a user is allowed to send only three redirect messages to a server with an interval between the messages of no less than two time units*".
(2) Used in a Travel Agency web service: "*a customer can change the date of his travel only two times and this must happen between one and five days of the initial reservation*".

This pattern concerns message repetition, which can be specified in two ways. We can (a) require the repetition to occur at a specified pace; i.e., requiring that messages can only be sent in intervals of time, or (b) specify an overall time-frame for the messages to be sent.

We generalize these patterns next. Keeping consistency with Fig. 18b, we will use t for the interval pattern and t', for the overall time-frame pattern. We use r and r' to denote the upper bound of the time-frames, i.e., $t \leq i \leq r$ (resp. $t' \leq i \leq r'$), where i stands for the "safe time" for sending messages.

(a) *Interval:* `A` is allowed to send `B` at most k messages, and at time intervals of at least t and at most r time units.
(b) *Overall time-frame:* `A` is allowed to send `B` at most k messages in the overall time-frame of at least t' and at most r' time units.

To represent these two variants of the pattern using $[\![\cdot]\!]$, we first present two π processes, annotated to indicate the timing constraints:

$$P_{ti} = (\boldsymbol{\nu} xy)(x\langle \underbrace{\mathtt{M}_1}_{t_1}\rangle.x\langle \underbrace{\mathtt{M}_2}_{t_1}\rangle.x\langle \underbrace{\mathtt{M}_3}_{t_1}\rangle.x\langle \mathtt{M}_4\rangle.P_x \mid P_y)$$

$$P_{to} = (\boldsymbol{\nu} xy)(\underbrace{x\langle \mathtt{M}_1\rangle.x\langle \mathtt{M}_2\rangle.x\langle \mathtt{M}_3\rangle.x\langle \mathtt{M}_4\rangle}_{t_2}.P_x \mid P_y)$$

Processes P_{ti} and P_{to} differ only on their timing constraints (i.e., the annotations below). They both send four messages that must be received by some P_y (which we leave unspecified). Process P_{ti} represents the interval pattern, in which we must leave *at least* t_1 time units between each message. Process P_{to} represents the overall time-frame pattern in which all the four messages must be sent in an overall time of t_2 time units. Here we use $clock(u_1)$ and $clock(u_2)$ to represent the time elapsed in clocks u_1 and u_2, respectively.

We start by considering the decompositions of $\mathfrak{D}(P_{ti})$ and $\mathfrak{D}(P_{to})$:

$$\mathfrak{D}(P_{to}) = \mathfrak{D}(P_{ti}) = (\boldsymbol{\nu}\widetilde{u})(\mathfrak{D}_1 \mid \mathfrak{D}_2 \mid \mathfrak{D}_3 \mid \mathfrak{D}_4 \mid R_x \mid R_y) \tag{10}$$

where we assume that R_x and R_y contain the decompositions of P_x and P_y, respectively. The sequence $\widetilde{u}$ can be derived from Def. 36. Processes $\mathfrak{D}_i$ ($i \in \{1,2,3,4\}$) correspond to:

$$\begin{aligned}
\mathfrak{D}_1 &= c_2(\widetilde{u_x}).x_{1,1}\langle \mathtt{M}_1\rangle.d_3\langle \widetilde{u_x} \setminus x_{1,1}\rangle.\mathbf{0}\\
\mathfrak{D}_2 &= c_3(\widetilde{u_x} \setminus x_{1,1}).x_{1,2}\langle \mathtt{M}_2\rangle.d_4\langle \widetilde{u_x} \setminus x_{1,1}x_{1,2}\rangle.\mathbf{0}\\
\mathfrak{D}_3 &= c_4(\widetilde{u_x} \setminus x_{1,1}x_{1,2}).x_{1,3}\langle \mathtt{M}_3\rangle.d_5\langle \widetilde{u_x} \setminus x_{1,1}x_{1,2}x_{1,3}\rangle.\mathbf{0}\\
\mathfrak{D}_4 &= c_5(\widetilde{u_x} \setminus x_{1,1}x_{1,2}x_{1,3}).x_{1,4}\langle \mathtt{M}_4\rangle.d_6\langle \widetilde{u_x} \setminus x_{1,1}x_{1,2}x_{1,3}x_{1,4}\rangle.\mathbf{0}
\end{aligned}$$

By applying $[\![\cdot]\!]$, and thanks to Thm. 9 and Thm. 10, we have:

$$[\![\mathfrak{D}(P_{ti})]\!] = [\![\mathfrak{D}(P_{to})]\!] = C_{\widetilde{x}\widetilde{y}}[[\![\mathfrak{D}_1]\!] \parallel [\![\mathfrak{D}_2]\!] \parallel [\![\mathfrak{D}_3]\!] \parallel [\![\mathfrak{D}_4]\!] \parallel [\![R_x]\!] \parallel [\![R_y]\!]]$$

We now represent the variants of the timed pattern above (i.e., (a) and (b)) using $[\![\cdot]\!]$:

(1) *Interval:* This variant requires that for every sent message, the next message is sent with a delay of at least t_1 time units. This means guarding the snippets obtained with the decomposition so that the processes are suspended until the interval t_1 has passed:

$$\begin{aligned}
Q_1 = C_{\widetilde{x}\widetilde{y}}[[\![\mathfrak{D}_1]\!] \parallel \forall\epsilon\big(clock(u_1) > t_1 \rightarrow [\![\mathfrak{D}_2]\!] \parallel \overline{rst(u_1)} \parallel\\
\forall\epsilon\big(clock(u_1) > t_1 \rightarrow [\![\mathfrak{D}_3]\!] \parallel \overline{rst(u_1)} \parallel\\
\forall\epsilon\big(clock(u_1) > t_1 \rightarrow [\![\mathfrak{D}_4]\!] \parallel \overline{rst(u_1)}\big)\big) \parallel\\
\forall\epsilon\big(clock(u_1) > t_1 \rightarrow [\![R_x]\!]\big)\big) \parallel [\![R_y]\!]]
\end{aligned}$$

Above, constraint $rst(u_1)$ tells the store that clock u_1 must be reset. Process Q_1 consists of nested abstractions. Each abstraction is used to make the next synchronization wait until the delay is satisfied. To achieve this, we guard each abstraction with constraint $clock(u_1) > t_1$. In this way, we ensure that the process representing each prefix is delayed accordingly. We also ensure that whenever the timing constraint is met, the clock is reset to allow for the time to count from the start once again.

(2) *Overall Time-Frame:* This variant can be represented by changing the timing constraint in Q_1 to $clock(u_2) \leq t_2$ and by not resetting the clock inside every abstraction:

$$\begin{aligned}
Q_2 = C_{\widetilde{x}\widetilde{y}}[\forall\epsilon\big(clock(u_2) \leq t_2 \rightarrow [\![\mathfrak{D}_1]\!] \parallel \forall\epsilon\big(clock(u_2) \leq t_2 \rightarrow [\![\mathfrak{D}_2]\!] \parallel\\
\forall\epsilon\big(clock(u_2) \leq t_2 \rightarrow [\![\mathfrak{D}_3]\!] \parallel \forall\epsilon\big(clock(u_2) \leq t_2 \rightarrow [\![\mathfrak{D}_4]\!] \parallel \overline{rst(u_2)}\big)\big) \parallel\\
\forall\epsilon\big(clock(u_2) \leq t_2 \rightarrow [\![R_x]\!]\big)\big)\big) \parallel [\![R_y]\!]]
\end{aligned}$$

The overall time of the communications can be checked because we only reset the clock at the end of the complete interaction.

Both Q_1 and Q_2 preserve the behavior of their source process, assuming that all timing constraints are satisfied. This is because of the operational correspondence property (cf. Thm. 11 and Thm. 12). Similarly to the `lcc` specification of the request-response timeout, we can use non-determinism in `lcc` to extend the behavior of the declarative implementations.

6 Related Work

The most related work is [31], already discussed, which uses `utcc` as target language in a translation of a session π-calculus different from π. By using `lcc` rather than `utcc`, we can correctly encode processes that cannot be represented in [31] (cf. Ex. 3). Also, linearity in `lcc` allows us to provide operational correspondence results (cf. Thm. 11 and Thm. 12) stronger than those in [31].

Haemmerlé [27] develops an encoding of an asynchronous π-calculus into `lcc`, and establishes operational correspondence for it. Since his encoding concerns two asynchronous models, this operational correspondence is more direct than in our case. Monjaraz and Mariño [35] encode the asynchronous π-calculus into Flat Guarded Horn Clauses. They consider compositionality and operational correspondence criteria, as we do here. In contrast to [27,35], here we consider a session π-calculus with synchronous communication, which adds challenges in the translation and its associated correctness proofs. The developments in [27,35] are not concerned with the analysis of message-passing programs in general, nor with session-based concurrency in particular.

Loosely related to our work are [8,18]. Bocchi et al. [8] integrate declarative requirements into *multiparty* session types by enriching (type-based) protocol descriptions with *logical assertions* which are globally specified within multiparty protocols and potentially projected onto specifications for local participants. Rather than a declarative process model based on constraints, the target process language in [8] is a π-calculus with predicates for checking (outgoing and incoming) communications. It should be interesting to see if such an extended session π-calculus can be encoded into `lcc` by adapting our encoding. Also in the context of choreographies, although in a different vein, Carbone et al. [18] explore declarative reasoning via a variant of Hennessy-Milner logic for global specifications.

Prior works on Web service contracts have integrated operational descriptions (akin to CCS specifications) and constraints, where constraint entailment represents the possibility for a service to comply with the requirements of a requester. Buscemi and Montanari's CC-pi [12,14] combines the message-passing communication model from the π-calculus with operations over a store as in `ccp` languages. Analysis techniques for CC-pi processes exploit behavioral equivalences [13]; logical characterizations of process behavior have not been studied. A challenge for obtaining such characterizations is CC-pi's *retract* construct, which breaks the monotonicity requirements imposed for stores in the `ccp` model. We do not know of any attempts on applying session-type analysis for specifications in CC-pi.

Coppo and Dezani-Ciancaglini [20] extend the session π-calculus in [29] with constraint handling operators, such as tells, asks and constraint checks. Session initiation is then bound to the satisfaction of constraint in the store. The merge of constraints and a session type system guarantees *bilinearity*, i.e. channels in use remain private, and that the communications proceed according to the prescribed session types. It is worth noticing that the underlying store in [20] is not linear, which can create potential races among different service providers.

The interplay of constraints and contracts has been also studied by Buscemi et al. [10]. In their model, service interactions follow three phases: negotiation, commitment, and execution. In the negotiation phase, processes agree in fulfilling certain desired behaviors, without guarantee of success. Once committed, it is guaranteed that process execution will honor promised behaviors. The model in [10] uses two languages: a variant of CCS is used as a source language, where the behavior of services and clients is specified; these specifications are then compiled to a language based on CC-pi with no retraction operator, where constraints ensure that interactions between clients and services do not deadlock. It would

be insightful to enrich this two-level model by using linear constraints as in lcc, so as to refine the consumption of resources in the environment.

Bartoletti et al. [4,2] promote contract-oriented computing as a novel vision for enforcing service behaviors at runtime. The premise is that in scenarios where third-party components can be used but not inspected, verification based on (session) types becomes a challenge. Contracts exhibit promises about the expected runtime behavior of each component; they can be used to establish new sessions (contract negotiation) and to enforce that components abide to their promised behavior (honesty). The calculus for contracting processes is based on PCL, a propositional contract logic with a contractual form of implication [4]; this enables to express multiparty assume-guarantee specifications where services only engage in a communication once there are enough guarantees that their requirements will be fulfilled. PCL is used as the underlying constraint system for the contract language used in [4], a variant of ccp with name-passing primitives. In [3], the expressive power of the contract calculus is analyzed with respect to the synchronous π-calculus; name-invariance, compositionality, and operational correspondence are established, as we do here. In [2] the authors introduce CO_2, a generic framework for contract-oriented computing. A characterization of contracts as processes and as formulas in PCL has been developed.

More applications of the encoding herein presented can be found in Cano's PhD thesis [15]. They include an extension of π with session establishment, which is encoded into an extension of lcc with private information, following [28]. Moreover, the thesis [15] also includes an extended account of the work in [16], in which different variants of the session π-calculus in [45] are encoded into the reactive language ReactiveML [32].

7 Concluding Remarks

We have presented an *encoding* of the session π-calculus π into lcc, a process language based on the ccp model. Our encoding is insightful because lcc and π are very different: lcc is declarative, whereas π is operational; communication in lcc is asynchronous, based on a shared memory, whereas communication in π is point-to-point, based on message passing. Our encoding reconciles these differences, and explains precisely how to simulate the operational behavior of π using declarative features in lcc. In a nutshell, our encoding "decouples" point-to-point communication in π by exploiting synchronization on two constraints. Remarkably, because lcc treats constraints as linear, consumable resources we can correctly represent well-typed π processes that should feature linear behavior—communication actions governed by session types must occur exactly once. Thus, linearity sharply arises as the common trait in our expressiveness results.

The strong correctness properties that we establish for our encoding demonstrate that lcc can provide a unified account of operational and declarative requirements in message-passing programs. We have followed the encodability criteria by Gorla [26], namely *name invariance*, *compositionality*, *operational correspondence*, and *success sensitiveness*. In particular, our encoding enjoys the exact same formulation of operational correspondence defined in [26]. These correctness properties guarantee that the behavior of source terms is preserved and reflected appropriately by target terms.

The correctness properties of our encoding hold for π processes that are well-typed. Types not only allow us to concentrate on a meaningful class of source processes; they also allow us to address the differences between π and lcc, already mentioned. In fact, well-typed π processes have a syntactic structure that can be precisely characterized and is stable under reductions. Moreover, the compositional nature of our encoding ensures that this

structure is retained by translated `lcc` processes (target terms) and turns out to be essential in analyzing their behavior. In this analysis, we reconstructed the behavior of source processes via the constraints that their corresponding target terms consume or add to the store during reductions. As such, this reconstruction is enabled by observables induced by the semantics of `lcc`. By combining information about the syntactic structure and the observable behavior of target terms, we were able to establish several invariant properties which are in turn central to prove operational correspondence, in particular soundness.

Well-typed session π-calculi processes can contain rather liberal forms of non-deterministic behavior, which are enabled by the unrestricted types in π (cf. [45]). The soundness property for our translation (Thm. 12) holds for a sub-class of the well-typed processes in [45], namely those without *output races*. We identified this sub-class in a relatively simple way, via a specialization of the predicates that govern typing in [45]. We conjecture that the machinery we have developed for proving soundness can be extended to the whole class of typable processes in [45], provided some additional mechanism that circumvents the value ambiguities mentioned in § 2. We leave this interesting open question for follow-up work.

As application of our results and approach, we have shown how to use our encoding to represent relevant timed patterns in communication protocols, as identified by Neykova et al. [36]. Such timed patterns are commonly found in practice (see, e.g., [19]). Hence, they serve as a valuable validation for our approach. Indeed, thanks to the operational correspondence and compositionality properties, translations of π processes can be used as "black boxes" whose behavior correctly mimics the source terms. These boxes can be plugged inside `lcc` contexts to obtain specifications that exhibit features not easily representable in π. This way, we can analyze message-passing programs in the presence of partial and contextual information.

Acknowledgments We are most grateful to the anonymous reviewers, whose precise, insightful remarks and suggestions helped us to substantially improve the paper.

References

1. Alen Arslanagic, Jorge A. Pérez, and Erik Voogd. Minimal Session Types (Pearl). In *33rd European Conference on Object-Oriented Programming, ECOOP 2019, July 15-19, 2019, London, United Kingdom.*, pages 23:1–23:28, 2019.
2. Massimo Bartoletti, Emilio Tuosto, and Roberto Zunino. Contract-oriented computing in CO2. *Sci. Ann. Comp. Sci.*, 22(1):5–60, 2012.
3. Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. Technical Report DISI-09-056, University of Trento, 2009.
4. Massimo Bartoletti and Roberto Zunino. A calculus of contracting processes. In *Proceedings of the 25th Annual IEEE Symposium on Logic in Computer Science, LICS 2010, 11-14 July 2010, Edinburgh, United Kingdom*, pages 332–341, 2010.
5. Jesper Bengtson, Magnus Johansson, Joachim Parrow, and Björn Victor. Psi-calculi: a framework for mobile processes with nominal data and logic. *Logical Methods in Computer Science*, 7(1), 2011.
6. Giovanni Bernardi, Ornela Dardha, Simon J. Gay, and Dimitrios Kouzapas. On duality relations for session types. In *Trustworthy Global Computing - 9th International Symposium, TGC 2014, Rome, Italy*, pages 51–66, 2014.
7. Giovanni Bernardi and Matthew Hennessy. Using higher-order contracts to model session types. *Logical Methods in Computer Science*, 12(2), 2016.
8. Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR 2010*, volume 6269 of *LNCS*, pages 162–176. Springer - Verlag, 2010.
9. Laura Bocchi, Maurizio Murgia, Vasco Thudichum Vasconcelos, and Nobuko Yoshida. Asynchronous timed session types - from duality to time-sensitive processes. In *Programming Languages and Systems*

- 28th European Symposium on Programming, ESOP 2019, Prague, Czech Republic, Proceedings, pages 583–610, 2019.
10. Maria Grazia Buscemi, Mario Coppo, Mariangiola Dezani-Ciancaglini, and Ugo Montanari. Constraints for service contracts. In *Trustworthy Global Computing - 6th International Symposium, TGC 2011, Aachen, Germany, June 9-10, 2011. Revised Selected Papers*, pages 104–120, 2011.
11. Maria Grazia Buscemi and Hernán C. Melgratti. Transactional service level agreement. In *Trustworthy Global Computing, Third Symposium, TGC 2007, Sophia-Antipolis, France, November 5-6, 2007, Revised Selected Papers*, pages 124–139, 2007.
12. Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint-based language for specifying service level agreements. In *ESOP 2007*, volume 4421 of *LNCS*, pages 18–32. Springer, 2007.
13. Maria Grazia Buscemi and Ugo Montanari. Open bisimulation for the concurrent constraint pi-calculus. In *Programming Languages and Systems, 17th European Symposium on Programming, ESOP 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, pages 254–268, 2008.
14. Maria Grazia Buscemi and Ugo Montanari. Cc-pi: A constraint language for service negotiation and composition. In *Results of the SENSORIA Project*, volume 6582 of *LNCS*, pages 262–281. Springer, 2011.
15. Mauricio Cano. *Session-Based Concurrency: Between Operational and Declarative Views*. PhD thesis, University of Groningen, 2020.
16. Mauricio Cano, Jaime Arias, and Jorge A. Pérez. Session-Based Concurrency, Reactively. In Ahmed Bouajjani and Alexandra Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, volume 10321 of *Lecture Notes in Computer Science*, pages 74–91. Springer, 2017.
17. Mauricio Cano, Camilo Rueda, Hugo A. López, and Jorge A. Pérez. Declarative interpretations of session-based concurrency. In *Proc. of the International Symposium on Principles and Practice of Declarative Programming (PPDP) 2015*, pages 67–78. ACM, 2015.
18. Marco Carbone, Davide Grohmann, Thomas T. Hildebrandt, and Hugo A. López. A logic for choreographies. In *Proc. of PLACES 2010*, pages 29–43, 2010.
19. Christian Colombo, Gordon J. Pace, and Gerardo Schneider. LARVA — safer monitoring of real-time java programs (tool paper). In *Seventh IEEE International Conference on Software Engineering and Formal Methods, SEFM 2009, Hanoi, Vietnam, 23-27 November 2009*, pages 33–37, 2009.
20. Mario Coppo and Mariangiola Dezani-Ciancaglini. Structured communications with concurrent constraints. In *Proc. of TGC 2008*, volume 5474 of *LNCS*, pages 104–125. Springer, 2009.
21. Juan F. Díaz, Camilo Rueda, and Frank D. Valencia. Pi+- calculus: A calculus for concurrent processes with constraints. *CLEI Electron. J.*, 1(2), 1998.
22. François Fages, Paul Ruet, and Sylvain Soliman. Linear concurrent constraint programming: Operational and phase semantics. *Inf. Comput.*, 165(1):14–41, 2001.
23. Simon J. Gay, Peter Thiemann, and Vasco T. Vasconcelos. Duality of session types: The final cut. In Stephanie Balzer and Luca Padovani, editors, *Proceedings of the 12th International Workshop on Programming Language Approaches to Concurrency- and Communication-cEntric Software, PLACES@ETAPS 2020, Dublin, Ireland, 26th April 2020*, volume 314 of *EPTCS*, pages 23–33, 2020.
24. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50:1–102, 1987.
25. Daniele Gorla. A taxonomy of process calculi for distribution and mobility. *Distributed Comput.*, 23(4):273–299, 2010.
26. Daniele Gorla. Towards a unified approach to encodability and separation results for process calculi. *Inf. Comput.*, 208(9):1031–1053, 2010.
27. Rémy Haemmerlé. Observational equivalences for linear logic concurrent constraint languages. *TPLP*, 11(4-5):469–485, 2011.
28. Thomas T. Hildebrandt and Hugo A. López. Types for secure pattern matching with local knowledge in universal concurrent constraint programming. In *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, pages 417–431, 2009.
29. Kohei Honda, Vasco T. Vasconcelos, and Makoto Kubo. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *Proc. of ESOP'98*, volume 1381, pages 122–138. Springer, 1998.
30. J. Klensin. Simple mail transfer protocol. Last Accessed: July, 2019. URL: https://tools.ietf.org/html/rfc5321, October 2008.
31. Hugo A. López, Carlos Olarte, and Jorge A. Pérez. Towards a unified framework for declarative structured communications. In *PLACES 2009, York, UK, 22nd March 2009.*, volume 17 of *EPTCS*, pages 1–15, 2009.

32. Louis Mandel and Marc Pouzet. ReactiveML: a reactive extension to ML. In *Proc. of PPDP'05*, pages 82–93. ACM, 2005.
33. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, I. *Inf. Comput.*, 100(1):1–40, 1992.
34. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, II. *Inf. Comput.*, 100(1):41–77, 1992.
35. Rubén Monjaraz and Julio Mariño. From the π-calculus to flat GHC. In *Proc. of PPDP'12*, pages 163–172. ACM, 2012.
36. Rumyana Neykova, Laura Bocchi, and Nobuko Yoshida. Timed runtime monitoring for multiparty conversations. *Formal Asp. Comput.*, 29(5):877–910, 2017.
37. Carlos Olarte and Frank D. Valencia. Universal concurrent constraint programing: symbolic semantics and applications to security. In *Proceedings of the 2008 ACM Symposium on Applied Computing (SAC), Fortaleza, Ceara, Brazil, March 16-20, 2008*, pages 145–150, 2008.
38. Joachim Parrow. Trios in concert. In *Proof, Language, and Interaction, Essays in Honour of Robin Milner*, pages 623–638, 2000.
39. Joachim Parrow. Expressiveness of process algebras. *Electr. Notes Theor. Comput. Sci.*, 209:173–186, 2008.
40. Kirstin Peters. Comparing process calculi using encodings. In Jorge A. Pérez and Jurriaan Rot, editors, *Proceedings Combined 26th International Workshop on Expressiveness in Concurrency and 16th Workshop on Structural Operational Semantics, EXPRESS/SOS 2019, Amsterdam, The Netherlands, 26th August 2019*, volume 300 of *EPTCS*, pages 19–38, 2019.
41. Kirstin Peters and Uwe Nestmann. Is it a "good" encoding of mixed choice? In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 210–224, 2012.
42. Benjamin C. Pierce. *Types and programming languages*. MIT Press, 2002.
43. Vijay A. Saraswat. *Concurrent constraint programming*. ACM Doctoral dissertation awards. MIT Press, 1993.
44. Sylvain Soliman. Pi-calcul et lcc, une odyssée de l'espace. In *Programmation en logique avec contraintes, JFPLC 2004, June 21-23 2004, Angers, France*, 2004.
45. Vasco T. Vasconcelos. Fundamentals of session types. *Inf. Comput.*, 217:52–70, 2012.
46. Nobuko Yoshida, Raymond Hu, Rumyana Neykova, and Nicholas Ng. The scribble protocol language. In *TGC 2013, Buenos Aires, Argentina, August 30-31, 2013, Revised Selected Papers*, pages 22–41, 2013.

A Additional Examples

In this appendix we further develop the examples presented in the main text.

Example 7 (Observables in the translation (cf. Def. 22)) We recall the processes, translations, and observables in Ex. 4:

$$P_3 = (\boldsymbol{\nu} xy)(x \triangleleft \mathit{buy}.\, x\langle 5406\rangle.\, x(\mathit{inv}).\mathbf{0} \mid y \triangleright \{\mathit{buy}\colon y(w).y\langle \mathsf{invoice}\rangle.\mathbf{0}, \mathit{quit}\colon y(w').\mathbf{0}\}) \tag{11}$$

The translation of P_3 is then given below:

$$\begin{aligned}[\![P_3]\!] = \exists x, y.\big(\, !\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{sel}(x,\mathit{buy})} \parallel \forall u_1(\mathsf{bra}(u_1,\mathit{buy}) \otimes \{x{:}u_1\} \to \\ \overline{\mathsf{snd}(x,5406)} \parallel \forall u_2(\mathsf{rcv}(u_2,5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!])) \parallel \\ \forall l, w(\mathsf{sel}(w,l) \otimes \{w{:}y\} \to \overline{\mathsf{bra}(y,l)} \parallel \\ \forall \epsilon(l = \mathit{buy} \to \forall w_1, w(\mathsf{snd}(w_1,w) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y,w)} \parallel [\![y\langle \mathsf{invoice}\rangle.\mathbf{0}]\!])) \parallel \\ \forall \epsilon(l = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2,u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y,u)} \parallel [\![y(w').\mathbf{0}]\!]))))\end{aligned}$$

with observables:

$$\begin{aligned}\mathcal{O}^{\mathcal{D}^\star_\pi}([\![P_3]\!]) =& \{(\exists x, y.\mathsf{sel}(x,\mathit{buy})), (\exists x, y.\mathsf{bra}(y,\mathit{buy})), (\exists x, y.\mathsf{snd}(x,5406)), \\ & (\exists x, y.\mathsf{rcv}(y,5406)), (\exists x, y.\mathsf{snd}(y,\mathsf{invoice})), (\exists x, y.\mathsf{rcv}(x,\mathsf{invoice})), (\exists x, y.\mathtt{tt})\} \\ \mathcal{O}^{\mathcal{D}_\pi}([\![P_3]\!]) =& \{(\exists x, y.\mathsf{sel}(x,\mathit{buy})), (\exists x, y.\mathsf{snd}(x,5406)), (\exists x, y.\mathsf{snd}(y,\mathsf{invoice}))\}\end{aligned}$$

Below, we analyze the single τ-transition for the translation of P_3 (i.e., $[\![P_3]\!] \longrightarrow_\ell S_1$):

$$\begin{aligned} S_1 = \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{bra}(y, \mathit{buy})} \parallel \forall u_1(\mathsf{bra}(u_1, \mathit{buy}) \otimes \{x{:}u_1\} \to \overline{\mathsf{snd}(x, 5406)} \parallel \\ \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!])) \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{buy} \to \forall w_1, w(\mathsf{snd}(w_1, w) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y, w)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!])) \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w').\mathbf{0}]\!]))\big) \end{aligned}$$

Let us now consider the output observables of S_1:

$$\mathcal{O}^{\mathcal{D}_\pi}(S_1) = \{\exists x, y.\mathsf{snd}(x, 5406), \exists x, y.\mathsf{snd}(y, \mathsf{invoice})\}$$

For the sake of comparison, consider the reduction of P_3, using Rule $\lfloor$SEL$\rfloor$, which discards the second labeled branch:

$$P_3 \longrightarrow (\boldsymbol{\nu} xy)(x\langle 5406\rangle.\, x(\mathit{inv}).\mathbf{0} \mid y(w).y\langle\mathsf{invoice}\rangle.\mathbf{0}) = P_3'$$

The translation of P_3' is as follows (we also show its reduction):

$$\begin{aligned} [\![P_3']\!] = \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{snd}(x, 5406)} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!]) \parallel \\ \forall w_1, w(\mathsf{snd}(w_1, w) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y, w)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!])\big) \\ \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!]) \parallel \overline{\mathsf{rcv}(y, 5406)} \parallel \\ [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!]\big) = S' \end{aligned}$$

and it is easy to see that $\mathcal{O}^{\mathcal{D}_\pi}(S_1) = \mathcal{O}^{\mathcal{D}_\pi}([\![P_3']\!]) = \{\exists x, y.\mathsf{snd}(x, 5406), \exists x, y.\mathsf{snd}(y, \mathsf{invoice})\}$. We will now show that $S_1 \longrightarrow_\ell^2 S_2$ and $\mathcal{O}^{\mathcal{D}_\pi}(S_1) = \mathcal{O}^{\mathcal{D}_\pi}(S_2)$. This step illustrates the fact that intermediate steps reduce to processes that are o-barbed congruent to their respective translations:

$$\begin{aligned} S_1 \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{snd}(x, 5406)} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!]) \parallel \\ \forall \epsilon(\mathit{buy}{=}\mathit{buy} \to \forall w_1, w(\mathsf{snd}(w_1, w){\otimes}\{w_1{:}y\} \to \overline{\mathsf{rcv}(y, w)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!])) \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w').\mathbf{0}]\!]))\big) \\ \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathsf{snd}(x, 5406)} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406) \otimes \{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!]) \parallel \\ \forall w_1, w(\mathsf{snd}(w_1, w) \otimes \{w_1{:}y\} \to \overline{\mathsf{rcv}(y, w)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!]) \parallel \\ \forall \epsilon(\mathit{buy}{=}\mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u){\otimes}\{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w').\mathbf{0}]\!]))\big) = S_2 \end{aligned}$$

where $\mathcal{O}^{\mathcal{D}_\pi}(S_1) = \mathcal{O}^{\mathcal{D}_\pi}(S_2) = \mathcal{O}^{\mathcal{D}_\pi}([\![P_3']\!]) = \{\exists x, y.\mathsf{snd}(x, 5406), \exists x, y.\mathsf{snd}(y, \mathsf{invoice})\}$.

We now informally argue that $S_2 \cong_\ell^\pi [\![P_3']\!]$. First, we show that $S_2 \approx_\ell^\pi [\![P_3']\!]$ and then argue that for every $\mathcal{D}_\pi\mathcal{C}$-context $C[-]$, $C[S_2] \approx_\ell^\pi C[[\![P_3']\!]]$. To justify the former claim, consider the relation $\mathcal{R} = \{(S_2, [\![P_3']\!]), (S_3, S'), (S_4, S_1'), (S_5, S_2'), (S_6, S_3')\}$, where:

$$\begin{aligned} S_2 \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \forall u_2(\mathsf{rcv}(u_2, 5406){\otimes}\{x{:}u_2\} \to [\![x(\mathit{inv}).\mathbf{0}]\!]) \parallel \overline{\mathsf{rcv}(y, 5406)} \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!] \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)}[\![y(w').\mathbf{0}]\!]))\big) = S_3 \\ \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel [\![x(\mathit{inv}).\mathbf{0}]\!] \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!] \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w').\mathbf{0}]\!]))\big) = S_4 \\ \longrightarrow_\ell S_5 \\ \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathtt{tt}} \parallel \overline{\mathtt{tt}} \parallel \\ \forall \epsilon(\mathit{buy} = \mathit{quit} \to \forall w_2, u(\mathsf{snd}(w_2, u) \otimes \{w_2{:}y\} \to \overline{\mathsf{rcv}(y, u)} \parallel [\![y(w).\mathbf{0}]\!]))\big) = S_6 \\ T' \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel [\![x(\mathit{inv}).\mathbf{0}]\!] \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!]\big) = S_1' \\ \longrightarrow_\ell S_2' \\ \longrightarrow_\ell \exists x, y.\big(!\,\overline{\{x{:}y\}} \parallel \overline{\mathtt{tt}} \parallel \overline{\mathtt{tt}}\big) = S_3' \end{aligned}$$

Notice that we have left out the expansion of the term $[\![x(\mathit{inv}).\mathbf{0}]\!] \parallel [\![y\langle\mathsf{invoice}\rangle.\mathbf{0}]\!]$ in both S_1' and S_4. Indeed, for simplicity, we have omitted the shapes of S_5 and S_2'.

We can see that $\mathcal{R}$ is a weak o-barbed bisimulation. Now, to prove $S \cong^{\pi}_{\ell} [\![P'_3]\!]$ we need to show that for each $\mathcal{D}_\pi\mathcal{C}$-context there exists a weak o-barbed bisimulation that makes the processes equivalent. Def. 16 ensures that contexts can only be formed with $\mathcal{D}_\pi\mathcal{C}$-processes. Hence, we need to only check processes that add/consume constraints that may in turn trigger new process reductions. To see this point, consider process S_6: a context that adds the (inconsistent) constraint $l_1 = l_2$ would wrongly trigger a behavior excluded by the source reduction of Q into Q'. One key point in our formal development concerns excluding this possibility (see Def. 22).

Example 8 (Labeled reductions for `lcc` *(cf. Fig. 13))* This example further clarifies the rôle of uniqueness of constraints in our translation. Consider the following target term:

$$S_4 = C_{\widetilde{x}\widetilde{y}}[[\![x_1\langle v\rangle.P_1]\!] \parallel [\![*\, y_1(z).P_2]\!]]$$

Process S_4 allows us to discuss the labeled semantics introduced for our translation. We have:

$$S_4 \equiv C_{\widetilde{x}\widetilde{y}}[[\![x_1\langle v\rangle.P_1]\!] \parallel [\![y_1(z).P_2]\!] \parallel [\![*\, y_1(z).P_2]\!]]$$

which means that there are two possible labeled transitions for S_4:

$$S_4 \xrightarrow{\mathtt{RP}(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid *\, y_1(z).P_2|\!)^1_{xy}] = S'_4$$
$$S_4 \xrightarrow{\mathtt{IO}(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid y_1(z).P_2|\!)^1_{xy} \parallel [\![*\, y_1(z).P_2]\!]] = S''_4$$

Then it is possible to show that $S'_4 \equiv S''_4$ and that each process can complete the synchronization by taking either an $\mathtt{IO}_1(x,y)$ label or an $\mathtt{RP}_1(x,y)$ label, reaching the same process:

$$S'_4 \xrightarrow{\mathtt{IO}_1(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel [\![*\, y_1(z).P_2]\!]]$$
$$S'_4 \xrightarrow{\mathtt{RP}_1(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel [\![*\, y_1(z).P_2]\!]]$$
$$S''_4 \xrightarrow{\mathtt{IO}_1(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel [\![*\, y_1(z).P_2]\!]]$$
$$S''_4 \xrightarrow{\mathtt{RP}_1(x,y)}_{\ell} C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel [\![*\, y_1(z).P_2]\!]]$$

Example 9 (Process decompositions (see Page 45)) Let us consider the behavior of $\mathfrak{D}(P_h)$, as given in Fig. 17:

$$\begin{aligned}
\mathfrak{D}(P_h) \longrightarrow\ & (\boldsymbol{\nu} c_1 d_1)(\boldsymbol{\nu} c_2 d_2)(\boldsymbol{\nu} c_3 d_3)(\boldsymbol{\nu} c_4 d_4)(\boldsymbol{\nu} c_5 d_5)(\boldsymbol{\nu} c_6 d_6)(\boldsymbol{\nu} c_7 d_7)(\boldsymbol{\nu} x_{1,1} y_{1,1})(\boldsymbol{\nu} x_{1,2} y_{1,2}) \\
& (c_2(\widetilde{u_x}).x_{1,1}\langle \mathtt{REQ}\rangle.d_3\langle x_{1,2}\rangle.\mathbf{0} \mid c_3(x_{1,2}).x_{1,2}(z).d_4\langle z\rangle.\mathbf{0} \mid c_4(w).\mathbf{0} \mid \\
& c_5(\widetilde{u_y}).y_{1,1}(z').d_6\langle z' y_{1,2}\rangle.\mathbf{0} \mid c_6(z' y_{1,2}).y_{1,2}\langle \mathtt{ACK}\rangle.d_7\langle z'\rangle.\mathbf{0} \mid c_7(w).\mathbf{0} \mid \\
& d_2\langle \widetilde{u_x}\rangle.d_5\langle \widetilde{u_y}\rangle.\mathbf{0}) \\
\longrightarrow^2\ & (\boldsymbol{\nu} c_1 d_1)(\boldsymbol{\nu} c_2 d_2)(\boldsymbol{\nu} c_3 d_3)(\boldsymbol{\nu} c_4 d_4)(\boldsymbol{\nu} c_5 d_5)(\boldsymbol{\nu} c_6 d_6)(\boldsymbol{\nu} c_7 d_7)(\boldsymbol{\nu} x_{1,1} y_{1,1})(\boldsymbol{\nu} x_{1,2} y_{1,2}) \\
& (x_{1,1}\langle \mathtt{REQ}\rangle.d_3\langle x_{1,2}\rangle.\mathbf{0} \mid c_3(x_{1,2}).x_{1,2}(z).d_4\langle z\rangle.\mathbf{0} \mid c_4(w).\mathbf{0} \mid \\
& y_{1,1}(z').d_6\langle z' y_{1,2}\rangle.\mathbf{0} \mid c_6(z' y_{1,2}).y_{1,2}\langle \mathtt{ACK}\rangle.d_7\langle z'\rangle.\mathbf{0} \mid c_7(w).\mathbf{0}) \\
\longrightarrow^3\ & (\boldsymbol{\nu} c_1 d_1)(\boldsymbol{\nu} c_2 d_2)(\boldsymbol{\nu} c_3 d_3)(\boldsymbol{\nu} c_4 d_4)(\boldsymbol{\nu} c_5 d_5)(\boldsymbol{\nu} c_6 d_6)(\boldsymbol{\nu} c_7 d_7)(\boldsymbol{\nu} x_{1,1} y_{1,1})(\boldsymbol{\nu} x_{1,2} y_{1,2}) \\
& (x_{1,2}(z).d_4\langle z\rangle.\mathbf{0} \mid c_4(w).\mathbf{0} \mid y_{1,2}\langle \mathtt{ACK}\rangle.d_7\langle z'\rangle.\mathbf{0} \mid c_7(w).\mathbf{0}) \longrightarrow^3 \mathbf{0}
\end{aligned}$$

Thus, $\mathfrak{D}(P_h)$ preserves the causal order of the synchronizations in P_h.

B Appendix for § 3.1

B.1 Additional Definitions and Examples

Def. 7 in the main text gives an inductive definition of duality, which suffices for our purposes. For the sake of completeness, and following Bernardi et al. [6], here we give the more general *coinductive* definition of duality:

Definition 37 (Coinductive Type Duality) Let $\mathcal{T}$ be the closed set of contractive session types. We say that types T_1 and T_2 are dual if the pair (T_1, T_2) is in the largest fixed point of the monotone function $\mathcal{D} : \mathcal{P}(\mathcal{T} \times \mathcal{T}) \to \mathcal{P}(\mathcal{T} \times \mathcal{T})$ defined by:

$$\begin{aligned}\mathcal{D}(\mathcal{R}) \stackrel{\text{def}}{=} & \{(\mathtt{end}, \mathtt{end})\} \cup \{(!U_1.T_2, ?U_1'.T_2') \mid U_1 \sim U_1' \wedge (T_2, T_2') \in \mathcal{R}\} \\ & \cup \{(?U_1.T_2, !U_1'.T_2') \mid U_1 \sim U_1' \wedge (T_2, T_2') \in \mathcal{R}\} \\ & \cup \{(\oplus\{l_i : T_i\}_{i\in I}, \&\{l_i : T_i'\}_{i\in I}) \mid \forall i \in I.(T_i, T_i') \in \mathcal{R}\} \\ & \cup \{(\&\{l_i : T_i\}_{i\in I}, \oplus\{l_i : T_i'\}_{i\in I}) \mid \forall i \in I.(T_i, T_i') \in \mathcal{R}\} \\ & \cup \{(\mu\mathtt{a}.T, T') \mid (T\{\mu\mathtt{a}.T/\mathtt{a}\}, T') \in \mathcal{R}\} \cup \{(T, \mu\mathtt{a}.T') \mid (T, T'\{\mu\mathtt{a}.T'/\mathtt{a}\}) \in \mathcal{R}\}\end{aligned}$$

where $\sim$ denotes equivalence up-to equality of trees (see [42,45]).

We illustrate the type system for π with one further example. For simplicity, we shall omit the application of the rules for boolean values and variables, focusing only on processes:

Example 10 (Typing delegation (cf. Fig. 3)) Consider process P_4 below:

$$P_4 = (\boldsymbol{\nu} wz)(\boldsymbol{\nu} xy)(x\langle z\rangle.w(u').\mathbf{0} \mid *\, y(u).u\langle \mathtt{tt}\rangle.\mathbf{0})$$

The typing derivation tree for P_4 is given below:

$$\text{(T:Res)}\times 2\ \dfrac{\text{(T:Par)}\ \dfrac{D \qquad \text{(T:RIn)}\ \dfrac{\text{(T:Out)}\ \dfrac{\text{(T:Nil)}\ \dfrac{\mathtt{un}^\star(y : *\,?(!\mathtt{bool}), u : \mathtt{end})}{y : *\,?(!\mathtt{bool}), u : \mathtt{end} \vdash \mathbf{0}}}{y : *\,?(!\mathtt{bool}), u :!\mathtt{bool} \vdash u\langle \mathtt{tt}\rangle.\mathbf{0}}}{y : *\,?(!\mathtt{bool}) \vdash *\, y(u).u\langle \mathtt{tt}\rangle.\mathbf{0}}}{x{:}*!(!\mathtt{bool}), y{:}*?(!\mathtt{bool}), w{:}?\mathtt{bool}, z{:}!\mathtt{bool} \vdash x\langle z\rangle.w(u').\mathbf{0} \mid *y(u).u\langle \mathtt{tt}\rangle.\mathbf{0}}}{\vdash (\boldsymbol{\nu} wz)(\boldsymbol{\nu} xy)(x\langle z\rangle.w(u').\mathbf{0} \mid *\, y(u).u\langle \mathtt{tt}\rangle.\mathbf{0})}$$

where the derivation sub-tree D corresponds to:

$$\text{(T:Out)}\ \dfrac{\text{(T:In)}\ \dfrac{\text{(T:WkNil)}\ \dfrac{\text{(T:Nil)}\ \dfrac{\mathtt{un}^\star(y : *\,?(!\mathtt{bool}), u' : \mathtt{bool}, w : \mathtt{end})}{y : *\,?(!\mathtt{bool}), u' : \mathtt{bool}, w : \mathtt{end} \vdash \mathbf{0}}}{x : *\,!(!\mathtt{bool}), y : *\,?(!\mathtt{bool}), u' : \mathtt{bool}, w : \mathtt{end} \vdash \mathbf{0}}}{y : *\,?(!\mathtt{bool}), w :?\mathtt{bool} + x : *\,!(!\mathtt{bool}) \vdash w(u').\mathbf{0}}}{x : *\,!(!\mathtt{bool}), y : *\,?(!\mathtt{bool}), w :?T, z :!\mathtt{bool} \vdash x\langle z\rangle.w(u').\mathbf{0}}$$

B.2 Proofs for the Type System

Here we introduce auxiliary lemmas to prove Thm. 1 (subject reduction). First, we show some basic properties of the typing predicates. The most salient one is that $\mathtt{un}^\star(T)$ does not necessarily imply $\mathtt{un}^\star(\overline{T})$. This occurs because although an input-like type can satisfy $\mathtt{un}^\star(\cdot)$, its dual necessarily satisfies $\mathtt{out}(\cdot)$, and therefore, $\mathtt{un}^\star(\overline{T})$ does not hold. Below, the requirement "$\overline{T}$ is defined" allows us to rule out unnecessary types such as $\mathtt{bool}$ or $\mu.\mathtt{bool}$, whose duality is undefined. For this result, following [45], we shall introduce a predicate $\mathtt{lin}(\cdot)$, which is defined as:

- $\mathtt{lin}(T)$ if and only if true.

This predicate will be useful to characterize well-formed types and it will be useful to simplify some of our proofs.

Lemma 16 (Basic Properties for Types) *Let T be a type such that $\overline{T}$ is defined. Then, all of the following holds:*

(1) If $\mathtt{un}^\star(T)$ then one of the following holds:
  (a) If $T = \mathtt{end}$ or $T = \mathtt{a}$ then $\neg\mathtt{out}(\overline{T})$ holds.
  (b) If $T = \mu\mathtt{a}.T$ or $T = qp$ then $\mathtt{out}(\overline{T})$ holds.

(2) If $\mathtt{lin}(T)$ *then* $\mathtt{lin}(\overline{T})$ *holds.*

Proof The proof of (2) proceeds by induction on the structure of T. All cases are immediate by Def. 7.

We consider (1). By assumption, $\mathtt{un}^\star(T)$ and $\neg\mathtt{out}(T)$ are true. We proceed by induction on the structure of T. The base cases are $T = \mathtt{end}$ and $T = \mathtt{a}$: they are immediate and fall on Item (a) (i.e., $\neg\mathtt{out}(\mathtt{a})$ and $\neg\mathtt{out}(\mathtt{end})$ hold). For the inductive step, we consider two cases: (1) whenever $T = \mu\mathtt{a}.T$, and (2) whenever $T = qp$. Case (1) is immediate by applying the IH. We detail Case (2): by the definition of $\mathtt{un}^\star(\cdot)$ (cf. Def. 6), it must be the case that $\neg\mathtt{out}(T)$ holds and $q = \mathtt{un}$. This implies that $T = ?T_1.T_2$ with $\neg\mathtt{out}(T_2)$ holds or $T = \&\{l_i : T_i\}_{i\in I}$ with $\neg\mathtt{out}(T_i)$ true, for every $i \in I$. For each of these cases we have that $\mathtt{out}(\overline{T})$ is true by Def. 7. □

We show some properties of typing contexts. Let $dom(\Gamma)$ denote the set of variables x such that $x : T$ is in Γ. Let $\mathcal{U}(\cdot)$ be a function defined over typing contexts, such that $\mathcal{U}(\Gamma)$ returns a context Γ' that only contains the entries $x : T \in \Gamma$ where $\mathtt{un}^\star(T)$ holds. Note that Property (2) below takes into account the predicate defined in Def. 6.

Lemma 17 (Properties of Typing Contexts) *Let* $\Gamma = \Gamma_1 \circ \Gamma_2$. *Then all of the following hold:*

(1) $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2)$.
(2) Suppose that $x : qp \in \Gamma \wedge (q = \mathtt{lin} \vee (q = \mathtt{un} \wedge \mathtt{out}(p) = \mathtt{tt}))$. *Then, either* $x : qp \in \Gamma_1$ *and* $x \notin dom(\Gamma_2)$ *or* $x : qp \in \Gamma_2$ *and* $x \notin dom(\Gamma_1)$.
(3) $\Gamma = \Gamma_2 \circ \Gamma_1$.
(4) If $\Gamma_1 = \Delta_1 \circ \Delta_2$ *then* $\Delta = \Delta_2 \circ \Gamma_2$ *and* $\Gamma = \Delta_1 \circ \Delta$.

Proof We prove each item by induction on the structure of Γ, using splitting and predicates $\mathtt{un}^\star(\cdot)$ and $\mathtt{lin}(\cdot)$ appropriately:

(1) The base case is $\Gamma = \emptyset$. Then, $\Gamma_1 = \emptyset$ and $\Gamma_2 = \emptyset$. Moreover, $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2) = \emptyset$. For the inductive step, consider $\Gamma = \Gamma', x : T$. We distinguish two possibilities. If $\mathtt{lin}(T)$ holds but $\mathtt{un}^\star(T)$ does not, then the thesis follows immediately since $x : T \notin \mathcal{U}(\Gamma)$. Otherwise, if $\mathtt{un}^\star(T)$ holds, we have that $\Gamma', x : T = \Gamma_1 \circ \Gamma_2$, and by Def. 8, $x : T \in \Gamma_1$ and $x : T \in \Gamma_2$. By IH, $\Gamma' = \Gamma_1' \circ \Gamma_2'$ and $\mathcal{U}(\Gamma') = \mathcal{U}(\Gamma_1') = \mathcal{U}(\Gamma_2')$. Since $\mathtt{un}^\star(T)$, then $x : T \in \mathcal{U}(\Gamma', x : T)$, $x : T \in \mathcal{U}(\Gamma_1)$, and $x : T \in \mathcal{U}(\Gamma_2)$. Thus, $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2)$.
(2) The base case is $\Gamma = \emptyset$, and is immediate as the empty context does not contain any elements. For the inductive step, assume $\Gamma = \Gamma, x : qp$. There are two cases: if $q = \mathtt{lin}$ then the proof is immediate by Def. 8; if $q = \mathtt{un}$ then, by assumption, $\mathtt{out}(p) = \mathtt{tt}$. This implies that $\mathtt{un}^\star(qp) = \mathtt{ff}$; therefore, since $\mathtt{lin}(qp)$ holds, we can conclude the proof by Def. 8.
(3) Immediate by commutativity of the ',' for contexts.
(4) Immediate by associativity of the '$\circ$' operation (cf. Def. 8). □

Lemma 18 (Unrestricted Weakening) *If* $\Gamma \vdash P$ *and* $\mathtt{un}^\star(T)$ *then* $\Gamma, x : T \vdash P$.

Proof By induction on the derivation $\Gamma \vdash P$. There are ten cases. The base case is given by Rule (T:NIL), which follows from inversion on the rule, the definition of $\mathtt{un}^\star(\cdot)$ (cf. Def. 6), and by applying Lem. 17(5). For the inductive step it is first necessary to prove a similar result for the two rules dealing with variables (Rules (T:BOOL) and (T:VAR)). This follows by a simple case analysis and inversion on the corresponding rule (while applying Lem. 17(5)). Using the result for variables, the inductive step for the statement above follows by inversion, applying the IH to the hypotheses obtained, and by reapplying the necessary rule. □

Lemma 19 (Strengthening) *Let* $\Gamma \vdash P$ *and* $x \notin \mathsf{fv}_\pi(P)$. *Then the following holds:*

(1) If $x : qp \wedge (q = \mathtt{lin} \vee (q = \mathtt{un} \wedge \mathtt{out}(p) = \mathtt{tt}))$ *then* $x : qp \notin \Gamma$.
(2) If $\Gamma = \Gamma', x : T$ *and* $\mathtt{un}^\star(T)$ *then* $\Gamma' \vdash P$.

Proof Each item proceeds by induction on the typing derivation $\Gamma \vdash P$ and there are ten cases. First, we must establish a similar result for values, which follows by a case analysis on the applied rule. For Item 1 notice that the Rule (T:NIL) follows immediately, because predicate $\mathtt{un}^\star(\cdot)$ rules out the possibility of $x : qp \in \Gamma$; the inductive cases proceed by applying the IH. In Item 2; the base case proceeds by considering that $x \notin \mathsf{fv}_\pi(\mathbf{0})$, and $\mathtt{un}^\star(\Gamma')$ still holds. All the inductive cases proceed by applying the IH. □

We now state the *subject congruence* property and the *substitution lemma* for our type system.

Lemma 20 (Subject Congruence) *If $\Gamma \vdash P$ and $P \equiv_\pi Q$ then $\Gamma \vdash Q$.*

Proof By a case analysis on the typing derivation for each member of each axiom for $\equiv_\pi$. The most interesting ones are: (1) $P \mid \mathbf{0} \equiv_\pi P$, and (2) $(\boldsymbol{\nu} xy)(P \mid Q)$ if $x, y \notin \mathsf{fv}_\pi(Q)$.

Case $P \mid \mathbf{0} \equiv_\pi P$:

$\Rightarrow$) (1) $\Gamma \vdash P \mid \mathbf{0}$ (Assumption)
(2) $\Gamma = \Gamma_1 \circ \Gamma_2$ (Inv. on Rule (T:PAR), (1))
(3) $\Gamma_1 \vdash P$ (Inv. on Rule (T:PAR), (2))
(4) $\Gamma_2 \vdash \mathbf{0}$ (Inv. on Rule (T:PAR), (2))
(5) $\mathtt{un}^\star(\Gamma_2)$ (Inv. on Rule (T:NIL) to (4))
(6) $\Gamma_1 \circ \Gamma_2 \vdash P$ (Lem. 18 to (3), (5))

$\Leftarrow$) (1) $\Gamma \vdash P$ (Assumption)
(2) $\emptyset \vdash \mathbf{0}$ (Rule (T:NIL) to $\emptyset$ - $\mathtt{un}^\star(\emptyset)$ is true)
(3) $\Gamma \vdash P \mid \mathbf{0}$ (Rule (T:PAR) to (1), (2))

Case $(\boldsymbol{\nu} xy)P \mid Q \equiv_\pi (\boldsymbol{\nu} xy)(P \mid Q)$ with $x, y \notin \mathsf{fv}_\pi(Q)$:

$\Rightarrow$) (1) $\Gamma \vdash (\boldsymbol{\nu} xy)P \mid Q$ (Assumption)
(2) $x, y \notin \mathsf{fv}_\pi(Q)$ (Assumption)
(3) $\Gamma = \Gamma_1 \circ \Gamma_2$ (Inv. on Rule (T:PAR), (1))
(4) $\Gamma_1 \vdash (\boldsymbol{\nu} xy)P$ (Inv. on Rule (T:PAR), (3))
(5) $\Gamma_2 \vdash Q$ (Inv. on Rule (T:PAR), (3))
(6) $\Gamma_1, x : T, y : \overline{T} \vdash P$ (Inv. on Rule (T:RES) on (4))

We now distinguish two cases. The first (and most interesting) one is when $\mathtt{un}^\star(T)$ is true. The second one corresponds to when $\mathtt{un}^\star(T)$ is false (i.e., which groups the remaining possibilities):

$\mathtt{un}^\star(T)$ is true: We distinguish two further sub-cases depending on whether $\mathtt{un}^\star(\overline{T})$ holds or not:
- $\mathtt{un}^\star(\overline{T})$ is true: We apply Lem. 18 twice in (6) to obtain (7) $\Gamma_2, x : T, y : \overline{T} \vdash Q$ and conclude by applying (T:PAR) and (T:RES) on (5) and (7).
- $\mathtt{un}^\star(\overline{T})$ is false: We only apply weakening once (cf. Lem. 18) in (6) to obtain (7) $\Gamma_2, x : T \vdash Q$ and conclude by applying (T:PAR) and (T:RES) to (5) and (7).

$\mathtt{un}^\star(T)$ is false: Then $\mathtt{lin}(T)$ holds; by Lem. 16(2), $\mathtt{lin}(\overline{T})$ also holds. Thus, we distinguish two cases, depending on whether $\mathtt{un}^\star(\overline{T})$ also holds or not. Each case proceeds similarly as above.

$\Leftarrow$) (1) $\Gamma \vdash (\boldsymbol{\nu} xy)(P \mid Q)$ (Assumption)
(2) $x, y \notin \mathsf{fv}_\pi(Q)$ (Assumption)
(3) $\Gamma, x : T, y : \overline{T} \vdash P \mid Q$ (Inv. on Rule (T:RES), (1))

We distinguish two further cases depending on whether $\mathtt{un}^\star(T)$ is true or false:

$\mathtt{un}^\star(T)$ is true: We distinguish two further sub-cases depending on whether the $\mathtt{un}^\star(\overline{T})$ holds or not:
- $\mathtt{un}^\star(\overline{T})$ is true: Then by inversion on (T:PAR) on (3), we have (4) $\Gamma, x : T, y : \overline{T} \vdash P$ and (5) $\Gamma_2, x : T, y : \overline{T} \vdash Q$. Then, we apply Rule (T:RES) on (4) to obtain (6) $\Gamma_1 \vdash (\boldsymbol{\nu} xy)P$ and we apply Lem. 19(2) to (5), obtaining (7) $\Gamma_2 \vdash Q$. We conclude by applying Rule (T:PAR) to (6) and (7).
- $\mathtt{un}^\star(\overline{T})$ is false: Then by inversion on (T:PAR) on (3), we have (4) $\Gamma, x : T, y : \overline{T} \vdash P$ and (5) $\Gamma_2 \vdash Q$. Moreover, by Lem. 19(1) applied to (2) and (5), it must be the case that $y : \overline{T} \notin \Gamma_2$. Therefore, we apply Lem. 19(2) once to remove $x : T$ from Γ_2. Then, we apply Rule (T:RES) in (4) to obtain (6) $\Gamma \vdash (\boldsymbol{\nu} xy)P$ and conclude by applying Rule (T:PAR).

$\mathtt{un}^\star(T)$ is false: As previously described, from Lem. 16(2), we distinguish two cases depending on $\mathtt{un}^\star(\overline{T})$. Each case proceeds similarly as above. □

Lemma 21 (Substitution) *If $\Gamma_1 \vdash v : T$ and $\Gamma_2, x : T \vdash P$ then $\Gamma \vdash P\{v/x\}$, with $\Gamma = \Gamma_1 \circ \Gamma_2$.*

Proof By induction on the structure of P. There are nine cases: one base case and eight inductive cases.

Base Case: $P = \mathbf{0}$. By inversion, $\mathtt{un}^\star(\Gamma_2, x : T)$, which implies $\mathtt{un}^\star(T)$. By inversion on the rules for values, we also have that $\mathtt{un}^\star(\Gamma_1)$. Thus, $\mathtt{un}^\star(\Gamma)$, with $\Gamma = \Gamma_1 \circ \Gamma_2$ holds. Since $\mathbf{0}\{v/x\} = \mathbf{0}$, the proof concludes by applying Rule (T:NIL).

Inductive Step: The proofs for $P = Q_1 \mid Q_2$, $P = (\boldsymbol{\nu} yz)Q$, and $P = u?\, Q_1 : Q_2$ follow by applying the IH. The other five cases proceed similarly. We only detail the case for $P = y\langle u \rangle.Q$.

$P = y\langle u \rangle.Q$: We distinguish four sub-cases: (1) $y = x$ and $\neg\mathtt{un}^\star(T)$, (2) $y = x$ and $\mathtt{un}^\star(T)$, (3) $u = x$ and $\neg\mathtt{un}^\star(T)$, (4) $u = x$ and $\mathtt{un}^\star(T)$. We only detail sub-cases (1), (2) and (4) since sub-case (3) proceeds similarly to (1) and (4).

1. By assumption, $P = x\langle u\rangle.Q$ and $\neg\mathtt{un}^\star(T)$. Moreover, since judgment $\Gamma_2 \vdash P$ can only be obtained with Rule (T:OUT), it must be the case that $T = q!U.U'$. Thus, by inversion on Rule (T:OUT), (1) $\Gamma_2 = \Delta_1 \circ \Delta_2 \circ \Delta_3$, (2) $\Delta_1 \vdash x : q!U.U'$, (3) $\Delta_2 \vdash u : U$, and (4) $\Delta_3 + x : U' \vdash Q$. We distinguish two further cases depending on whether $q = \mathtt{lin}$ or $q = \mathtt{un}$. We only show the case $q = \mathtt{un}$, as the other proceeds similarly. By (2), it must be the case that $x : T \in \Delta_1$. Moreover, by inversion on (2) and (3), we have that $\mathtt{un}^\star(\Delta_1')$ holds with (5) $\Delta_1' = \Delta_1 \setminus x : T$, and that (6) $\mathtt{un}^\star(\Delta_2)$ holds. By Lem. 19(1), we also have that $x : T \notin \Delta_2$ and $x : T \notin \Delta_3$. Moreover, by Lem. 17(1), $\Delta_3 = \Gamma_2$, which implies that $\Delta_3 + x : U' = \Gamma_2, x : U'$. By applying the IH, $\Gamma_1 \circ (\Gamma_2, v : U') \vdash Q\{v/x\}$. We then distinguish two further cases depending on whether $\mathtt{un}^\star(U)$ or $\neg\mathtt{un}^\star(U)$. In both cases we proceed similarly: we add all the missing hypothesis applying Lem. 18 and conclude by reapplying Rule (T:OUT).
2. This sub-case is not possible: by assumption, $\Gamma, x : T \vdash x\langle u\rangle.P$, which means that the judgment proceeds with Rule (T:OUT). Thus, by inversion $T = \mathtt{un}!U.U'$. Now, by Def. 6, $\mathtt{un}^\star(T)$ cannot hold, which contradicts the assumptions of this case (i.e., $y = x$ and $\mathtt{un}^\star(T)$).
4. By assumption, $P = y\langle x\rangle.Q$ and $\mathtt{un}^\star(T)$. Moreover, since judgment $\Gamma_2 \vdash P$ can only be obtained with Rule (T:OUT), it must be the case that $T = q!U.U'$. We distinguish cases depending on whether $\mathtt{un}^\star(U)$ or $\neg\mathtt{un}^\star(U)$ are true. Both cases proceed similarly, so we only detail the latter. If $\mathtt{un}^\star(U)$ holds, we have that (1) $\Gamma_2 = \Delta_1 \circ \Delta_2 \circ \Delta_3$, (2) $\Delta_1 \vdash x : q!U.U'$, (3) $\Delta_2 \vdash x : U$, and (4) $\Delta_3 + y : U' \vdash P$, and $x : U \in \Delta_1$, $x : U \in \Delta_2$, and $x : U \in \Delta_3$. Following a similar line of reasoning as the one above, we conclude that $\Delta_3 = \Gamma_2$ and that $\Delta_1 = \Delta_2$. Moreover, by assumption $\Gamma_1 \vdash v : U$ and by IH $\Gamma_1 \circ (\Gamma_2, v : U, y : U') \vdash P'\{v/x\}$. We then distinguish two further cases depending on whether $\mathtt{un}^\star(U')$ or $\neg\mathtt{un}^\star(U')$. In both cases we conclude similarly as above. □

Theorem 1 (Subject Reduction) *If $\Gamma \vdash P$ and $P \longrightarrow Q$ then $\Gamma \vdash Q$.*

Proof (see Page 13) By induction on the derivation of the reduction $P \longrightarrow Q$. The interesting cases are when the derivation stops with Rules $\lfloor$COM$\rfloor$ and $\lfloor$REP$\rfloor$. The other cases proceed similarly; the case of Rule $\lfloor$STR$\rfloor$ follows by the IH and Lem. 20.

Case $\lfloor$COM$\rfloor$:

(1)	$P = (\boldsymbol{\nu} xy)(x\langle v\rangle.Q_1 \mid y(z).Q_2 \mid Q_3)$	(Assumption)
(2)	$P \longrightarrow (\boldsymbol{\nu} xy)(Q_1 \mid Q_2\{v/z\} \mid Q_3)$	(Assumption)
(3)	$\Gamma \vdash P$	(Assumption)
(4)	$\Gamma, x : T, y : \overline{T} \vdash x\langle v\rangle.Q_1 \mid y(z).Q_2 \mid Q_3$	(Inv. on Rule (T:RES), (1))
(5)	$\Gamma = \Gamma_1 \circ \Gamma_2 \circ \Gamma_3$	(Inv. on Rule (T:PAR), (3))
(6)	$\Gamma_1, x : T \vdash x\langle v\rangle.Q_1$	(Inv. on Rule (T:PAR), (3))
(7)	$\Gamma_2, y : \overline{T} \vdash y(z).Q_2$	(Inv. on Rule (T:PAR), (3))
(8)	$\Gamma_3 \vdash Q_3$	(Inv. on Rule (T:PAR), (3))
(9)	$\Gamma_1 = \Gamma_1' \circ \Gamma_1'' \circ \Gamma_1'''$	(Inv. on Rule (T:OUT), (6))
(10)	$\Gamma_1', x : T \vdash x : \mathtt{un}!U.U'$	(Inv. on Rule (T:OUT), (6))
(11)	$\Gamma_1'' \vdash v : U$	(Inv. on Rule (T:OUT), (6))
(12)	$\Gamma_1''' + x : U' \vdash Q_1$	(Inv. on Rule (T:OUT), (6))
(13)	$\mathtt{un}^\star(\Gamma_1') \wedge \mathtt{un}^\star(\Gamma_1'')$ hold	(Inv. on Rule Val. Rules, (10), (11))
(14)	$\Gamma_1''' = \Gamma_1$	(Lem. 17 to (13), (12))
(15)	$\Gamma_1 + x : U' \vdash Q_1$	((14), (12))
(16)	$(\Gamma_2, z : U') + y : \overline{U'} \vdash Q_2$	(Derived similarly to (15))

By Lem. 16(2), $\mathtt{lin}(T)$ implies $\mathtt{lin}(\overline{T})$. Thus, it is enough to distinguish cases depending on T and $\overline{T}$. We consider these combinations:

Case $\neg\mathtt{un}^\star(T) \wedge \neg\mathtt{un}^\star(\overline{T})$: We have that (a) $\Gamma_1, x : U' \vdash Q_1$, by (15) and the definition of $+$; (b) $\Gamma_2, z : U, y : \overline{U'} \vdash Q_2$, by (16) and the definition of $+$; (c) $\Gamma_3 \vdash Q_3$, by (8), and (d) $\Gamma_1'' \vdash v : U$, by (11). By Lem. 21 (substitution) applied to (d) and (b), we have (e) $\Gamma_2, y : \overline{U'} \vdash Q_2$, and we can finish the proof by applying rules (T:PAR), (T:PAR), (T:RES).

Case $\neg\mathtt{un}^\star(T) \wedge \mathtt{un}^\star(\overline{T})$: We have that (a) $\Gamma_1, x : U' \vdash Q_1$, by (15) and the definition of $+$; $(\Gamma_2, z : U') + y : \overline{U'} \vdash Q_2$ and $y : \mathtt{un}?U.\overline{U'} \in \Gamma_2$, by (16) and Definition of $\mathtt{un}^\star(\cdot)$. Thus, $\overline{U'} = \mathtt{un}?U.\overline{U'}$ and thus, (b) $\Gamma_2, z : U', y : \overline{T} \vdash Q_2$. Similarly as above, we also have (c) $\Gamma_3 \vdash Q_3$, by (8); and (d) $\Gamma_1'' \vdash v : U$, by (11). By Lem. 18 on (a), we have (e) $\Gamma_1, x : U', y : \overline{T} \vdash Q_1$, and we can conclude by applying Lem. 21 and rules (T:PAR), (T:PAR), (T:RES) as above.

Other Cases: Notice that cases (i) $\mathtt{un}^\star(T) \wedge \mathtt{un}^\star(\overline{T})$ and (ii) $\mathtt{un}^\star(T) \wedge \neg\mathtt{un}^\star(\overline{T})$ are not possible, because $T = \mathtt{un}!U.U'$. Therefore, $\mathtt{out}(T) = \mathtt{tt}$ and the definition of $\mathtt{un}^\star(T)$ (cf. Def. 6) would not hold.

Case $\lfloor$REP$\rfloor$: Assuming $P = (\boldsymbol{\nu} xy)(x\langle v\rangle.Q_1 \mid *\, y(z).Q_2 \mid Q_3)$, we proceed similarly as above. By inversion on Rule (T:RES), $\Gamma, x : T, y : \overline{T} \vdash x\langle v\rangle.Q_1 \mid *\, y(z).Q_2 \mid Q_3$, with $\Gamma = \Gamma_1 \circ \Gamma_2 \circ \Gamma_3$. Following a similar derivation as above, we conclude that $T = q!U.U'$, and $\overline{T} = q?U.\overline{U'}$. Then we deduce:

(a) $\Gamma_1 = \Gamma_1' \circ \Gamma_1'' \circ \Gamma_1'''$;
(b) $\Gamma_1''' + x : U' \vdash Q_1$, by inversion Rule (T:OUT);
(c) $\Gamma_1'', x : T \vdash x : q!U.U'$, by inversion Rule (T:OUT);
(d) $\mathtt{un}^\star(\Gamma_1'')$, by inversion Rule (T:VAR);
(e) $\Gamma' \vdash v : U$, by inversion Rule (T:OUT);
(f) $\mathtt{un}^\star(\Gamma_1')$, by inversion Rule (T:VAR) (or (T:BOOL));
(g) $(\Gamma_2, y : \overline{T}, z : U'') + y : \overline{U'} \vdash Q_2$, by inversion Rule (T:RIN);
(h) $\mathtt{un}^\star(\Gamma_2, y : \overline{T}) \wedge (\mathtt{un}^\star(U'') \vee U'' = \mathtt{lin}\ p)$, by inversion Rule (T:RIN);
(i) $\Gamma_3 \vdash Q_3$, by inversion Rule (T:PAR).

By (h), we have two possible cases:

Case $\mathtt{un}^\star(\Gamma_2, y : \overline{T}) \wedge \mathtt{un}^\star(U'')$: we have that $\mathtt{un}^\star(\overline{T})$ holds, which implies: $\neg\mathtt{out}(\overline{T})$ and $\overline{T} = U'$ and $U'' = U$; hence T is a recursive type. Moreover, by Lem. 16(1b), we have that $\mathtt{un}^\star(T)$ does not hold. Hence, we have that: $x : T \notin \Gamma_1'''$, $x : T \notin \Gamma_1'$. Then, by Lem. 17(1) to (a), (d), and (f), we have that $\Gamma_1''' = \Gamma_1$. Then, by applying Lem. 21 and Lem. 18 to (e) and (c), we have that $\Gamma_2, y : \overline{T} \circ \Gamma_1' \circ \Gamma_1'' \vdash Q_2\{v/z\}$. We then apply Lem. 18 to (i) to obtain $(\Gamma_3, y : \overline{T}) \circ \Gamma_1' \circ \Gamma_1'' \vdash Q_3$. Notice that we also know, by Lem. 18, that $\Gamma_1', \Gamma_1'' \circ (\Gamma_2, y : \overline{T}) \vdash *\, y(z).Q_2$. We then apply Rule (T:PAR) (which is applicable, since $\mathtt{un}^\star(\overline{T})$ holds) to the previous hypotheses to obtain:

$$\Gamma, x : T, y : \overline{T} \vdash Q_1 \mid Q_2 \mid *\, y(z).Q_2 \mid Q_3$$

We then conclude by applying Rule (T:RES).

Case $\mathtt{un}^\star(\Gamma_2, y : \overline{T}) \wedge U'' = \mathtt{lin}\ p$: This case proceeds as above. The only difference is that the assumption implies that $v : \mathtt{lin}\ p$. This in turn means that $v : \mathtt{lin}\ p \notin \Gamma_1'''$ and $v : \mathtt{lin}\ p \notin \Gamma_1'$, which allows us to conclude with the same argument as above.

□

Theorem 4 (Type Safety) *If $\vdash P$ then P is well-formed.*

Proof (see Page 14) For the sake of contradiction, assume that $\vdash P$ and P is not well-formed, i.e., it does not satisfy any of the three items in Def. 10. If $\vdash P$ then there exists a derivation

$$\vdash (\boldsymbol{\nu} x_1 \dots x_n y_1 \dots y_n)(Q_1 \mid Q_2 \mid R) \qquad \text{(with } n \geq 1\text{)}$$

Therefore, by applying inversion n times, there exists a context $x_1 : T_n, \dots, x_n : T_n, y_1 : \overline{T}_1, \dots, y_n : \overline{T}_n = \Gamma_1 \circ \Gamma_2 \circ \Gamma_3$ that types $Q_1 \mid Q_2 \mid R$. We now show that if $Q_1 \mid Q_2 \mid R$ does not satisfy any item in Def. 10, then we reach a contradiction:

1. *If $Q_1 = v?\, Q_1 : Q_2$ then $v \in \{\mathtt{tt}, \mathtt{ff}\}$:* Assume that does not satisfy this condition. Then, the derivation $\Gamma_1 \circ \Gamma_2 \circ \Gamma_3 \vdash Q_1 \mid Q_2 \mid R$ is not possible as Rule (T:IF) requires $v : \mathtt{bool}$.
2. *There are not Q_1 and Q_2 such that they are both prefixed with the same variable:* Assume there exist Q_1 and Q_2 that are prefixed in the same variable. Then there are two cases: (i) if the prefixes are of different nature, then we reach a contradiction: it is not possible for $x : T$ and $x : T'$ with $T \neq T'$ to appear in a typing derivation in the same context; (ii) if the prefixes are of the same nature, we just need to notice that splitting does not allow for session types that satisfy $\mathtt{out}(\cdot)$ to be shared among contexts (cf. Def. 8). Thus, only unrestricted input and branching types can be shared.
3. *If Q_1 is prefixed on $x_1 \in \widetilde{x}$ and Q_2 is prefixed on $y_1 \in \widetilde{y}$ then $Q_1 \mid Q_2$ is a redex:* Suppose that this does not hold (i.e., $Q_1 \mid Q_2$ is not a redex). Then, the typing derivation is not possible, since Rule (T:RES) requires the types of two co-variables to be dual, thus reaching a contradiction. □

C Appendix for § 4

C.1 Junk Processes

Lemma 2 *Let J be junk. Then: (1) $J \not\xrightarrow{\tau}_\ell$ (and) (2) there is no $c \in \mathcal{C}$ (cf. Def. 21) such that $J \parallel \overline{c} \xrightarrow{\tau}_\ell$.*

Proof (see Page 28) We prove each item individually.

(1) By induction on the structure of J. We show two cases:
Case $J = \forall\epsilon((b = \neg b) \to P)$: This case is immediate by inspecting the rules in Fig. 6; in particular, since Rule (C:SYNC) cannot be applied $J \not\xrightarrow{\tau}_\ell$.
Case $J = J_1 \parallel J_2$: By IH, $J_1 \not\xrightarrow{\tau}_\ell$ and $J_2 \not\xrightarrow{\tau}_\ell$. We prove that a reduction $J_1 \parallel J_2 \xrightarrow{\tau}_\ell J_1' \parallel J_2'$ cannot occur. By Def. 28, junk processes are either ask-guarded processes or $\overline{\mathtt{tt}}$. To reduce, one of J_1 and J_2 must add a constraint to the store; two parametric ask processes in parallel cannot reduce (cf. Fig. 6). Now, since $\overline{\mathtt{tt}}$ does not add any information to the store, we have that $J_1 \parallel J_2 \not\xrightarrow{\tau}_\ell$.

(2) By induction on the structure of J. We detail three cases:
Case $J = \overline{\mathtt{tt}}$: This case is immediate, as $\overline{\mathtt{tt}} \parallel \overline{c} \not\xrightarrow{\tau}_\ell$, for any constraint $c \in \mathcal{C}$.
Case $J = \forall\epsilon((b = \neg b) \to P)$**:** Suppose, for the sake of contradiction, that there is a $c \in \mathcal{C}$ (cf. Def. 21) such that $J \parallel \overline{c} \xrightarrow{\tau}_\ell S$, for some S. Then, by Def. 21, c must be composed of the predicates snd, rcv, sel, bra, $\{\cdot\vdots\cdot\}$ (cf. Fig. 7), the multiplicative conjunction $\otimes$, replication, and the existential quantifier. We now apply induction on the structure of c: there are five base cases (one for each predicate) and three inductive cases (one for each logical connective). We show only two representative cases:
Sub-case $c = \mathsf{snd}(x, v)$**:** This base case is immediate, as $\mathsf{snd}(x, v) \not\vdash (b = \neg b)$ using the Rules in Fig. 5. Thus, Rule (C:SYNC) is not applicable, therefore contradicting our assumption.
Sub-case $c = c_1 \otimes c_2$**:** By IH, $c_i \not\vdash (b = \neg b)$. Then, by the rules in Fig. 5, $c_1 \otimes c_2 \not\vdash (b = \neg b)$, leading to a contradiction as in the previous sub-case.
Case $J = J_1 \parallel J_2$: Then $J \parallel \overline{c} \not\xrightarrow{\tau}_\ell$ follows immediately from the IH (which ensures $J_1 \parallel \overline{c} \not\xrightarrow{\tau}_\ell$ and $J_2 \parallel \overline{c} \not\xrightarrow{\tau}_\ell$) and Item (1). □

Lemma 3 (Junk Observables) *For every junk process J and every $\mathcal{D}_\pi\mathcal{C}$-context $C[-]$, we have that: (1) $\mathcal{O}^{\mathcal{D}_\pi}(J) = \emptyset$ (and) (2) $\mathcal{O}^{\mathcal{D}_\pi}(C[J]) = \mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}])$.*

Proof (see Page 28) We prove each item separately:

1. By induction on the structure of J. We show the most representative 3 cases:
Case $J = \forall\epsilon((b = \neg b) \to P)$: By Lem. 2(1), $J \not\xrightarrow{\tau}_\ell$. Therefore, by Def. 15 $\mathcal{O}^{\mathcal{D}_\pi}(J) = \emptyset$.
Case $J = \overline{\mathtt{tt}}$: By Lem. 2(1), $J \not\xrightarrow{\tau}_\ell$. Moreover, since $\mathtt{tt} \notin \mathcal{D}_\pi$, by Def. 15, $\mathcal{O}^{\mathcal{D}_\pi}(J) = \emptyset$.
Case $J = J_1 \parallel J_2$: By IH, $\mathcal{O}^{\mathcal{D}_\pi}(J_i) = \emptyset, i \in \{1, 2\}$. By Lem. 2(1), $J \not\xrightarrow{\tau}_\ell$ and therefore, $\mathcal{O}^{\mathcal{D}_\pi}(J) = \mathcal{O}^{\mathcal{D}_\pi}(J_1) \cup \mathcal{O}^{\mathcal{D}_\pi}(J_2) = \emptyset$.
2. The proof is by induction on the structure of J, followed in each case by a case analysis on $C[-]$ (cf. Def. 16). All cases follow from the definitions; we detail two representative cases:
Case $J = \forall\epsilon((b = \neg b) \to P_1)$: We apply a case analysis on context $C[-]$. We will show only two sub-cases, as the third one is symmetrical:
Sub-case $C[-] = \exists\widetilde{x}.\,-$: This case follows immediately from Lem. 2(1,2).
Sub-case $C[-] = - \parallel P_2$: Then $C[J] = \forall\epsilon((b = \neg b) \to P_1) \parallel P_2$. By Lem. 2(1,2), $\forall\epsilon((b = \neg b) \to P_1) \not\xrightarrow{\tau}_\ell$ and there is no $c \in \mathcal{C}$ (cf. Def. 21) such that $J \parallel \overline{c} \xrightarrow{\tau}_\ell$. As such, a reduction $C[J] \xRightarrow{\tau} C'[J']$ is not possible (i.e., J cannot reduce in the context). Therefore, by Def. 15 and Item (1), $\mathcal{O}^{\mathcal{D}_\pi}(C[J]) = \mathcal{O}^{\mathcal{D}_\pi}(J) \cup \mathcal{O}^{\mathcal{D}_\pi}(P_2) = \mathcal{O}^{\mathcal{D}_\pi}(P_2)$. Following a similar analysis, and using Lem. 2(1,2), Item(1) and Def. 15, we have that $\mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}]) = \mathcal{O}^{\mathcal{D}_\pi}(\overline{\mathtt{tt}}) \cup \mathcal{O}^{\mathcal{D}_\pi}(P_2) = \mathcal{O}^{\mathcal{D}_\pi}(P_2)$. We conclude $\mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}]) = \mathcal{O}^{\mathcal{D}_\pi}(C[J])$, as wanted.
Case $J = J_1 \parallel J_2$: We apply a case analysis on context $C[-]$. We will show only two sub-cases, as the third one is symmetrical:
Sub-case $C[-] = \exists\widetilde{x}.(\,-\,)$: This case follows immediately from Lem. 2(1,2).
Sub-case $C[-] = - \parallel P_2$: By IH, we have $\mathcal{O}^{\mathcal{D}_\pi}(C[J_1]) = \mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}])$ and $\mathcal{O}^{\mathcal{D}_\pi}(C[J_2]) = \mathcal{O}^{\mathcal{D}_\pi}(C[\overline{\mathtt{tt}}])$. Also, by Lem. 2(1), $J_1 \parallel J_2 \not\xrightarrow{\tau}_\ell$ and by Lem. 2(2), there is no $c \in \mathcal{C}$ (cf. Def. 21) such that $J \parallel \overline{c} \xrightarrow{\tau}_\ell$. Hence, by Def. 15 and Item (1), $\mathcal{O}^{\mathcal{D}_\pi}(C[J_1 \parallel J_2]) = \mathcal{O}^{\mathcal{D}_\pi}(\overline{\mathtt{tt}})$, as wanted. □

Lemma 4 (Junk Behavior) *For every junk J, every $\mathcal{D}_\pi\mathcal{C}$-context $C[-]$, and every process P, we have $C[P \parallel J] \approx_\ell^\pi C[P]$.*

Proof (see Page 28) By coinduction, i.e., by exhibiting a weak o-barbed bisimulation containing the pair $(C[P \parallel J], C[P])$. To build the needed bisimulation, we recall Def. 17. That is, we must define a symmetric relation $\mathcal{R}$ such that $(R, Q) \in \mathcal{R}$ implies:

1. $\mathcal{O}^{\mathcal{D}_\pi}(R) = \mathcal{O}^{\mathcal{D}_\pi}(Q)$ (and),
2. whenever $R \xrightarrow{\tau}_\ell R'$ there exists Q' such that $Q \xRightarrow{\tau} Q'$ and $R' \mathcal{R} Q'$.

Then, let us consider:

$$\mathcal{R} = \{(R,Q) \mid C[P \parallel J] \xrightarrow{\tau}_\ell{}^{n} R \wedge C[P] \xrightarrow{\tau}_\ell{}^{n} Q, n \geq 0\}$$
$$\cup\ \{(Q,R) \mid C[P \parallel J] \xrightarrow{\tau}_\ell{}^{n} R \wedge C[P] \xrightarrow{\tau}_\ell{}^{n} Q, n \geq 0\} \quad (1)$$

Observe that $\mathcal{R}$ is symmetric by definition. Moreover, it immediately satisfies Item (1) thanks to Lem. 3(2).

As for Item (2), first notice that $(R,Q) \in \mathcal{R}$: we have $R = C[P \parallel J]$ and $Q = C[P]$ (with $n = 0$). Now suppose that $R \xrightarrow{\tau}_\ell R'$; we show a matching transition $Q \xRightarrow{\tau} Q'$ such that $R'\mathcal{R}Q'$. To this end, we use a case analysis on the reduction(s) possible from $C[P \parallel J]$. There are six possibilities:

(a) $C[P \parallel J] \xrightarrow{\tau}_\ell C[P \parallel J']$ (i.e., an autonomous reduction from J);
(b) $C[P \parallel J] \xrightarrow{\tau}_\ell C'[P \parallel J']$ (i.e., a reduction from the interplay of C and J);
(c) $C[P \parallel J] \xrightarrow{\tau}_\ell C[P' \parallel J']$ (i.e., a reduction from the interplay of P and J);
(d) $C[P \parallel J] \xrightarrow{\tau}_\ell C'[P \parallel J]$ (i.e., an autonomous reduction from C);
(e) $C[P \parallel J] \xrightarrow{\tau}_\ell C'[P' \parallel J]$ (i.e., an interaction between C and P);
(f) $C[P \parallel J] \xrightarrow{\tau}_\ell C[P' \parallel J]$ (i.e., an autonomous reduction from P).

Notice that Lem. 2(1,2) and Lem. 3(1,2) exclude cases (a)–(c). Thus, reductions for $C[J]$ will only be of the form (d)–(f). Clearly, this transition from R can be matched by Q as follows:

- $Q = C[P] \xrightarrow{\tau}_\ell C'[P] = Q'$, with $(C'[P \parallel J], C'[P]) \in \mathcal{R}$ (or)
- $Q = C[P] \xrightarrow{\tau}_\ell C'[P'] = Q'$, with $(C'[P' \parallel J], C'[P']) \in \mathcal{R}$ (or)
- $Q = C[P] \xrightarrow{\tau}_\ell C[P'] = Q'$, with $(C[P' \parallel J], C[P']) \in \mathcal{R}$.

With these reductions, we conclude the proof for this case. The case when $Q \xrightarrow{\tau}_\ell Q'$ is similar. □

Lemma 5 (Occurrences of Junk) *Let R be a redex (Def. 9).*

1. *If $R = x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$, with $j \in I$ then:*
   $[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}_\ell{}^{3} \exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!] \parallel J\big)$, *where*
   $J = \prod_{i\in I'} \forall\epsilon(l_j = l_i \to [\![Q_i]\!])$, *with $I' = I \setminus \{j\}$, and*
   $\exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!] \parallel J\big) \cong^\pi_\ell \exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![Q_j]\!]\big)$.
2. *If $R = b?\,P_1 : P_2$, $b \in \{\mathtt{tt},\mathtt{ff}\}$, then:*
   $[\![R]\!] \xrightarrow{\tau}_\ell [\![P_i]\!] \parallel J$, $i \in \{1,2\}$ *with* $J = \forall\epsilon(b = \neg b \to [\![P_j]\!])$, $j \neq i$, *and*
   $[\![P_i]\!] \parallel J \cong^\pi_\ell [\![P_i]\!]$.
3. *If $R = x\langle v\rangle.P \mid y(z).Q$ then*
   $[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}_\ell{}^{2} \cong^\pi_\ell \exists x,y.\big([\![P]\!] \parallel [\![Q\{v/z\}]\!] \parallel J\big)$ *with* $J = \overline{\mathtt{tt}}$.
4. *If $R = x\langle v\rangle.P \mid *\,y(z).Q$ then:*

$$[\![(\boldsymbol{\nu} xy)R]\!] \xrightarrow{\tau}_\ell{}^{2} \cong^\pi_\ell \exists x,y.\big([\![P]\!] \parallel [\![Q\{v/z\}]\!] \parallel [\![*\,y(z).P]\!] \parallel \overline{\mathtt{tt}}\big)$$

Proof (see Page 28) Each item follows from the definition of $[\![\cdot]\!]$ (cf. Def. 25 and Fig. 8). Items (1) and (2) refer to reductions that induce junk (no junk is generated in Items (3) and (4)); those cases rely on the definition of weak o-barbed congruence (cf. Def. 23) and Cor. 2.

1. Given $R = x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$ (with $j \in I$), by Def. 25, Fig. 8, and the operational semantics of `lcc` in Def. 6:

$$\begin{aligned}
[\![(\boldsymbol{\nu} xy)R]\!] &= \exists x,y.\big(!\overline{\{x{:}y\}} \parallel \overline{\mathsf{sel}(x,l_j)} \parallel \forall z\big(\mathsf{bra}(z,l_j) \otimes \{x{:}z\} \to [\![P]\!]\big) \\
&\qquad \forall l,w\big(\mathsf{sel}(w,l) \otimes \{w{:}x\} \to \overline{\mathsf{bra}(x,l)} \parallel \prod_{i\in I} \forall\epsilon(l = l_i \to [\![Q_i]\!])\big)\big) \\
&\xrightarrow{\tau}_\ell \exists x,y.\big(!\overline{\{x{:}y\}} \parallel \forall z\big(\mathsf{bra}(z,l_j) \otimes \{x{:}z\} \to [\![P]\!]\big) \\
&\qquad \overline{\mathsf{bra}(x,l_j)} \parallel \prod_{i\in I} \forall\epsilon(l_j = l_i \to [\![Q_i]\!])\big) \\
&\xrightarrow{\tau}_\ell \exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel \prod_{i\in I} \forall\epsilon(l_j = l_i \to [\![Q_i]\!])\big) \\
&\xrightarrow{\tau}_\ell \exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![P_j]\!] \parallel \underbrace{\prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to [\![Q_i]\!])}_{J}\big) \\
&\cong^\pi_\ell \exists x,y.\big(!\overline{\{x{:}y\}} \parallel [\![P]\!] \parallel [\![P_j]\!]\big)
\end{aligned}$$

   where the last step is justified by Cor. 2.

2. Given that $R = b?\,P_1 : P_2$, we distinguish two cases: $b = \mathtt{tt}$ and $b = \mathtt{ff}$. We only detail the analysis when $R = \mathtt{tt}?\,P_1 : P_2$, as the other case is analogous. By the translation definition (cf. Def. 25 and Fig. 8), $[\![R]\!] = \forall\epsilon(\mathtt{tt} = \mathtt{tt} \to [\![P_1]\!]) \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P_2]\!])$. Then, by the rules in Fig. 6, $[\![R]\!] \xrightarrow{\tau}_\ell [\![P_1]\!] \parallel J$, with $J = \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P_2]\!])$. By Cor. 2, we may conclude as follows:

$$[\![R]\!] \xrightarrow{\tau}_\ell [\![P]\!] \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q]\!]) \cong^\pi_\ell [\![P]\!]$$

3. Given $R = x\langle v\rangle.P \mid y(z).Q$, by the translation definition (cf. Fig. 8), and the semantics in Fig. 6:

$$\begin{aligned}
[\![(\boldsymbol{\nu} xy)R]\!] &\equiv \exists x,y.\big(\overline{!\{x{:}y\} \otimes \mathsf{snd}(x,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]) \parallel \\
&\qquad \forall z,w(\mathsf{snd}(w,z) \otimes \{w{:}y\} \to \overline{\mathsf{rcv}(y,z)} \parallel [\![Q]\!])\big) \\
&\xrightarrow{\tau}_\ell \exists x,y.\big(\overline{!\{x{:}y\}} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]) \parallel \overline{\mathsf{rcv}(y,v)} \parallel [\![Q\{v,x/z,w\}]\!]\big) \\
&\equiv \exists x,y.\big(\overline{!\{x{:}y\} \otimes \mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{x{:}z\} \to [\![P]\!]) \parallel [\![Q\{v,x/z,w\}]\!]\big) \\
&\xrightarrow{\tau}_\ell \exists x,y.\big(\overline{!\{x{:}y\}} \parallel [\![P\{y/z\}]\!] \parallel [\![Q\{v,x/z,w\}]\!]\big) \\
&\cong^\pi_\ell \exists x,y.\big(\overline{!\{x{:}y\}} \parallel [\![P\{y/z\}]\!] \parallel [\![Q\{v,x/z,w\}]\!] \parallel \overline{\mathtt{tt}}\big)
\end{aligned}$$

   Let $J = \overline{\mathtt{tt}}$. Finally, we conclude by Cor. 2.
4. When $R = x\langle v\rangle.Q \mid *\,y(z).P$, the proof follows the same reasoning as above. □

C.2 Operational Completeness

Theorem 11 (Completeness for $[\![\cdot]\!]$) *Let $[\![\cdot]\!]$ be the translation in Def. 25. Also, let P be a well-typed π program. Then, if $P \longrightarrow^* Q$ then $[\![P]\!] \overset{\tau}{\Longrightarrow}\cong^\pi_\ell [\![Q]\!]$.*

Proof (see Page 25) By induction on the length of the reduction $\longrightarrow^*$, with a case analysis on the last applied rule. The base case is when $P \longrightarrow^0 P$: it is trivially true since $[\![P]\!] \overset{\tau}{\Longrightarrow} [\![P]\!]$. For the inductive step, assume by IH that $P \longrightarrow^* P_0 \longrightarrow Q$ and $[\![P]\!] \overset{\tau}{\Longrightarrow}\cong^\pi_\ell [\![P_0]\!]$. We then have to prove that $[\![P_0]\!] \overset{\tau}{\Longrightarrow}\cong^\pi_\ell [\![Q]\!]$. There are nine cases, since cases for Rules (RES), (PAR) and (STR) are immediate by IH.

Rule $\lfloor$IFT$\rfloor$:
(i) $P_0 = \mathtt{tt}?\,P' : P''$.
(ii) By (i), $P_0 \longrightarrow P' = Q$.
(iii) By Def. 25, $[\![P_0]\!] = \forall\epsilon(\mathtt{tt} = \mathtt{tt} \to [\![P']\!]) \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P'']\!])$.
(iv) By Rule (C:SYNC) (cf. Fig. 6), with $c = \mathtt{tt}$ we have the following (note that $\Vdash \mathtt{tt} = \mathtt{tt}$):

$$[\![P_0]\!] \xrightarrow{\tau}_\ell [\![P']\!] \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P'']\!]) = R$$

(v) By (iv) note that the process $\forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P'']\!])$ is junk (cf. Def. 28). Then, by Cor. 2 $R \cong^\pi_\ell [\![Q]\!]$, which is what we wanted to prove.

Rule $\lfloor$IFF$\rfloor$: Analogous to the previous case.

Rule $\lfloor$COM$\rfloor$:
(i) $P_0 = (\boldsymbol{\nu} xy)(x\langle v\rangle.P' \mid y(z).P'' \mid T)$, with T corresponding to a parallel composition of processes that may contain y. Notice that by typing, T can only contain (replicated) input processes on y and not any processes containing x.
(ii) By (i) $P_0 \longrightarrow (\boldsymbol{\nu} xy)(P' \mid P''\{v/z\} \mid T) = Q$.
(iii) By Fig. 8:

$$\begin{aligned}
[\![P_0]\!] = \exists x,y.\big(\overline{!\{x{:}y\}} \parallel \overline{(\mathsf{snd}(x,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
\forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!]) \parallel [\![T]\!]\big)
\end{aligned}$$

(iv) By Fig. 6, Def. 13:

$$\begin{aligned}
[\![P_0]\!] &\equiv \exists x,y.\,((\overline{!\{x{:}y\} \otimes \mathsf{snd}(x,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
&\qquad \forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!]) \parallel [\![T]\!]) \\
&\longrightarrow_\ell \exists x,y.\,((\overline{!\{x{:}y\}} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \overline{\mathsf{rcv}(y,v)} \parallel \\
&\qquad [\![P''\{v,x/z_2,w\}]\!] \parallel [\![T]\!]) \\
&\equiv \exists x,y.(\overline{!\{x{:}y\} \otimes \mathsf{rcv}(y,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\} \to [\![P']\!]) \parallel \\
&\qquad [\![P''\{v,x/z_2,w\}]\!] \parallel [\![T]\!]) \\
&\longrightarrow_\ell \exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P'\{y/z_1\}]\!] \parallel [\![P''\{v,x/z_2,w\}]\!] \parallel [\![T]\!])
\end{aligned}$$

(v) By Fig. 8 we have that $w \notin \mathsf{fv}_\pi(P'')$ and $z_1 \notin \mathsf{fv}_\pi(P')$. Therefore:

$$\begin{aligned}
&\exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P'\{y/z\}]\!] \parallel [\![P''\{v,x/z,w\}]\!] \parallel [\![T]\!]) \\
=&\exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P''\{v/z\}]\!] \parallel [\![T]\!]) = [\![Q]\!]
\end{aligned}$$

(vi) Finally, by reflexivity of $\cong_\ell^\pi$ (Def. 23), $[\![Q]\!] \cong_\ell^\pi [\![Q]\!]$.

Rule $\lfloor$REPL$\rfloor$:

(i) Assume $P_0 = (\boldsymbol{\nu} xy)(x\langle v\rangle.P' \mid *\,y(z).P'' \mid T)$, with T collecting all the processes that may contain x and y. Notice that by typing, T can only contain (replicated) input processes on y.

(ii) By (i) $P_0 \longrightarrow (\boldsymbol{\nu} xy)(P' \mid P''\{v/z\} \mid *\,y(z).P'' \mid T) = Q$ using Rule $\lfloor$REP$\rfloor$.

(iii) By definition of $[\![\cdot]\!]$:

$$\begin{aligned}
[\![P_0]\!] &= \exists x,y.(\overline{!\{x{:}y\}} \parallel \overline{(\mathsf{snd}(x,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
&\qquad !\forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!]) \parallel [\![T]\!]) \\
&\equiv \exists x,y.(\overline{!\{x{:}y\}} \parallel \overline{(\mathsf{snd}(x,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
&\qquad \forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!])) \parallel \\
&\qquad !\forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!]) \parallel [\![T]\!])
\end{aligned}$$

(iv) Let $R = !\forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!]))$. By using the rules of structural congruence and reduction of `1cc` the following transitions can be shown:

$$\begin{aligned}
[\![P_0]\!] &\equiv \exists x,y.(\overline{!\{x{:}y\} \otimes \mathsf{snd}(x,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
&\qquad \forall z_2,w(\mathsf{snd}(w,z_2) \otimes \{w{:}y\}) \to (\overline{\mathsf{rcv}(y,z_2)} \parallel [\![P'']\!])) \parallel R \parallel [\![T]\!]) \\
&\xrightarrow{\tau}_\ell \exists x,y.(\overline{!\{x{:}y\}} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \overline{\mathsf{rcv}(y,v)} \parallel \\
&\qquad [\![P''\{v,x/z_2,w\}]\!]) \parallel R \parallel [\![T]\!]) \\
&\equiv \exists x,y.(\overline{!\{x{:}y\} \otimes \mathsf{rcv}(y,v)} \parallel \forall z_1((\mathsf{rcv}(z_1,v) \otimes \{x{:}z_1\}) \to [\![P']\!]) \parallel \\
&\qquad [\![P''\{v,x/z_2,w\}]\!] \parallel R \parallel [\![T]\!]) \\
&\xrightarrow{\tau}_\ell \exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P'\{y/z_1\}]\!] \parallel [\![P''\{v,x/z_2,w\}]\!] \parallel R \parallel [\![T]\!])
\end{aligned}$$

(v) As in **Case** $\lfloor$COM$\rfloor$, we have that

$$\begin{aligned}
&\exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P'\{y/z_1\}]\!] \parallel [\![P''\{v,x/z_2,w\}]\!] \parallel R \parallel [\![T]\!]) \\
&= \exists x,y.(\overline{!\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P''\{v/z_2\}]\!] \parallel R \parallel [\![T]\!])
\end{aligned}$$

since $w \notin \mathsf{fv}_\pi(P'')$ and $z_1 \notin \mathsf{fv}_\pi(P'')$, by Fig. 8.

(vi) Finally, observe that:

$$\begin{aligned}
[\![Q]\!] &= [\![(\boldsymbol{\nu} xy)(P' \mid P''\{v/z\} \mid\ *\,y(z).P'' \mid T)]\!] \\
&= \exists x,y.(!\overline{\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P''\{v/z_2\}]\!] \parallel R \parallel [\![T]\!]))
\end{aligned}$$

Rule $\lfloor$SEL$\rfloor$:

(i) Assume $P_0 = (\boldsymbol{\nu} xy)(x \triangleleft l_j.P' \mid y \triangleright \{l_i : P_i\}_{i\in I} \mid T)$. Notice that since P_0 is a well-formed program, typing implies that process $T \equiv \mathbf{0}$, since x, y cannot be shared. Thus, we do not consider T below.
(ii) By (i) $P_0 \longrightarrow (\boldsymbol{\nu} xy)(P' \mid P_j) = Q$ using Rule $\lfloor$SEL$\rfloor$.
(iii) By definition of $[\![\cdot]\!]$ (cf. Fig. 8):

$$[\![P_0]\!] = \exists x, y.\big(!\overline{\{x{:}y\}} \parallel \overline{\mathsf{sel}(x, l_j)} \parallel \forall z\big(\mathsf{bra}(z, l_j) \otimes \{x{:}z\} \to [\![P']\!]\big) \\ \forall l, w\big(\mathsf{sel}(w, l) \otimes \{w{:}x\} \to \overline{\mathsf{bra}(x, l)} \parallel \prod_{i\in I} \forall \epsilon(l = l_i \to [\![P_i]\!])\big)\big)$$

(iv) By using the semantics of `lcc` (cf. Def. 6) and Cor. 2, we obtain the following derivation

$$\begin{aligned}
[\![P_0]\!] &\xrightarrow{\tau}_\ell \exists x, y.\big(!\overline{\{x{:}y\}} \parallel \forall z\big(\mathsf{bra}(z, l_j) \otimes \{x{:}z\} \to [\![P']\!]\big) \\
&\qquad \overline{\mathsf{bra}(x, l_j)} \parallel \prod_{i\in I} \forall \epsilon(l_j = l_i \to [\![P_i]\!])\big) \\
&\xrightarrow{\tau}_\ell \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P']\!] \parallel \prod_{i\in I} \forall \epsilon(l_j = l_i \to [\![P_i]\!])\big) \\
&\xrightarrow{\tau}_\ell \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P_j]\!] \parallel \underbrace{\prod_{i\in I\setminus\{j\}} \forall \epsilon(l_j = l_i \to [\![P_i]\!])}_{J}\big) \\
&\cong^{\pi}_{\ell} \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P_j]\!]\big)
\end{aligned}$$

(v) By definition of $[\![\cdot]\!]$ (cf. Fig. 8), $[\![Q]\!] = [\![(\boldsymbol{\nu} xy)(P' \mid P_j)]\!] = \exists x, y.\big(!\overline{\{x{:}y\}} \parallel [\![P']\!] \parallel [\![P_j]\!]\big)$.
(vi) By (iv) and (v) we conclude the proof. □

C.3 Invariants for Pre-Redexes and Redexes

Lemma 6 (Invariants of $[\![\cdot]\!]$ for Pre-Redexes and the Inaction) *Let P be a pre-redex or the inactive process in π. Then the following properties hold:*

1. *If $\mathcal{I}^{\mathcal{D}^\star_\pi}([\![P]\!]) = \{\mathsf{snd}(x, v)\}$ then $P = x\langle v\rangle.P_1$, for some P_1.*
2. *If $\mathcal{I}^{\mathcal{D}^\star_\pi}([\![P]\!]) = \{\mathsf{sel}(x, l)\}$ then $P = x \triangleleft l.P_1$, for some P_1.*
3. *If $\mathcal{I}^{\mathcal{D}^\star_\pi}([\![P]\!]) = \{\mathtt{tt}\}$ then $P = \mathbf{0}$.*
4. *If $\mathcal{I}^{\mathcal{D}^\star_\pi}([\![P]\!]) = \emptyset$ then $P = \diamond\, y(z).P_1$ (cf. Not. 2) or $P = x \triangleright \{l_1 : P_i\}_{i\in I}$, for some P_i. Moreover, $[\![P]\!] \not\xrightarrow{\tau}_\ell$.*

Proof (see Page 29) By assumption $P = \mathbf{0}$ or P is a pre-redex (Def. 9): $P = x\langle v\rangle.P_1$, $P = x \triangleleft l.P_1$, $P = y(z).P_1$, $P = *y(z).P_1$ or $P = x \triangleright \{l_1 : P_i\}_{i\in I}$. Given these six possible forms for P, we then check the immediate observables (cf. Def. 29) of their `lcc` translations (cf. Fig. 8):

$$\begin{aligned}
\mathcal{I}^{\mathcal{D}^\star_\pi}([\![x\langle v\rangle.P_1]\!]) &= \{\mathsf{snd}(x, v)\} & \mathcal{I}^{\mathcal{D}^\star_\pi}([\![x(y).P_1]\!]) &= \emptyset \\
\mathcal{I}^{\mathcal{D}^\star_\pi}([\![x \triangleleft l.P_1]\!]) &= \{\mathsf{sel}(x, l)\} & \mathcal{I}^{\mathcal{D}^\star_\pi}([\![* x(y).P_1]\!]) &= \emptyset \\
\mathcal{I}^{\mathcal{D}^\star_\pi}([\![\mathbf{0}]\!]) &= \{\mathtt{tt}\} & \mathcal{I}^{\mathcal{D}^\star_\pi}([\![x \triangleright \{l_1 : P_i\}_{i\in I}]\!]) &= \emptyset
\end{aligned}$$

This way, the thesis holds. □

Lemma 7 (Invariants of $[\![\cdot]\!]$ for Input-Like Pre-Redexes) *Let P be a pre-redex such that $\mathcal{I}^{\mathcal{D}^\star_\pi}([\![P]\!]) = \emptyset$. Then one of the following holds:*

1. *If $[\![P]\!] \parallel \overline{\mathsf{sel}(x, l_j) \otimes \{y{:}x\}} \xrightarrow{\tau}_\ell S$ then $\mathsf{bra}(y, l_j) \in \mathcal{I}^{\mathcal{D}^\star_\pi}(S)$ and $P = y \triangleright \{l_i : P_i\}_{i\in I}$, with $j \in I$.*
2. *If $[\![P]\!] \parallel \overline{\mathsf{snd}(x, v) \otimes \{y{:}x\}} \xrightarrow{\tau}_\ell S$ then $\mathsf{rcv}(y, v) \in \mathcal{I}^{\mathcal{D}^\star_\pi}(S)$ and $P = \diamond\, y(z).P_1$.*

Proof (see Page 29) By assumption, P is a pre-redex and $\mathcal{I}^{\mathcal{D}^\star_\pi}(P) = \emptyset$. By Lem. 6(4), we have that $[\![P]\!] \not\xrightarrow{\tau}_\ell$ and that $P = y(z).P_1$, $P = *\,y(z).P_1$ or $P = y \triangleright \{l_1 : P_i\}_{i\in I}$. We now apply a case analysis on each numeral in the statement:

Case $[\![P]\!] \parallel \overline{\mathsf{sel}(w,l) \otimes \{y{:}x\}} \xrightarrow{\tau}_\ell S$**:** We observe the behavior of each possibilities for P in the presence of constraint $\mathsf{sel}(w,l_j) \otimes \{y{:}x\}$ for some l_j, following Fig. 8. First we observe:

$$[\![y(z).P_1]\!] \parallel \overline{\mathsf{sel}(w,l) \otimes \{y{:}x\}} = \forall z, w\big(\mathsf{snd}(w,z) \otimes \{w{:}y\} \to \overline{\mathsf{rcv}(y,z)} \parallel [\![P_1]\!]\big) \parallel \overline{\mathsf{sel}(x,l_j) \otimes \{y{:}x\}} \not\xrightarrow{\ell}$$

$$[\![* \, y(z).P_1]\!] \parallel \overline{\mathsf{sel}(w,l) \otimes \{y{:}x\}} = !(\forall z, w\big(\mathsf{snd}(w,z) \otimes \{w{:}y\} \to \overline{\mathsf{rcv}(y,z)} \parallel [\![P_1]\!]\big)) \parallel \overline{\mathsf{sel}(x,l_j) \otimes \{y{:}x\}} \not\xrightarrow{\ell}$$

In contrast, process $[\![y \triangleright \{l_1 : P_i\}_{i\in I}]\!] \parallel \overline{\mathsf{sel}(w,l) \otimes \{y{:}x\}}$ can reduce: from the semantics of `lcc` (cf. Fig. 6) and under the assumption that $j \in I$ for l_j, we have:

$$\forall l, w\big(\mathsf{sel}(w,l) \otimes \{w{:}y\} \to \overline{\mathsf{bra}(y,l)} \parallel \prod_{1\le i\le n} \forall\epsilon(l = l_i \to [\![P_i]\!])\big) \parallel \overline{\mathsf{sel}(x,l_j) \otimes \{y{:}x\}} \xrightarrow{\tau}_\ell \overline{\mathsf{bra}(y,l_j)} \parallel \prod_{1\le i\le n} \forall\epsilon(l = l_i \to [\![P_i]\!]) = S$$

Finally, by Def. 29: $\mathcal{I}^{\mathcal{D}^\star_\pi}(S) = \{\mathsf{bra}(y,l_j)\} \cup \mathcal{I}^{\mathcal{D}^\star_\pi}(\prod_{1\le i\le n} \forall\epsilon(l_j = l_i \to [\![P_i]\!]))$, thus concluding the proof.

Case $[\![P]\!] \parallel \overline{\mathsf{snd}(x,v) \otimes \{y{:}x\}} \xrightarrow{\tau}_\ell S$**:** This case proceeds as above by noticing that a reduction into S is enabled only when $P = y(z).P_1$ or $P = *\,y(z).P_1$. □

Lemma 8 (Invariants for Redexes and Intermediate Redexes) *Let R be a redex enabled by $\widetilde{x},\widetilde{y}$, such that $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})R'$. Then one of the following holds:*

1. *If $R \equiv_\pi v?\,P_1 : P_2$ and $v \in \{\mathtt{tt},\mathtt{ff}\}$, then $[\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R]\!] \xrightarrow{\tau}_\ell \cong^\pi_\ell (\boldsymbol{\nu}\widetilde{x}\widetilde{y})[\![P_i]\!]$, with $i \in \{1,2\}$.*
2. *If $R \equiv_\pi x\langle v\rangle.P \mid \diamond\ y(w).Q$, then $[\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R]\!] \longrightarrow_\ell \equiv C_{\widetilde{x}\widetilde{y}}[(\!|R|\!)^1_{\widetilde{x}\widetilde{y}}] \longrightarrow_\ell \cong^\pi_\ell [\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R']\!]$.*
3. *If $R \equiv_\pi x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$, with $j \in I$, then we have the reductions in Fig. 12.*

Proof (see Page 30) This proof proceeds by using the translation (cf. Fig. 8) the `lcc` semantics (cf. Fig. 6). All items are shown in the same way; we detail only Item (3), which is arguably the most interesting case:

3 By assumption, $R \equiv_\pi x \triangleleft l_j.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}$, with $j \in I$ and $(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R \longrightarrow (\boldsymbol{\nu}\widetilde{x}\widetilde{y})R'$. By Fig. 1, $(\boldsymbol{\nu}xy)(x \triangleleft l_j.P \mid y \triangleright \{l_i{:}Q_i\}_{i\in I}) \longrightarrow (\boldsymbol{\nu}xy)(P \mid Q_j)$, with $j \in I$. Finally, by Fig. 8, Fig. 6 and expanding Not. 13:

$$\begin{aligned}
[\![(\boldsymbol{\nu}\widetilde{x}\widetilde{y})R]\!] &= \exists\widetilde{x},\widetilde{y}.\big(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}} \{x_i{:}y_i\}} \parallel \overline{\mathsf{sel}(x,l_j)} \parallel \forall z\big(\mathsf{bra}(z,l_j) \otimes \{x{:}z\} \to [\![P]\!]\big) \parallel \\
&\qquad \forall l, w\big(\mathsf{sel}(w,l) \otimes \{w{:}y\} \to \overline{\mathsf{bra}(y,l)} \parallel \prod_{1\le i\le n} \forall\epsilon(l = l_i \to [\![P_i]\!])\big)\big) \\
&\xrightarrow{\tau}_\ell \exists\widetilde{x},\widetilde{y}.\big(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}} \{x_i{:}y_i\}} \parallel \forall z\big(\mathsf{bra}(z,l_j) \otimes \{x{:}z\} \to [\![P]\!]\big) \\
&\qquad\qquad \parallel \overline{\mathsf{bra}(y,l_j)} \parallel \prod_{1\le i\le n} \forall\epsilon(l_j = l_i \to [\![P_i]\!])\big) \\
&\equiv \exists\widetilde{x},\widetilde{y}.\big(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}} \{x_i{:}y_i\}} \parallel \forall z\big(\mathsf{bra}(z,l_j) \otimes \{x{:}z\} \to [\![P]\!]\big) \parallel \\
&\qquad \overline{\mathsf{bra}(y,l_j)} \parallel \forall\epsilon(l_j = l_j \to [\![P_j]\!]) \parallel \prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to [\![P_i]\!])\big) \\
&\equiv \exists\widetilde{x},\widetilde{y}.\big(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}} \{x_i{:}y_i\}} \parallel (\!|x \triangleleft l_j.P \mid y \triangleright \{l_i{:}Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}}\big) = T
\end{aligned}$$

Up to this point, we have shown that $[\![(\nu\widetilde{x}\widetilde{y})R]\!] \xrightarrow{\tau}_{\ell} \equiv \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel (\!|R|\!)^1_{\widetilde{x}\widetilde{y}})$. We now distinguish cases for the next reduction, as there are two possibilities:

(a) From Fig. 6 and Cor. 2:

$$\begin{aligned}
T &\xrightarrow{\tau}_{\ell} \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel [\![P]\!] \parallel \forall\epsilon(l_j = l_j \to [\![P_j]\!]) \parallel \prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to [\![P_i]\!]))\\
&\equiv \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel (\!|x \triangleleft l_j.P \mid y \triangleright \{l_i{:}Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}})\\
&\xrightarrow{\tau}_{\ell\cong^{\pi}_{\ell}} \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel [\![P]\!] \parallel [\![P_j]\!])
\end{aligned}$$

(b) From Fig. 6 and Cor. 2:

$$\begin{aligned}
T &\xrightarrow{\tau}_{\ell} \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel \forall z\big(\mathsf{bra}(z,l_j)\otimes\{x{:}z\} \to [\![P]\!]\big) \parallel \overline{\mathsf{bra}(y,l_j)} \parallel [\![P_j]\!] \parallel\\
&\qquad \prod_{i\in I\setminus\{j\}} \forall\epsilon(l_j = l_i \to [\![P_i]\!]))\\
&\equiv \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel (\!|x \triangleleft l_j.P \mid y \triangleright \{l_i{:}Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}})\\
&\xrightarrow{\tau}_{\ell\cong^{\pi}_{\ell}} \exists\widetilde{x},\widetilde{y}.(!\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}}\{x_i{:}y_i\}} \parallel [\![P]\!] \parallel [\![P_j]\!])
\end{aligned}$$

□

C.4 Invariants for Well-Typed Translated Programs

Lemma 9 *Let P be a well-typed program. If $[\![P]\!] \xrightarrow{\tau}\!\!\!\!\rightarrow S$ then*

$$S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n \parallel J]$$

where $n \geq 1$, J is some junk, and for all $i \in \{1,\ldots,n\}$ we have $U_i = \overline{\mathtt{tt}}$ or one the following:

1. *$U_i = [\![R_k]\!]$, where R_k is a conditional redex (cf. Def. 9) reachable from P;*
2. *$U_i = [\![R_k]\!]$, where R_k is a pre-redex reachable from P;*
3. *$U_i \in \{[\![R_k \mid R_j]\!]\}$ (cf. Def. 30), where redex $R_k \mid R_j$ is reachable from P.*

Proof (see Page 31) By induction on the length k of the reduction $\xrightarrow{\tau}\!\!\!\!\rightarrow$. The base case ($k = 0$) is immediate: since $[\![P]\!] \xrightarrow{\tau}\!\!\!\!\rightarrow [\![P]\!]$, by Lem. 1 we have $S = [\![P]\!] = C_{\widetilde{x}\widetilde{y}}[[\![R_1]\!] \parallel \cdots \parallel [\![R_n]\!]]$, and the property holds because every $[\![R_i]\!]$ is captured by Cases (1) and (2).

The inductive step ($k > 0$) proceeds by a case analysis of the transition $S_0 \xrightarrow{\tau}_{\ell} S$. We state the IH:

IH1: If $[\![P]\!] \xrightarrow{\tau}\!\!\!\!\rightarrow S_0 \xrightarrow{\tau}_{\ell} S$, then $S_0 = C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel W_m \parallel J_0]$ where $m \geq 1$, for some junk J_0, and every W_i is either $\overline{\mathtt{tt}}$ or satisfies one of the three cases.

The transition $S_0 \xrightarrow{\tau}_{\ell} S$ can only originate in some $W_i \neq \overline{\mathtt{tt}}$. There are then three cases to consider: W_i is a conditional redex, a pre-redex, or an intermediate process. We have:

Case $W_i = [\![b?\, P_1 : P_2]\!]$ with $b \in \{\mathtt{tt},\mathtt{ff}\}$: There are two sub-cases, depending on whether $b = \mathtt{tt}$ or $b = \mathtt{ff}$. We only detail the case $b = \mathtt{tt}$, as the case $b = \mathtt{ff}$ proceeds similarly. We have:

(1) $W_i = \forall\epsilon(\mathtt{tt} = \mathtt{tt} \to [\![P_1]\!]) \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P_2]\!])$ (Fig. 8).
(2) $\exists P'.(P \longrightarrow^* P' = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(\mathtt{tt}?\, P_1 : P_2 \mid Q))$ (IH1).
(3) $P' \longrightarrow P'' = (\boldsymbol{\nu}\widetilde{x}\widetilde{y})(P_1 \mid Q)$ (Fig. 1, (2)).
(4) $S_0 \xrightarrow{\tau}_\ell S = C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![P_1]\!] \cdots \parallel W_m \parallel J]$, with $J = \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![P_2]\!]) \parallel J_0$ (Fig. 6, (1)).

To conclude this case, we proceed by induction on the structure of P_1:

Case $P_1 = \mathbf{0}$: By (4) and Fig. 8, $S = C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel \overline{\mathtt{tt}} \parallel \cdots \parallel W_m \parallel J]$, and so the thesis follows.

Case $P_1 = b?\, Q_1 : Q_2$: By (4) and Fig. 8, $S = C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![P_1]\!] \parallel \cdots \parallel W_m \parallel J]$. Hence, the thesis follows under Case (1).

Cases $P_1 = x\langle v\rangle.P$, $P_1 = x(y).Q$, $P_1 = x \triangleleft l_j.Q$, $P_1 = *\,x(y).Q$, and $P_1 = x \triangleright \{l_i : Q_i\}_{i\in I}$: From the rules in Fig. 8 and (4), $S = C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![P_1]\!] \parallel \cdots \parallel W_m \parallel J]$. Hence, the thesis follows under Case (2).

Case $P_1 = Q_1 \mid Q_2$: By IH, the thesis holds for $[\![Q_1]\!]$ and $[\![Q_2]\!]$, and the reduction from S_0 to S generates one additional parallel process inside $C_{\widetilde{x}\widetilde{y}}[\cdot]$.

Case $P_1 = (\boldsymbol{\nu}xy)Q$: By IH, the thesis holds for $[\![Q]\!]$. By noticing that

$$C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![(\boldsymbol{\nu}xy)Q]\!] \parallel \cdots \parallel W_m \parallel J] = C_{\widetilde{x}x\widetilde{y}y}[W_1 \parallel \cdots \parallel [\![Q]\!] \parallel \cdots \parallel W_m \parallel J]$$

the thesis follows.

Case $W_i = [\![R_k]\!]$, for some pre-redex R_j: Then the transition from S_0 to S can only occur if there exists a $W_j = [\![R_j]\!]$, such that $R_k \mid R_j$ is a redex reachable from P. There are multiple sub-cases, depending on the shape of R_k and R_j. We only detail a representative sub-case; the rest are similar:

Sub-case $R_k = x\langle v\rangle.P$: We then have that $R_k = y(z).Q$ and so

$$\begin{aligned} S_0 &= C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel W_i \parallel \cdots \parallel W_j \parallel \cdots \parallel W_m] \\ &= C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![x\langle v\rangle.P]\!] \parallel \cdots \parallel [\![y(z).Q]\!] \parallel \cdots \parallel W_m] \\ &\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel (\!|x\langle v\rangle.P \mid y(z).Q|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel W_m] = S \end{aligned}$$

where the transition to S follows Lem. 8(2). The thesis then follows Case (3).

Case $W_i \in \{[R_k \mid R_j]\}$, for some redex $R_k \mid R_j$: Then, depending on the shape of R_k and R_j (and relying on Not. 14), the transition from S_0 to S corresponds to one of the following five sub-cases:

(a) $W_i = (\!|x\langle v\rangle.P \mid y(z).Q|\!)^1_{\widetilde{x}\widetilde{y}}$
(b) $W_i = (\!|x\langle v\rangle.P \mid *\,y(z).Q|\!)^1_{\widetilde{x}\widetilde{y}}$
(c) $W_i = (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}}$
(d) $W_i = (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}}$
(e) $W_i = (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}}$

We only detail Sub-cases (a), (c) and (e); the rest are similar:

Sub-case $W_i = (\!|x\langle v\rangle.P \mid y(z).Q|\!)^1_{\widetilde{x}\widetilde{y}}$: Then we have:

$$\begin{aligned} S_0 = \; & C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel (\!|x\langle v\rangle.P \mid y(z).Q|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel W_m] \\ & \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![P]\!] \parallel [\![Q]\!]\{v/z\} \parallel \cdots \parallel W_m] = S \end{aligned}$$

and the proof proceeds by a simultaneous induction on the structure of both P and Q, as shown for the case of the conditional redex above.

Sub-case $W_i = (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}}$: Then we have:

$$\begin{aligned} S_0 = \; & C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel W_m] \\ & \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^k_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel W_m] = S \end{aligned}$$

Sub-case $W_i = (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}}$: Assuming $l = l_j$ for some $j \in I$, then we have:

$$\begin{aligned} S_0 = \; & C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel (\!|x \triangleleft l.P \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel W_m] \\ & \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[W_1 \parallel \cdots \parallel [\![P]\!] \parallel [\![Q_j]\!] \parallel \cdots \parallel W_m] = S \end{aligned}$$

and the proof proceeds by a simultaneous induction on the structure of both P and Q, as shown for the case of the conditional redex above. □

Lemma 10 *Let P be a well-typed π program. Then, for every S, S' such that $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \stackrel{\tau}{\longrightarrow}_\ell S'$ one of the following holds:*

(a) $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$ (cf. Not. 15) and one of the following holds:
  (1) $S \equiv C_{\widetilde{xy}}[[\![b?\, P_1 : P_2]\!] \parallel U]$ and $S' = C_{\widetilde{xy}}[[\![P_i]\!] \parallel U]$, with $i \in \{1,2\}$;
  (2) $S \equiv C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{xy}} \parallel U]$ and $S' = C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{xy}} \parallel U]$;
  (3) $S \equiv C_{\widetilde{xy}}[(\!|y \triangleleft l_j.P' \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{xy}} \parallel U]$ and $S' = C_{\widetilde{xy}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U]$.
(b) $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$ and $|\mathcal{I}_S \setminus \mathcal{I}_{S'}| = 1$.

Proof (see Page 32) We first use Lem. 9 to characterize every parallel sub-process U_i of S; then, by a case analysis on the shape of the U_i that originated the transition $S \stackrel{\tau}{\longrightarrow}_\ell S'$ we show that each case falls under either (a) or (b). More in details, by Lem. 9 we have:

$$S = C_{\widetilde{xy}}[U_1 \parallel \cdots \parallel U_n]$$

where for every U_i either $U_i = \overline{\mathtt{tt}}$ or

(i) $U_i = [\![R_k]\!]$, where R_k is a conditional redex reachable from P;
(ii) $U_i = [\![R_k]\!]$, where R_k is a pre-redex reachable from P;
(iii) $U_i \in \{[\![R_k \mid R_j]\!]\}$, where redex $R_k \mid R_j$ is reachable from P.

Hence, transition $S \stackrel{\tau}{\longrightarrow}_\ell S'$ must originate from some U_i. There are 12 different possibilities for this transition:

A. $U_i = [\![\mathtt{tt}?\, Q_1 : Q_2]\!]$;
B. $U_i = [\![\mathtt{ff}?\, Q_1 : Q_2]\!]$;
C. $U_i = [\![x\langle v\rangle.Q_1]\!]$;
D. $U_i = [\![x(z).Q_1]\!]$;
E. $U_i = [\![x \triangleleft l.Q_1]\!]$;
F. $U_i = [\![x \triangleright \{l_i : Q_i\}_{i\in I}]\!]$;
G. $U_i = [\![*\, x(z).Q_1]\!]$.
H. $U_i = (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{\widetilde{xy}}$;
I. $U_i = (\!|x\langle v\rangle.Q_1 \mid *\, y(z).Q_2|\!)^1_{\widetilde{xy}}$;
J. $U_i = (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{xy}}$;
K. $U_i = (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{xy}}$;
L. $U_i = (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{xy}}$;

Notice that in Sub-cases A-B and H-L, the U_i can transition by itself; in Sub-cases C-G, the U_i needs to interact with some other U_j (with $i \neq j$) to produce the transition. Also, notice that in Sub-case J, two more sub-cases are generated, which depend on the transition induced by $U_i = (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{xy}}$:

J(1). $S \stackrel{\tau}{\longrightarrow}_\ell S' = C_{\widetilde{xy}}[U_1 \parallel \cdots \parallel (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{xy}} \parallel \cdots \parallel U_n]$
J(2). $S \stackrel{\tau}{\longrightarrow}_\ell S' = C_{\widetilde{xy}}[U_1 \parallel \cdots \parallel (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{xy}} \parallel \cdots \parallel U_n]$

These two additional sub-cases are distinguished according to Lem. 8(3). All sub-cases are proven in the same way: first, identify the exact shape of S involved, and use the appropriate rule(s) in Fig. 6 to obtain S'. Next, compare the stores $\mathcal{I}_S$ and $\mathcal{I}_{S'}$. If $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$, then the sub-case falls under (a). Otherwise, the sub-case falls under (b). We detail two representative sub-cases:

Sub-Case A: Since the U_i that originates the transition is a conditional redex then $S = C_{\widetilde{x}y}[U_1 \parallel \cdots \parallel [\![\mathtt{tt}?\, Q_1 : Q_2]\!] \parallel \cdots \parallel U_n]$. By Fig. 6, and eliminating the junk with Cor. 2, we have:

$$S \stackrel{\tau}{\longrightarrow}_\ell \cong^\pi_\ell S' = C_{\widetilde{x}y}[U_1 \parallel \cdots \parallel [\![Q_1]\!] \parallel \cdots \parallel U_n]$$

Then, we are left to prove that $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$. This follows straightforwardly by considering that:

$$\forall e \in \mathcal{I}_S.(e \in \mathcal{I}_{S'})$$

because the transition of a conditional redex does not consume any constraint and that:

$$\forall e \in \mathcal{I}_{[\![Q_1]\!]}.(e \in \mathcal{I}_{S'})$$

because new constraints are added by $[\![Q_1]\!]$. Hence, this sub-case falls under (a).

Sub-Case H: Notice that well-typedness, via Lem. 1, ensures that there will never be two processes in parallel prefixed with the same variable, unless they are input processes. Furthermore, it is not possible for more than a single input process to interact with its corresponding partner, ensuring the uniqueness of the constraint. Using this, we can detail the case:

Sub-Case	S'	(a)	(b)
A	$C_{\widetilde{x}y}[U_1 \parallel \cdots \parallel [\![P_1]\!] \parallel \cdots \parallel U_n]$	✓	
B	$C_{\widetilde{x}y}[U_1 \parallel \cdots \parallel [\![P_1]\!] \parallel \cdots \parallel U_n]$	✓	
C	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x\langle v\rangle.Q_1 \mid y(z).Q_2\|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
D	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x(z).Q_1 \mid y\langle v\rangle.Q_2\|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
E	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}\|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
F	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}\|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
G	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|y\langle v\rangle.Q_1 \mid *\, x(z).Q_2\|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
H	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![Q_1]\!] \parallel [\![Q_2]\!]\{v/z\} \parallel \cdots \parallel U_n]$		✓
I	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![Q_1]\!] \parallel [\![Q_2]\!]\{v/z\} \parallel [\![*\, y(z).Q_2]\!] \parallel \cdots \parallel U_n]$		✓
J(1)	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}\|\!)^2_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$		✓
J(2)	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!\|x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}\|\!)^3_{\widetilde{x}\widetilde{y}} \parallel \cdots \parallel U_n]$	✓	
K	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![Q]\!] \parallel [\![Q_j]\!] \parallel \cdots \parallel U_n]$		✓
L	$C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![Q]\!] \parallel [\![Q_j]\!] \parallel \cdots \parallel U_n]$	✓	

Table 1: Proof of Lem. 10: Case analysis. Recall that $S = C_{\widetilde{x}y}[U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n]$.

1. $U_i = (\!|x\langle .\rangle R' \mid y(z).R''|\!)^1_{\widetilde{x}\widetilde{y}} = \overline{\mathsf{rcv}(y,v)} \parallel \forall z(\mathsf{rcv}(z,v) \otimes \{z{:}x\} \multimap [\![R']\!]) \parallel [\![R''\{v/x\}]\!]$ (Not. 14).
2. $\mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n]) = \{\exists \widetilde{x}, \widetilde{y}.\mathsf{rcv}(y,v)\} \cup \mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n])$ (Def. 29,(1)).
3. $S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![R']\!] \parallel [\![R''\{v/x\}]\!] \parallel \cdots \parallel U_n]$ (Fig. 6 - Rule (C:Sync), (1)).
4. $\mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![R']\!] \parallel [\![R''\{v/x\}]\!] \parallel \cdots \parallel U_n]) = \{\exists \widetilde{x}\widetilde{y}.c \mid c \in \mathcal{I}^{\mathcal{D}^\star_\pi}([\![R']\!] \parallel [\![R''\{v/x\}]\!])\} \cup \mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n])$ (Def. 29, (3)).
5. $\mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n]) \setminus \mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![R']\!] \parallel [\![R''\{v/x\}]\!] \parallel \cdots \parallel U_n]) = \{\exists \widetilde{x}, \widetilde{y}.\mathsf{rcv}(y,v)\}$ (Set difference, (2),(4)).

We can then conclude by observing that:
$\mathcal{I}_S = \mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_i \parallel \cdots \parallel U_n])$;
$\mathcal{I}_{S'} = \mathcal{I}^{\mathcal{D}^\star_\pi}(C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![R']\!] \parallel [\![R''\{v/x\}]\!] \parallel \cdots \parallel U_n])$
and considering that:
$|\mathcal{I}_S \setminus \mathcal{I}_{S'}| = |\{\exists \widetilde{x}, \widetilde{y}.\mathsf{rcv}(y,v)\}| = 1$. Hence, this sub-case falls under (b).

Table 1 summarizes the results for all sub-cases. □

Lemma 11 (Invariants of Target Terms (I): Adding Information) *Let P be a well-typed π program. For any S, S' such that $[\![P]\!] \xRightarrow{\tau} S \xrightarrow{\tau}_\ell S'$ and $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$ (cf. Not. 15) one of the following holds, for some U:*

1. *$S \equiv C_{\widetilde{z}}[[\![b?\, P_1 : P_2]\!] \parallel U \parallel J_1]$ and $S' = C_{\widetilde{z}}[[\![P_i]\!] \parallel \forall \epsilon(b = \neg b \multimap P_j) \parallel U \parallel J_1]$ with $i, j \in \{1,2\}, i \neq j$;*
2. *$[\![P]\!] \xRightarrow{\tau} S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![x \triangleleft l_j.P' \mid y \triangleright \{l_i\ Q_i\}_{i\in I}]\!] \parallel U \parallel J_1]$ and either:*
   - *(a) All of the following hold:*
     - *(i) $S_0 \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1] \xrightarrow{\tau}_\ell S$,*
     - *(ii) $S = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$ (and)*
     - *(iii) $S' = C_{\widetilde{x}\widetilde{y}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U \parallel J_1 \parallel J_2]$.*
   - *(b) All of the following hold:*
     - *(i) $S_0 \xrightarrow{\tau}_\ell S = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$,*
     - *(ii) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x \triangleleft l_j.P' \mid y \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel U \parallel J_1]$ (and)*
     - *(iii) $S' \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P']\!] \parallel [\![Q_j]\!] \parallel U \parallel J_1 \parallel J_2]$.*

   where $J_2 = \prod_{k\in I\setminus\{j\}} \forall \epsilon(l_j = l_k \multimap [\![P_k]\!])$.

Proof *(see Page 32)* By induction on the length of the transition $\xRightarrow{\tau} \xrightarrow{\tau}_\ell$. First, by Lem. 1:

$$[\![P]\!] \equiv C_{\widetilde{x}\widetilde{y}}[[\![R_1]\!] \parallel \cdots \parallel [\![R_n]\!]] \tag{1}$$

where every R_i is either a pre-redex or a conditional process. We apply induction on the length of transition $\stackrel{\tau}{\Longrightarrow}$:

Base Case: We analyze whenever $[\![P]\!] \stackrel{\tau}{\Longrightarrow} [\![P]\!] \stackrel{\tau}{\longrightarrow}_\ell S'$. Thus, let $S = [\![P]\!]$. Since $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$, then by Lem. 10(a), we have:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![R_j]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus j} [\![R_i]\!]]$$
$$S' \equiv C_{\widetilde{x}\widetilde{y}}[S_j \parallel \prod_{i\in\{1\ldots n\}\setminus\{j\}} [\![R_i]\!]]$$

where $[\![R_j]\!] = [\![b?\, Q_1 : Q_2]\!]$. Notice that only Item (1) of the statement is possible: Item (2) requires S to contain intermediate redexes, which is not possible since S is the translation of a process without any preceding transition. By assumption, P is a well-typed program, therefore, by Def. 10, $b \in \{\mathtt{tt}, \mathtt{ff}\}$. We distinguish cases for each $b = \mathtt{tt}$ and $b = \mathtt{ff}$. We only show the case $b = \mathtt{tt}$, as the other is similar.

Case $b = \mathtt{tt}$**:** By Fig. 8:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[\forall\epsilon(\mathtt{tt} = \mathtt{tt} \to [\![Q_1]\!]) \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q_2]\!]) \parallel \prod_{i\in\{1\ldots n\}\setminus\{j\}} [\![R_i]\!]]$$

By applying the rules in Fig. 6:

$$S \stackrel{\tau}{\longrightarrow}_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q_1]\!] \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q_2]\!]) \parallel \prod_{i\in\{1\ldots n\}\setminus j} [\![R_i]\!]] \equiv S'$$

By Def. 28, let $J = \overline{\mathtt{tt}}$ and $J' = J \parallel \forall\epsilon(\mathtt{tt} = \mathtt{ff} \to [\![Q_2]\!])$. Therefore, by Def. 13:

$$U \equiv C_{\widetilde{x}\widetilde{y}}[\prod_{i\in\{1\ldots n\}\setminus j} [\![R_i]\!] \parallel J]$$
$$U' \equiv C_{\widetilde{x}\widetilde{y}}[\parallel \prod_{i\in\{1\ldots n\}\setminus\{j\}} [\![R_i]\!] \parallel J']$$

Finally, let $[\![R_i]\!] = U_i$ for every $i \in i \in \{1 \ldots n\} \setminus \{j\}$, finishing the proof.

Inductive Step: By IH, $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S_0 \stackrel{\tau}{\longrightarrow}_\ell S$ satisfies the property for m steps (i.e., $[\![P]\!] \stackrel{\tau}{\longrightarrow}_\ell^{m-1} S_0 \stackrel{\tau}{\longrightarrow}_\ell S$). We must prove for $k = m + 1$:

$$[\![P]\!] \stackrel{\tau}{\longrightarrow}_\ell^{m} S \stackrel{\tau}{\longrightarrow}_\ell S'$$

by Lem. 9:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n \parallel J]$$

for some junk J and for all $i \in \{1, \ldots, n\}$ either:

1. $U_i = [\![R_k]\!]$, where R_k is a conditional redex reachable from P;
2. $U_i = [\![R_k]\!]$, where R_k is a pre-redex reachable from P;
3. $U_i \in \{[\![R_k \mid R_j]\!]\}$, where redex $R_k \mid R_j$ is reachable from P.

Then, by Lem. 10(a), there exists $[\![R_j]\!]$ such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[U_j \parallel \prod_{i\in\{1\ldots n\}\setminus j} U_i \parallel J]$$
$$S' \equiv C_{\widetilde{x}\widetilde{y}}[U'_j \parallel \prod_{i\in\{1\ldots n\}\setminus j} U_i \parallel J']$$

and only cases (1) and (3) aree considered:

Case $U_j = [\![R_j]\!]$ with R_j a conditional redex: Since $\mathcal{I}_S \subseteq \mathcal{I}_{S'}$, by inspection on Fig. 8, we have $R_j = b?\, P_1 : P_2$, with $b \in \{\mathtt{tt}, \mathtt{ff}\}$ and the thesis follows as in the base case.

Case $U_j \in \{[\![R_j \mid R_k]\!]\}$: By inspection on Def. 30, combined with Cor. 2, $U_j \in \{[\![x \triangleleft l_j.Q \mid y \triangleright \{l_i : Q_i\}_{i\in I}]\!]\}$, for some Q, Q_i, l_j, and either

(i) $U_j \cong_\ell^\pi \overline{\mathsf{bra}(y, l_j)} \parallel \forall z(\mathsf{bra}(z, l_j) \otimes \{z{:}x\} \to [\![Q]\!]) \parallel \forall\epsilon(l_j = l_j \to [\![Q_j]\!])$, or

(ii) $U_j \cong_\ell^\pi [\![Q]\!] \parallel \forall\epsilon(l_j = l_j \to [\![Q_j]\!])$.

We analyze each case:

Case (i): If $U_j = \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to [\![Q]\!]) \parallel \forall \epsilon(l_j = l_j \to [\![Q_j]\!])$ then, by Cor. 3, there exists S_0 such that:

$$S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![x \triangleleft l_j.Q \mid y \triangleright \{l_i\, Q_i\}_{i\in I}]\!] \parallel U_1 \parallel \cdots \parallel U_n \parallel J]$$

by the semantics in Fig. 6 and Cor. 2:

$$\begin{aligned} S_0 &\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel U_j \parallel U_1 \parallel \cdots \parallel U_n \parallel J] = S \\ &\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to [\![Q]\!]) \parallel [\![Q_j]\!] \\ &\qquad \parallel U_1 \parallel \cdots \parallel U_n \parallel J \parallel] = S' \\ &\xrightarrow{\tau}_\ell \cong^\pi_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![Q]\!] \parallel [\![Q_j]\!] \parallel U_1 \parallel \cdots \parallel U_n \parallel J \parallel \prod_{h\in I} \forall \epsilon(l_h = l_j \to [\![Q_i]\!])] \end{aligned}$$

The proof finalizes by letting $J' = J \parallel \prod_{h\in I} \forall \epsilon(l_h = l_j \to [\![Q_i]\!])$.

Case (ii): If $U_j = [\![Q]\!] \parallel \forall \epsilon(l_j = l_j \to [\![Q_j]\!])$ then, by Def. 30 and Not. 14, there exists S_0 such that:

$$S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel (\!|x \triangleleft l_j.Q \mid y \triangleright \{l_i\, Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U_1 \parallel \cdots \parallel U_n \parallel J]$$

by the semantics in Fig. 6:

$$\begin{aligned} S_0 &\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel \overline{\mathsf{bra}(y,l_j)} \parallel \forall z(\mathsf{bra}(z,l_j) \otimes \{z{:}x\} \to [\![Q]\!]) \parallel [\![Q_j]\!] \\ &\qquad \parallel U_1 \parallel \cdots \parallel U_n \parallel J] = S \\ &\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![Q]\!] \parallel [\![Q_j]\!] \parallel U_1 \parallel \cdots \parallel U_n \parallel J \parallel \prod_{h\in I} \forall \epsilon(l_h = l_j \to [\![Q_i]\!])] = S' \end{aligned}$$

The proof finishes by letting $J' = J \parallel \prod_{h\in I} \forall \epsilon(l_h = l_j \to [\![Q_i]\!])$. □

Proposition 1 *Suppose S is a target term (cf. Def. 26).*

1. *$S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{c_1} \parallel \cdots \parallel \overline{c_n} \parallel Q_1 \parallel \cdots \parallel Q_k]$ with $n, k \geq 1$, where every $c_j = \gamma_j(x_j, m_j)$ (with $1 \leq j \leq n$), for some value or label m_j, and every Q_i (with $1 \leq i \leq k$) is an abstraction (possibly replicated).*
2. *For every $i, j \in \{1, \ldots, n\}$, $i \neq j$ implies $c_i = \gamma_i(x_i, m_i)$, $c_j = \gamma_j(x_j, m_j)$, and $x_i \neq x_j$.*

Proof *(see Page 32)* The first part of the statement follows immediately from the definition of $[\![\cdot]\!]$ (cf. Def. 25), Lem. 9, and by applying the structural congruence of `lcc` in S.

The second part of the statement is proven by contradiction. We assume that $S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{c_1} \parallel \cdots \parallel \overline{c_n} \parallel Q]$, where $Q = Q_1 \parallel \cdots \parallel Q_k$ where every Q_r, $r \in \{1, \ldots, k\}$ is an abstraction (possibly replicated) and that there exist $i, j \in \{1, \ldots, n\}$ such that $i \neq j$, $c_i = \gamma(x_1, m_1)$, $c_j = \gamma(x_2, m_2)$, and $x_1 = x_2$. We proceed by a case analysis on γ; there are four cases, we only show the cases $\gamma = \mathsf{snd}$ and $\gamma = \mathsf{rcv}$.

1. Suppose that $\gamma = \mathsf{snd}$. By assumption,

$$S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{c_1} \parallel \cdots \parallel \overline{\mathsf{snd}(x, v_1)} \parallel \cdots \parallel \overline{\mathsf{snd}(x, v_2)} \parallel \cdots \parallel \overline{c_n} \parallel Q]$$

   Moreover, by Def. 26, S must come from the translation of a well-typed term. By Fig. 8, it must be the case that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{c_1} \parallel \cdots \parallel [\![x\langle v_1\rangle.P_1]\!] \parallel \cdots \parallel [\![x\langle v_2\rangle.P_2]\!] \parallel \cdots \parallel \overline{c_n} \parallel Q']$$

   for some Q' that does not contain the abstractions used to build the translations $[\![x\langle v_k\rangle.P_k]\!]$, $k \in \{1, 2\}$. This implies, by Fig. 8, that S comes from a π process that contains two outputs on the same channel x in parallel. This contradicts the well-formedness assumption that follows from Thm. 4, finishing the proof.
2. Suppose that $\gamma = \mathsf{rcv}$. The proof has the same structure as the one above. The only difference is that rather than the translation of output processes, we must consider intermediate redexes (cf. Fig. 11). Similarly as above, we will find that the well-formedness assumption induced by typing is violated, thus reaching a contradiction. □

Lemma 12 (Invariants of Target Terms (II): Consuming Information) *Let P be a well-typed π program. For any S, S' such that $[\![P]\!] \xRightarrow{\tau} S \xrightarrow{\tau}_\ell S'$ and $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$ the following holds, for some U:*

(1) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{snd}(x_1, v)\}$ then all the following hold:
- *(a) $S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![x_1\langle v\rangle.P_1 \mid \diamond\, y_1(z).P_2]\!] \parallel U]$;*
- *(b) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x_1\langle v\rangle.P_1 \mid \diamond\, y_1(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;*
- *(c) $S' \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_2\{^v/_z\}]\!] \parallel S'' \parallel U]$, where $S'' = *\,[\![y(z).P_2]\!]$ or $S'' = \overline{\mathtt{tt}}$.*

(2) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{rcv}(x_1, v)\}$ then there exists S_0 such that $[\![P]\!] \xRightarrow{\tau} S_0 \xrightarrow{\tau}_\ell S$ and all of the following hold:
- *(a) $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![y_1\langle v\rangle.P_1 \mid \diamond\; x_1(z).P_2]\!] \parallel U]$;*
- *(b) $S = C_{\widetilde{x}\widetilde{y}}[(\!|y_1\langle v\rangle.P_1 \mid \diamond\, x_1(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;*
- *(c) $S' = C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_2\{^v/_z\}]\!] \parallel S'_1 \parallel U]$, where $S'_1 = *\,[\![y(z).P_2]\!]$ or $S'_1 = \overline{\mathtt{tt}}$.*

(3) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{sel}(x_1, l_j)\}$ then all of the following hold:
- *(a) $S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x_1{:}y_1\}} \parallel [\![x_1 \triangleleft l.P_1 \mid y_1 \triangleright \{l_i : P_i\}_{i\in I}]\!]U]$;*
- *(b) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|x_1 \triangleleft l.P_1 \mid y_1 \triangleright \{l_i : P_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U]$;*
- *(c) $S_1 \xrightarrow{\tau}{}^2_\ell \cong^\pi_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1 \mid P_j]\!] \parallel U']$, with $U' \equiv U \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_h]\!])$.*

(4) If $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{bra}(x, l_j)\}$, then there exists $S_0 \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![y \triangleleft l_j.Q \mid x \triangleright \{l_i\; Q_i\}_{i\in I}]\!] \parallel U]$ such that $[\![P]\!] \xRightarrow{\tau} S_0$ and either:
- *(a) All of the following hold:*
  - *(i) $S_0 \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.Q \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U] \xrightarrow{\tau}_\ell S$,*
  - *(ii) $S = C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.Q \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^3_{\widetilde{x}\widetilde{y}} \parallel U]$ (and)*
  - *(iii) $S' = C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![Q \mid Q_j]\!] \parallel U']$.*
- *(b) All of the following hold:*
  - *(i) $S_0 \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.P \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U] \equiv S$,*
  - *(ii) $S' = C_{\widetilde{x}\widetilde{y}}[(\!|y \triangleleft l_j.P \mid x \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel U]$ (and)*
  - *(iii) $S' \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[\overline{\{x{:}y\}} \parallel [\![P \mid Q_j]\!] \parallel U']$.*

with $U' \equiv U \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_h]\!])$.

Proof *(see Page 33)* By induction on the transition $\xRightarrow{\tau}\xrightarrow{\tau}_\ell$. By Lem. 1:

$$[\![P]\!] \equiv \exists\widetilde{x}, \widetilde{y}.([\![R_1]\!] \parallel \cdots \parallel [\![R_n]\!] \parallel !\overline{\bigotimes_{\substack{x_i\in\widetilde{x},\\ y_i\in\widetilde{y}}} \{x_i{:}y_i\}}) \tag{1}$$

where each R_i is a pre-redex or a conditional process. Also, by Lem. 10(b), the difference of observables between a process and the process obtained in a single τ-transition is a singleton. Therefore, we apply a case analysis on each one of those singletons.

Base Case: Then $[\![P]\!] \xRightarrow{\tau} [\![P]\!] \xrightarrow{\tau}_\ell S'$. Let $[\![P]\!] = S$. By assumption, $S \xrightarrow{\tau}_\ell S'$ and $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$. Thus, there is a $c \in \mathcal{I}_S$ such that $c \notin \mathcal{I}_{S'}$. Considering Eq. (1) and by inspection on Fig. 8 we only analyze from Case (2), as cases (1) and (3) do not apply: Case (1) does not apply as it does not entail constraint consumption (cf. Lem. 10(a)) and Case (3) does not apply as there are no intermediate redexes in S.
Case $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{snd}(x_1, v)\}$: By Lem. 6, there exists j such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![R_j]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus j} [\![R_i]\!]]$$

where $[\![R_j]\!] \equiv [\![x_j\langle v\rangle.Q]\!]$. Since $\mathcal{I}_S \not\subseteq \mathcal{I}_{S'}$, and every $c \in \mathcal{I}_S$ is unique, by inspection on Fig. 8 and Lem. 7, there must exist an R_k such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![x_j\langle v\rangle.Q]\!] \parallel [\![R_k]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]]$$

where $[\![R_k]\!] \equiv [\![y_j(z).Q']\!]$ or $[\![R_k]\!] \equiv [\![*\,y_j(z).Q']\!]$. Without loss of generality, we only show the case for $[\![R_k]\!] \equiv [\![y_j(z).Q']\!]$:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![x_j\langle v\rangle.Q]\!] \parallel [\![y_j(z).Q']\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]]$$

By the semantics of `lcc` (cf. Fig. 6) and Lem. 8:

$$S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x_j\langle v\rangle.Q \mid y_j(z).Q'|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]]$$
$$\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q]\!] \parallel [\![Q'\{v/z\}]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]] = S'$$

Case $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{sel}(x_1, l_j)\}$: By Lem. 6, there exists j such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![R_j]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus j} [\![R_i]\!]]$$

where $[\![R_j]\!] \equiv [\![x_j \triangleleft l_j.Q]\!]$. Furthermore, by following the same analysis as in the previous case, there must exists R_j, such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[[\![x_j \triangleleft l_j.Q]\!] \parallel [\![R_k]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]]$$

where $[\![R_k]\!] \equiv [\![y_j \triangleright \{l_i : Q_i\}_{i\in I}]\!]$. Then, by the semantics of `lcc` (cf. Fig. 6):

$$S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y_j \triangleright \{l_i : Q_i\}_{i\in I}|\!)^1_{\widetilde{x}\widetilde{y}} \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]] = S'$$
$$\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|y_j \triangleright \{l_i : Q_i\}_{i\in I}|\!)^2_{\widetilde{x}\widetilde{y}} \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!]]$$
$$\xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q]\!] \parallel [\![Q_j]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!] \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_i]\!])]$$
$$\cong^\pi_\ell \exists\widetilde{x},\widetilde{y}.([\![Q]\!] \parallel [\![Q_j]\!] \parallel \prod_{i\in\{1\ldots n\}\setminus\{j,k\}} [\![R_i]\!] \parallel \prod_{h\in I} \forall\epsilon(l_h = l_j \to [\![Q_i]\!]))$$

Inductive Case: By IH, $[\![P]\!] \stackrel{\tau}{\Longrightarrow} S \xrightarrow{\tau}_\ell S'$ satisfies the property for m steps (i.e., $[\![P]\!] \xrightarrow{\tau}^{m-1}_\ell S \xrightarrow{\tau}_\ell S'$). Therefore:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n \parallel J]$$

for some junk J and for all $i \in \{1, \ldots, n\}$ either:

1. $U_i = [\![R_k]\!]$, where R_k is a conditional redex reachable from P;
2. $U_i = [\![R_k]\!]$, where R_k is a pre-redex reachable from P;
3. $U_i \in \{[\![R_k \mid R_j]\!]\}$, where redex $R_k \mid R_j$ is reachable from P.

We now have to prove for $k = m + 1$:

$$[\![P]\!] \xrightarrow{\tau}^k_\ell S \xrightarrow{\tau}_\ell S'$$

Since $S \xrightarrow{\tau}_\ell S'$, there exists $[\![R_j]\!]$ such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[U_j \parallel \prod_{i\in\{1\ldots n\}\setminus j} U_i \parallel J]$$
$$S' \equiv C_{\widetilde{x}\widetilde{y}}[U'_j \parallel \prod_{i\in\{1\ldots n\}\setminus j} U_i \parallel J']$$

As above, we distinguish only cases for (2), (3). Notice that using Lem. 10(a) and Lem. 11 we can discard case (1):

Case (2): Proceeds as the base case, by distinguishing cases between the consumed constraints. The cases correspond to constraints snd, sel (cf. Fig. 7).

Case (3): By inspection on Def. 30 and Not. 14, we distinguish two cases corresponding to predicates rcv, bra (cf. Fig. 7):

Case $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{rcv}(x_1, v)\}$: By Lem. 6 and Lem. 7, there exists j such that:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[U_j \parallel U_1 \parallel \cdots \parallel U_n \parallel J]$$

where $U_j = \overline{\mathsf{rcv}(x_j, v)} \parallel W$, for some W. By inspection on Def. 30 and Lem. 7, there exists U_k such that $U_j \parallel U_k \in \{\![x_j(z).Q \mid y_j\langle v\rangle.Q']\!\}$ or $U_j \parallel U_k \in \{\![* x_j(z).Q \mid y_j\langle v\rangle.Q']\!\}$, for some x_j, y_j, v. Without loss of generality, we will only analyze the case when $U_j \parallel U_k \in \{\![x_j(z).Q \mid y_j\langle v\rangle.Q']\!\}$. By Def. 30 and Not. 14, $U_j = (\!|x_j(y).Q \mid y_j\langle v\rangle.Q'|\!)^1_{\widetilde{x}\widetilde{y}}$. By expanding the previous definitions:

$$S \equiv C_{\widetilde{x}\widetilde{y}}[\overline{\mathsf{rcv}(x_j, v)} \parallel \forall w(\mathsf{rcv}(w, v) \otimes \{w{:}y_j\} \to [\![Q]\!]) \parallel [\![Q'\{v/z\}]\!] \parallel U_1 \parallel \cdots \parallel U_n \parallel J]$$

and by the application of Rule (C:SYNC) in Fig. 6 (i.e., the `lcc` semantics):

$$S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![Q]\!] \parallel [\![Q'\{v/z\}]\!] \parallel U_1 \parallel J]$$

Case $\mathcal{I}_S \setminus \mathcal{I}_{S'} = \{\mathsf{bra}(x, l_j)\}$: This case proceeds as above. The conclusion is reached using the same analysis as in the inductive case in the proof of Lem. 11. □

C.5 A Diamond Property for Target Terms

Lemma 13 *Let S be a target term (cf. Def. 26) and x, y be endpoints. Then, $S \xrightarrow{\tau}_\ell S'$ if and only if $S \xrightarrow{\eta}_\ell S'$ where $\eta \in \{\alpha(x, y) \mid \alpha \in \{\mathtt{IO}, \mathtt{SL}, \mathtt{RP}, \mathtt{IO}_1, \mathtt{RP}_1, \mathtt{SL}_1, \mathtt{SL}_2, \mathtt{SL}_3\} \wedge x, y \in \mathcal{V}_\pi\} \cup \{\mathtt{CD}(-)\}$.*

Proof (see Page 35) We prove both directions:

⇒) By Cor. 4, $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel U_n]$, with $U_i = [\![R_i]\!]$ for some pre-redex R_i (cf. Def. 9). We take then an arbitrary U_i, $i \in \{1, \ldots, n\}$. We apply a case analysis on U_i. There are 11 cases corresponding to each possible shape of U_i. We only show three cases; the rest are similar:

Case $U_i = [\![x\langle v\rangle.P_1]\!]$: We distinguish two sub-cases, depending on whether there exists U_j such that $U_j = [\![y(z).P_2]\!]$ and $x \in \widetilde{x}, y \in \widetilde{y}$ or not. The latter case is vacuously true, as there would not be any transition to check. We show the former case:

Sub-case $\exists U_j.(U_j = [\![y(z).P_2]\!] \wedge x \in \widetilde{x}, y \in \widetilde{y})$:

1. $S \equiv C_{\widetilde{x}\widetilde{y}}[[\![x\langle v\rangle.P_1]\!] \parallel [\![y(z).P_2]\!] \parallel U_1 \parallel \cdots \parallel U_n]$ (Assumption).
2. $S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[(\!|x\langle v\rangle.P_1 \mid y(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U_1 \parallel \cdots \parallel U_n] = S'$ (Rule (C:SYNC) - Fig. 6, (1), Assumption).
3. $S \xrightarrow{\mathtt{IO}_1(x,y)}_\ell S'$ (Def. 31, (1), (2)).

Case $U_i = [\![x(y).P_1]\!]$: Symmetric to the previous case, as $U_j = [\![y\langle v\rangle.P_2]\!]$.

Case $U_i = (\!|x\langle v\rangle.P_1 \mid y(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}}$:

1. $S \equiv C_{\widetilde{x}\widetilde{y}}[(\!|x\langle v\rangle.P_1 \mid y(z).P_2|\!)^1_{\widetilde{x}\widetilde{y}} \parallel U_1 \parallel \cdots \parallel U_n]$ (Assumption).
2. $S \xrightarrow{\tau}_\ell C_{\widetilde{x}\widetilde{y}}[[\![P_1]\!] \parallel [\![P_2]\!]\{v/z\} \parallel U_1 \parallel \cdots \parallel U_n] = S'$ (Rule (C:SYNC) - Fig. 6, (1), Assumption).
3. $S \xrightarrow{\mathtt{IO}_1(x,y)}_\ell S'$ (Def. 31, (1), (2)).

⇐) This direction proceeds by applying a case analysis on label η. Each case then will proceed by applying Rule (C:SYNC) in Fig. 6 and showing that the transition yields the correct process. □

Lemma 14 *Let S be a target term such that $S \xrightarrow{\omega}_\ell S_1$ and $S \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_2$, where $\gamma(\widetilde{x}\widetilde{y})$ is a closing sequence (cf. Not. 16). Then, there exists S_3 such that $S_1 \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_3$ and $S_2 \xrightarrow{\omega}_\ell S_3$.*

Proof (see Page 37) By induction on the length n of $|\gamma(\widetilde{x}\widetilde{y})|$.

Base Case: $n = 0$. Then:

1. $S \xrightarrow{\omega}_\ell S_1$ (Assumption).
2. $S \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S$ (Assumption)

3. $S_1 \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_1$ (Fig. 6)

Conclude by letting $S_1 = S_1$, $S_2 = S$ and $S_3 = S_1$ and using (1),(3).

Inductive Step: $n \geq 1$. We state the IH:

IH: If $S \xrightarrow{\omega}_\ell S_1$ and $S \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0 \xrightarrow{\kappa}_\ell S_2$, then there exists S_0' such that $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0'$ and $S_0 \xrightarrow{\omega}_\ell S_0'$.

We distinguish cases for $\kappa \in \{\mathtt{IO}_1, \mathtt{RP}_1, \mathtt{CD}, \mathtt{SL}_2, \mathtt{SL}_3\}$. There are five cases and each one has four sub-cases, corresponding to the opening labels $\{\mathtt{IO}, \mathtt{SL}, \mathtt{RP}, \mathtt{SL}_1\}$.

We detail three cases: $\kappa = \mathtt{IO}_1$, $\kappa = \mathtt{RP}_1$ and $\kappa = \mathtt{CD}$, as the other are similar:

Case $\kappa = \mathtt{IO}_1$: As mentioned above, there are four sub-cases depending on ω. We enumerate them below and only detail $\omega = \mathtt{IO}$, $\omega = \mathtt{RP}$:

Sub-case $\omega = \mathtt{IO}$: Suppose, without loss of generality, that the actions take place on endpoints x, y and w, z. Furthermore, by typing and Cor. 1 and Def. 10, it cannot be the case that $x = w$ and $y = z$, because this would imply that there are more than one output prefixed on x. Then, we we only consider the case when $x \neq w$ and $y \neq z$:

Sub-case $x \neq w \wedge y \neq z$: We proceed as follows:

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J'], m \geq 1$ (IH, (1), Lem. 9).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_0'$ (IH)
6. $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0'$ (IH).

We can then reduce the proof to the existence of some S_3 such that

- $S_2 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_3$ and
- $S_0' \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$

We have:

(a) $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U_m' \parallel J']$.

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$.

then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$$

and we can show by Fig. 6 that $S_0' \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_3$, which concludes the proof.

Sub-case $\omega = \mathtt{RP}$: As above, assume the actions take place in variables x, y and w, z. It is not possible for them to happen in the same variable, by Cor. 1.

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid * y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J'], m \geq 1$ (IH, (1), Lem. 9).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_0'$ (IH)
6. $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid * y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0'$ (IH).

We reduce the proof to show to the existence of some S_3 such that

– $S_2 \xrightarrow{\texttt{RP}(x,y)}_\ell S_3$ and
– $S_0' \xrightarrow{\texttt{IO}_1(w,z)}_\ell S_3$

We have:

(a) $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U_m' \parallel J']$.

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$.

Then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U_m' \parallel J']$$

and we can show by Fig. 6 that $S_0' \xrightarrow{\texttt{IO}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\texttt{RP}(x,y)}_\ell S_3$, which concludes the proof.

Sub-case $\omega = \texttt{SL}()$: Similarly as above.

Sub-case $\omega = \texttt{SL}_1()$: Similarly as above.

Case $\kappa = \texttt{RP}_1$: We proceed similarly as above. The most interesting case is whenever $\omega = \texttt{RP}$:

Sub-case $\omega = \texttt{RP}$: As above, assume the actions take place in variables x, y and w, z. It is not possible for them to happen in the same variable, by Cor. 1.

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid *\, z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J'], m \geq 1$ (IH, (1), Lem. 9).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \ldots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![*\, z(u_2).Q_4]\!] \parallel \ldots \parallel U_m' \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\texttt{RP}(x,y)}_\ell S_0'$ (IH)
6. $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid *\, z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xrightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0'$ (IH).

We reduce the proof to show that there exists some S_3 such that

– $S_2 \xrightarrow{\texttt{RP}(x,y)}_\ell S_3$ and
– $S_0' \xrightarrow{\texttt{RP}_1(w,z)}_\ell S_3$

We have:

(a) $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid *\, z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U_m' \parallel J']$

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \ldots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![*\, z(y_2).Q_4]\!] \parallel \ldots \parallel U_m' \parallel J']$

Then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![*\, z(y_2).Q_4]\!] \cdots \parallel U_m' \parallel J']$$

and we can show by Fig. 6 that $S_0' \xrightarrow{\texttt{RP}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\texttt{RP}(x,y)}_\ell S_3$, which concludes the proof.

Sub-case $\omega = \texttt{IO}$: Similarly as above.

Sub-case $\omega = \texttt{SL}$: Similarly as above.

Sub-case $\omega = \texttt{SL}_1$: Similarly as above.

Case $\kappa = \texttt{CD}$: As above we distinguish four cases. We only show Sub-case $\omega = \texttt{IO}$, as the other cases are similar:

Sub-case $\omega = \texttt{IO}$: Assume that the $\texttt{IO}$ transition happens on endpoints x, y. Since $\texttt{CD}$ does not occur on any channels, we do not need to assume more endpoints:

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(z).Q_2]\!] \parallel \cdots \parallel U_n], n \geq 1$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n], n \geq 1$ ((1), Fig. 13).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(z).Q_2]\!] \parallel [\![b?\, Q_3 : Q_4]\!] \parallel \cdots \parallel U_m'], m \geq 1$, with $b \in \{\texttt{tt}, \texttt{ff}\}$ (IH, Fig. 13).

4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(z).Q_2]\!] \parallel [\![Q_i]\!] \parallel \cdots \parallel U'_m]$, $i \in \{3,4\}$ ((3), Fig. 13).
5. $S'_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel [\![b?\,Q_3 : Q_4]\!] \parallel \cdots \parallel U'_m]$, $m \geq 1$, with $b \in \{\texttt{tt}, \texttt{ff}\}$ (IH, Fig. 13).

Now, let $S_3 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel [\![Q_i]\!] \parallel \cdots \parallel U'_m]$, $m \geq 1$, with $b \in \{\texttt{tt}, \texttt{ff}\}$, $i \in \{3,4\}$.

It can be shown, by Fig. 13 that $S_2 \xrightarrow{\texttt{IO}(x,y)}_\ell S_3$ and $S'_0 \xrightarrow{\texttt{CD}(-)}_\ell S_3$ which by IH imply that $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})\texttt{CD}(-)} S_3$, concluding the proof.

Sub-case $\omega = \texttt{RP}$: Similarly as above.
Sub-case $\omega = \texttt{SL}$: Similarly as above.
Sub-case $\omega = \texttt{SL}_1$: Similarly as above.
Case $\kappa = \texttt{SL}_2$: Similarly as Case $\omega = \texttt{IO}_1$.
Case $\kappa = \texttt{SL}_3$: Similarly as Case $\omega = \texttt{IO}_1$. □

Lemma 15 *Suppose a well-typed π program P. For every sequence of labels $\gamma(\widetilde{x}\widetilde{y})$ such that $[\![P]\!] \xRightarrow{\gamma(\widetilde{x}\widetilde{y})}$ S, there exist Q, S', and $\gamma'(\widetilde{x}\widetilde{y})$ such that $P \longrightarrow^* Q$ and $S \xRightarrow{\gamma'(\widetilde{x}\widetilde{y})} S'$, with $\gamma'(\widetilde{x}\widetilde{y}) = \gamma(\widetilde{x}\widetilde{y})\downarrow$ (cf. Def. 35). Moreover, $[\![Q]\!] \cong^\pi_\ell S'$.*

Proof (see Page 38) By induction on $|\gamma(\widetilde{x}\widetilde{y})|$ and a case analysis on the last label of the sequence. The base case is immediate since $[\![P]\!] \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} [\![P]\!]$ and $P \longrightarrow^* P$.

For the inductive step, assume $|\gamma(\widetilde{x}\widetilde{y})| = n \geq 0$. We state the IH:

IH: if $[\![P]\!] \xRightarrow{\gamma(\widetilde{x}\widetilde{y})_0} S_0 \xrightarrow{\alpha_{n+1}}_\ell S$ then there exists Q_0, S'_0 and $\gamma'_0(\widetilde{x}\widetilde{y})$ such that $P \longrightarrow^* Q$, $S_0 \xRightarrow{\gamma'(\widetilde{x}\widetilde{y})} S'_0$, $\gamma'_0(\widetilde{x}\widetilde{y}) = \gamma(\widetilde{x}\widetilde{y})\downarrow$ and $S'_0 \cong^\pi_\ell [\![Q]\!]$.

Using the IH, the proof can be summarized by the diagram in Fig. 14, where we must show the existence of the dotted arrows. Details follow:

Base Case: $n = 0$. Then:
1. $S \xrightarrow{\omega}_\ell S_1$ (Assumption).
2. $S \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S$ (Assumption)
3. $S_1 \xRightarrow{\gamma(\widetilde{x}\widetilde{y})} S_1$ (Fig. 6)

Conclude by letting $S_1 = S_1$, $S_2 = S$ and $S_3 = S_1$ and using (1),(3).

Inductive Step: $n \geq 1$. We state the IH:

IH: If $S \xrightarrow{\omega}_\ell S_1$ and $S \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0 \xrightarrow{\kappa}_\ell S_2$, then there exists S'_0 such that $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S'_0$ and $S_0 \xrightarrow{\omega}_\ell S'_0$.

We distinguish cases for $\kappa \in \{\texttt{IO}_1, \texttt{RP}_1, \texttt{CD}, \texttt{SL}_2, \texttt{SL}_3\}$. There are five cases and each one has four sub-cases, corresponding to the opening labels $\{\texttt{IO}, \texttt{SL}, \texttt{RP}, \texttt{SL}_1\}$. We detail three cases: $\kappa = \texttt{IO}_1$, $\kappa = \texttt{RP}_1$ and $\kappa = \texttt{CD}$, as the other are similar:

Case $\kappa = \texttt{IO}_1$: As mentioned above, there are four sub-cases depending on ω. We enumerate them below and only detail $\omega = \texttt{IO}$, $\omega = \texttt{RP}$:

Sub-case $\omega = \texttt{IO}$: Suppose, without loss of generality, that the actions take place on endpoints x, y and w, z. Furthermore, by typing and Cor. 1 and Def. 10, it cannot be the case that $x = w$ and $y = z$, because this would imply that there is more than one output prefixed on x. Then, we we only consider the case when $x \neq w$ and $y \neq z$:

Sub-case $x \neq w \wedge y \neq z$: We proceed as follows:
1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U'_m \parallel J']$, $m \geq 1$ (IH, (1), Lem. 9).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\texttt{IO}(x,y)}_\ell S'_0$ (IH)

6. $S'_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U'_m \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S'_0$ (IH).

We can then reduce the proof to the existence of some S_3 such that

- $S_2 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_3$ and
- $S'_0 \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$

We have:

(a) $S'_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U'_m \parallel J']$.

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$.

Then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$$

and we can show by Fig. 6 that $S'_0 \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_3$, which concludes the proof.

Sub-case $\omega = \mathtt{RP}$: As above, assume the actions take place in variables x, y and w, z. It is not possible for them to happen in the same variable, by Cor. 1.

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U'_m \parallel J'], m \geq 1$ (IH, (1), Lem. 9).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\mathtt{RP}(x,y)}_\ell S'_0$ (IH)
6. $S'_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U'_m \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S'_0$ (IH).

We reduce the proof to the existence of some S_3 such that

- $S_2 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_3$
- $S'_0 \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$

We have:

(a) $S'_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U'_m \parallel J']$.

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$.

Then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel \cdots \parallel U'_m \parallel J']$$

and we can show by Fig. 6 that $S'_0 \xrightarrow{\mathtt{IO}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_3$, which concludes the proof.

Sub-cases $\omega = \mathtt{SL}$ and $\omega = \mathtt{SL}_1$: Similarly as above.

Case $\kappa = \mathtt{RP}_1$: We proceed similarly as above. The most interesting case is whenever $\omega = \mathtt{RP}$:

Sub-case $\omega = \mathtt{RP}$: As above, assume the actions take place in variables x, y and w, z. It is not possible for them to happen in the same variable, by Cor. 1.

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel \cdots \parallel U_n \parallel J]$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid *\, y(u_1).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n \parallel J]$ (Fig. 6,(1)).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U'_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![*\, y(u_1).Q_2]\!] \parallel (\!|w\langle v'\rangle.Q_3 \mid *\, z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U'_m \parallel J'], m \geq 1$ (IH, (1), Lem. 9).

4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \ldots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![* z(u_2).Q_4]\!] \parallel \ldots \parallel U_m' \parallel J']$ ((3), Fig. 6).
5. $S_0 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_0'$ (IH)
6. $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid * y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid * z(u_2).Q_4|\!)^1_{wz} \parallel \cdots \parallel U_m' \parallel J']$ (Fig. 6, (5)).
7. $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})} S_0'$ (IH).

We reduce the proof to the existence of some S_3 such that

- $S_2 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_3$
- $S_0' \xrightarrow{\mathtt{RP}_1(w,z)}_\ell S_3$

We have:

(a) $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid * y(u_1).Q_2|\!)^1_{xy} \parallel (\!|w\langle v'\rangle.Q_3 \mid * z(u_2).Q_4|\!)^1_{xy} \parallel \cdots \parallel U_m' \parallel J']$.

(b) $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \ldots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![* y(u_1).Q_2]\!] \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![* z(y_2).Q_4]\!] \parallel \ldots \parallel U_m' \parallel J']$.

Then, let

$$S_3 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid * y(u_1).Q_2|\!)^1_{xy} \parallel [\![Q_3]\!] \parallel [\![Q_4]\!]\{v'/u_2\} \parallel [\![* z(y_2).Q_4]\!] \cdots \parallel U_m' \parallel J']$$

and we can show by Fig. 6 that $S_0' \xrightarrow{\mathtt{RP}_1(w,z)}_\ell S_3$ and $S_2 \xrightarrow{\mathtt{RP}(x,y)}_\ell S_3$, which concludes the proof.

Sub-cases $\omega = \mathtt{IO}$, $\omega = \mathtt{SL}$, and $\omega = \mathtt{SL}_1$: Similarly as above.

Case $\kappa = \mathtt{CD}$: As above we distinguish four cases. We only show Sub-case $\omega = \mathtt{IO}$, as the other cases are similar:

Sub-case $\omega = \mathtt{IO}$: Assume that the $\mathtt{IO}$ transition happens on endpoints x, y. Since $\mathtt{CD}$ does not occur on any channels, we do not need to assume more endpoints:

1. $S = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(z).Q_2]\!] \parallel \cdots \parallel U_n]$, $n \geq 1$ (Assumption, Fig. 13, Lem. 9).
2. $S_1 = C_{\widetilde{x}\widetilde{y}}[U_1 \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel \cdots \parallel U_n]$, $n \geq 1$ ((1), Fig. 13).
3. $S_0 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!][\![y(z).Q_2]\!] \parallel [\![b?\, Q_3 : Q_4]\!] \parallel \cdots \parallel U_m']$, $m \geq 1$, with $b \in \{\mathtt{tt}, \mathtt{ff}\}$ (IH, Fig. 13).
4. $S_2 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel [\![x\langle v\rangle.Q_1]\!] \parallel [\![y(z).Q_2]\!] \parallel [\![Q_i]\!] \parallel \cdots \parallel U_m']$, $i \in \{3, 4\}$ ((3), Fig. 13).
5. $S_0' = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel [\![b?\, Q_3 : Q_4]\!] \parallel \cdots \parallel U_m']$, $m \geq 1$, with $b \in \{\mathtt{tt}, \mathtt{ff}\}$ (IH, Fig. 13).

Now, let $S_3 = C_{\widetilde{x}\widetilde{y}}[U_1' \parallel \cdots \parallel (\!|x\langle v\rangle.Q_1 \mid y(z).Q_2|\!)^1_{xy} \parallel [\![Q_i]\!] \parallel \cdots \parallel U_m']$, $m \geq 1$, with $b \in \{\mathtt{tt}, \mathtt{ff}\}$, $i \in \{3, 4\}$.

It can be shown, by Fig. 13 that $S_2 \xrightarrow{\mathtt{IO}(x,y)}_\ell S_3$ and $S_0' \xrightarrow{\mathtt{CD}(-)}_\ell S_3$ which by IH imply that $S_1 \xRightarrow{\gamma_0(\widetilde{x}\widetilde{y})\mathtt{CD}(-)} S_3$, concluding the proof.

Sub-case $\omega = \mathtt{RP}$, $\omega = \mathtt{SL}$, and $\omega = \mathtt{SL}_1$: Similarly as above.

Cases $\kappa = \mathtt{SL}_2$ and $\kappa = \mathtt{SL}_3$: Similarly as Case $\omega = \mathtt{IO}_1$.

□