Concurrent Flexible Reversibility*

Ivan Lanese[1], Michael Lienhardt[2], Claudio Antares Mezzina[3], Alan Schmitt[4], and Jean-Bernard Stefani[4]

[1] Focus Team, University of Bologna/Inria, Italy lanese@cs.unibo.it
[2] PPS Laboratory, Paris Diderot University, France lienhard@cs.unibo.it
[3] SOA Unit, FBK, Trento, Italy mezzina@fbk.eu
[4] Inria, France alan.schmitt@inria.fr, jean-bernard.stefani@inria.fr

Abstract. Concurrent reversibility has been studied in different areas, such as biological systems and dependable distributed systems. However, only "rigid" reversibility has been considered, allowing to go back to a past state and restart the exact same computation, possibly leading to divergence. In this paper we present croll-π, a concurrent calculus featuring *flexible reversibility* where it is possible to specify alternatives to a computation, to be used upon rollback. Alternatives in croll-π are attached to messages. We show the robustness of this mechanism by encoding more complex idioms for specifying flexible reversibility. Moreover, we illustrate the benefits of our approach from both the programming and the theoretical point of view. From the programming point of view we present a simple solution of the Eight Queens problem. From the theoretical point of view we encode a calculus based on interacting transactions. Our encoding improves on the original approach by avoiding some spurious undo of actions.

1 Introduction

Reversible programs can be executed both in the standard, forward direction as well as in the backward direction, to go back to past states. Reversible programming is attracting much interest for its potential in several areas. For instance, chemical and biological reactions are typically bidirectional, and the direction of execution is fixed by environmental conditions such as temperature. Similarly, quantum computations are reversible as long as they are not observed. Reversibility is also used for backtracking in the exploration of a program state-space toward a solution, either as part of the design of the programming language as in Prolog, or to implement transactions. We are particularly interested in the use of reversibility for modeling and programming concurrent reliable systems. In this setting, the main idea is that in case of an error the program backtracks to a past state where the decisions leading to the error have not been taken yet, so that a new forward execution may avoid repeating the (same) error.

* This work has been partially supported by the French National Research Agency (ANR), projects REVER ANR 11 INSE 007 and PiCoq ANR 10 BLAN 0305.

Reversibility has a non trivial interplay with concurrency. Understanding this interplay is fundamental in many of the areas above, e.g., for biological or reliable distributed systems, which are naturally concurrent. In the spirit of concurrency, independent threads of execution should be rolled-back independently, but causal dependencies between related threads should be taken into account.

This form of reversibility, termed *causal consistent*, was first introduced by RCCS [11], a reversible variant of CCS. RCCS paved the way to the definition of reversible variants of more expressive concurrent calculi [8, 18, 20, 22]. This line of research considered rigid, uncontrolled, step-by-step reversibility. *Step-by-step* means that each single step can be undone, as opposed, e.g., to checkpointing where many steps are undone at once. *Uncontrolled* means that there is no hint as to when to go forward and when to go backward, and up to where. *Rigid* means that the execution of a forward step followed by the corresponding backward step leads back to the starting state, where an identical computation can restart.

While these works have been useful to understand the basics of concurrent reversibility in different settings, the form of reversibility they provide is not sufficient in practice. Subsequent works in the literature concentrated on the problem of *control* of reversibility. Different forms of control have been proposed, tailored to the expected application area. For instance, in the case of chemical systems, it is reasonable to relate the direction of execution to some energy parameters, as studied in [2]. In the area of reliable systems, two forms of control of reversibility have been proposed: irreversible actions [12] and rollback [17].

These works were all based on rigid reversibility. However, rigid reversibility may not always be the best choice. In the setting of reliable systems, for instance, rigid reversibility means that to recover from an error a past state is reached. From this past state the computation that lead to the error is still possible. If the error was due to a transient fault, retrying the same computation may be enough to succeed. If the failure was permanent, instead, the program may redo the same error again and again, possibly forever. As a consequence, most of the processes in the calculi cited above have divergent computations.

Our goal is to overcome this limitation by providing the programmer with suitable linguistic constructs to specify what to do after a backward computation. Such constructs can be used to ensure that new forward computations explore new possibilities. To this end, we build on our previous work on roll-π [17], a calculus where concurrent reversibility is controlled by a specific operator roll γ. Executing this construct reverses the action referred by γ together with all the actions depending on it. Here, we propose a new calculus called croll-π, for compensating roll-π, as a framework for *flexible reversibility*. We attempt to keep croll-π as close as possible to roll-π while enabling many new possible applications. We thus simply replace roll-π communication messages $a\langle P\rangle$ by *messages with alternative* $a\langle P\rangle \div c\langle Q\rangle$. In forward computation, a message $a\langle P\rangle \div c\langle Q\rangle$ behaves exactly as $a\langle P\rangle$. However, if the interaction consuming it is reversed, the original message is not recreated—as would be the case with rigid reversibility—but the alternative $c\langle Q\rangle$ is released instead.

Our contributions are as follows. We show that such a small addition to the calculus greatly extends its expressive power. First we show that messages with alternative allow for programming different patterns for flexible reversibility. Second, we describe a solution for the Eight Queens problem based on state exploration, programmed on top of a croll-π proof-of-concept interpreter written in Maude [10]. And third, we show that croll-π can be used to model communicating transactions as described in [13]. Notably, the tracking of causality of croll-π is more precise than the one in [13], thus allowing to improve on the original proposal by avoiding some spurious undo of actions. Additionally, we study some relevant aspects of the behavioral theory of croll-π, including a context lemma for barbed congruence. This allows us to reason on croll-π programs, in particular to prove the correctness of the encodings of various primitives for flexible reversibility and of the transactional calculus of [13].

Outline. Section 2 gives an informal introduction to croll-π. Section 3 defines the croll-π calculus and its reduction semantics. It also introduces the basics of its behavioral theory. Section 4 presents various croll-π idioms for flexible reversibility. Section 5 outlines the croll-π interpreter in Maude and the solution for the Eight Queens problem. Section 6 presents an encoding and an analysis of the TransCCS constructs from [13]. Section 7 concludes the paper with related work and a mention of future studies. The paper includes short proof sketches for the main results. We refer to the online technical report [16] for full proofs.

2 Informal Presentation

Rigid reversibility in roll-π. The croll-π calculus is a conservative extension of the roll-π calculus introduced in [17].[5] We briefly review the roll-π constructs before presenting the extension added by croll-π. Processes in roll-π are essentially processes of the asynchronous higher-order π-calculus [24], extended with a rollback primitive. Processes in roll-π cannot directly execute, only *configurations* can. A configuration is essentially a parallel composition of *tagged processes* along with *memories* tracking past interactions and *connectors* tracing causality information. In a tagged process of the form $k : P$, the tag k uniquely identifies the process P in a given configuration. We often use the term *key* instead of tag.

The uniqueness of tags in configurations is achieved thanks to the following reduction rule that defines how parallel processes are split.

$$k : P \mid Q \longrightarrow \nu k_1, k_2.\, k \prec (k_1, k_2) \mid k_1 : P \mid k_2 : Q$$

In the above reduction, $\mid$ is the parallel composition operator and ν is the restriction operator, both standard from the π-calculus. Connector $k \prec (k_1, k_2)$ is used to remember that the process tagged by k has been split into two subprocesses identified by the new keys k_1 and k_2. Thus complex processes can be split into *threads*, where a thread is either a *message*, of the form $a\langle P\rangle$ (where a is

[5] The version of roll-π presented here is slightly refined w.r.t. the one in [17].

a channel name), a receiver process (also called a *trigger*), of the form $a(X) \triangleright_\gamma P$, or a *rollback* instruction of the form $\mathsf{roll}\ k$, where k is a key.

A *forward* communication step occurs when a message on a channel can be received by a trigger on the same channel. It takes the following form (roll-π is an asynchronous higher-order calculus).

$$(k_1 : a\langle P\rangle) \mid (k_2 : a(X) \triangleright_\gamma Q) \longrightarrow \nu k.\, k : Q\{^{P,k}/_{X,\gamma}\} \mid [\mu; k]$$

In this forward step, keys k_1 and k_2 identify threads consisting respectively of a message $a\langle P\rangle$ on channel a and a trigger $a(X) \triangleright_\gamma Q$ expecting a message on channel a. The result of the message input yields, as in higher-order π, the body of the trigger Q with the formal parameter X instantiated by the received value, i.e., process P. Message input also has three side effects: (i) the tagging of the newly created process $Q\{^{P,k}/_{X,\gamma}\}$ by a fresh key k; (ii) the creation of a memory $[\mu; k]$, which records the original two threads,[6] $\mu = (k_1 : a\langle P\rangle) \mid (k_2 : a(X) \triangleright_\gamma Q)$, together with key k; and (iii) the instantiation of variable γ with the newly created key k (the trigger construct is a binder both for its process parameter and its key parameter).

In roll-π, a forward computation, i.e., a series of forward reduction steps as above, can be perfectly undone by backward reductions triggered by the occurrence of an instruction of the form $\mathsf{roll}\ k$, where k refers to a previously instantiated memory. In roll-π, we have for instance the following forward and backward steps, where $M = (k_1 : a\langle Q\rangle) \mid (k_2 : a(X) \triangleright_\gamma X \mid \mathsf{roll}\ \gamma)$:

$$\begin{aligned} &M \longrightarrow \nu k.\, (k : Q \mid \mathsf{roll}\ k) \mid [M; k] \longrightarrow \\ &\nu k, k_3, k_4.\, k \prec (k_3, k_4) \mid k_3 : Q \mid k_4 : \mathsf{roll}\ k \mid [M; k] \longrightarrow M \end{aligned}$$

The communication between threads k_1 and k_2 in the first step and the split of process k into k_3 and k_4 are perfectly undone by the third (backward) step.

More generally, the set of memories and connectors of a configuration M provides us with an ordering $<:$ between the keys of M that reflects their causal dependency: $k' <: k$ means that key k' has key k as *causal descendant*. Thus, the effects of a rollback can be characterized as follows. When a rollback takes place in a configuration M, triggered by an instruction $k_r : \mathsf{roll}\ k$, it suppresses all threads and processes whose tag is a causal descendant of k, as well as all connectors $k' \prec (k_1, k_2)$ and memories $m = [k_1 : \tau_1 \mid k_2 : \tau_2; k']$ whose key k' is a causal descendant of k. When suppressing such a memory m, the rollback operation may release a thread $k_i : \tau_i$ if k_i is not a causal descendant of k (at least one of the threads of m must have k as causal antecedent if k' has k as causal antecedent). This is due to the fact that a thread that is not a causal descendant of k may be involved in a communication (and then captured into a memory) by a descendant of k. This thread can be seen as a resource that is taken from the environment through interaction, and it should be restored in case of rollback. Finally, rolling-back also releases the content μ of the memory $[\mu; k]$ targeted by the roll, reversing the corresponding communication step.

[6] Work can be done to store memories in a more efficient way. We will not consider this issue in the current paper; an approach can be found in [20].

Flexible reversibility in croll-π. In roll-π, a rollback perfectly undoes a computation originated by a specific message receipt. However, nothing prevents the same computation from taking place again and again (although not necessarily in the same context, as independent computations may have proceeded on their own in parallel). To allow for flexible reversibility, we extend roll-π with a single new construct, called a *message with alternative*. In croll-π, a message may now take the form $a\langle P\rangle \div C$, where alternative C may either be a message $c\langle Q\rangle \div \mathbf{0}$ with null alternative or the null process $\mathbf{0}$. When the message receipt of $k : a\langle P\rangle \div C$ is rolled-back, configuration $k : C$ is released instead of the original $k : a\langle P\rangle$, as would be the case in roll-π. (Only the alternative associated to the message in the memory $[\mu; k]$ targeted by the roll is released: other processes may be restored, but not modified.) For example, if $M = (k_1 : a\langle Q\rangle \div \mathbf{0}) \mid (k_2 : a(X) \rhd_\gamma X \mid \mathsf{roll}\ \gamma)$ then we have the following computation, where the communication leading to the rollback becomes disabled.

$$
\begin{aligned}
&M \longrightarrow \nu k.\,(k : Q \mid \mathsf{roll}\ k) \mid [M; k] \longrightarrow \\
&\nu k, k_3, k_4.\, k \prec (k_3, k_4) \mid k_3 : Q \mid k_4 : \mathsf{roll}\ k \mid [M; k] \longrightarrow \\
&k_1 : \mathbf{0} \mid (k_2 : a(X) \rhd_\gamma X \mid \mathsf{roll}\ \gamma)
\end{aligned}
$$

We will show that croll-π is powerful enough to devise various kinds of alternatives (see Section 4), whose implementation is not possible in roll-π (cf. Theorem 2). Also, thanks to the higher-order aspect of the calculus, the behavior of roll-π can still be programmed: rigid reversibility can be seen as a particular case of flexible reversibility. Thus, the introduction of messages with alternatives has limited impact on the definition of the syntax and of the operational semantics, but it has a strong impact on what can actually be modeled in the calculus and on its theory.

3 The croll-π Calculus: Syntax and Semantics

3.1 Syntax

Names, keys, and variables. We assume the existence of the following denumerable infinite mutually-disjoint sets: the set $\mathcal{N}$ of *names*, the set $\mathcal{K}$ of *keys*, the set $\mathcal{V}_\mathcal{K}$ of *key variables*, and the set $\mathcal{V}_\mathcal{P}$ of *process variables*. $\mathbb{N}$ denotes the set of natural numbers. We let (together with their decorated variants): a, b, c range over $\mathcal{N}$; h, k, l range over $\mathcal{K}$; u, v, w range over $\mathcal{N} \cup \mathcal{K}$; γ range over $\mathcal{V}_\mathcal{K}$; X, Y, Z range over $\mathcal{V}_\mathcal{P}$. We denote by $\tilde{u}$ a finite set $\{u_1, \ldots, u_n\}$.

Syntax. The syntax of the croll-π calculus is given in Figure 1. *Processes*, given by the P, Q productions, are the standard processes of the asynchronous higher-order π-calculus [24], except for the presence of the roll primitive, the extra bound tag variable in triggers, and messages with alternative that replace roll-π messages $a\langle P\rangle$. The alternative operator $\div$ binds more strongly than any other operator. *Configurations* in croll-π are given by the M, N productions. A configuration is built up from *tagged processes* $k : P$, *memories* $[\mu; k]$, and *connectors*

$$
\begin{array}{rl}
P, Q ::= & \mathbf{0} \mid X \mid \nu a.\, P \mid (P \mid Q) \mid a(X) \triangleright_\gamma P \mid a\langle P\rangle \div C \mid \mathsf{roll}\ k \mid \mathsf{roll}\ \gamma \\
M, N ::= & \mathbf{0} \mid \nu u.\, M \mid (M \mid N) \mid k : P \mid [\mu; k] \mid k \prec (k_1, k_2) \qquad C ::= a\langle P\rangle \div \mathbf{0} \mid \mathbf{0} \\
\mu ::= & (k_1 : a\langle P\rangle \div C) \mid (k_2 : a(X) \triangleright_\gamma Q) \\
& a, b, c \in \mathcal{N} \quad X, Y, Z \in \mathcal{V}_\mathcal{P} \quad \gamma \in \mathcal{V}_\mathcal{K} \quad u, v, w \in \mathcal{N} \cup \mathcal{K} \quad h, k, l \in \mathcal{K}
\end{array}
$$

Fig. 1. Syntax of croll-π

$k \prec (k_1, k_2)$. In a memory $[\mu; k]$, we call μ the *configuration part* of the memory and k its *key*. $\mathcal{P}$ denotes the set of croll-π processes and $\mathcal{C}$ the set of croll-π configurations. We let (together with their decorated variants) P, Q, R range over $\mathcal{P}$ and L, M, N range over $\mathcal{C}$. We call *thread* a process that is either a message with alternative $a\langle P\rangle \div C$, a trigger $a(X) \triangleright_\gamma P$, or a rollback instruction $\mathsf{roll}\ k$. We let τ and its decorated variants range over threads. We write $\prod_{i \in I} M_i$ for the parallel composition of configurations M_i for each $i \in I$ (by convention $\prod_{i \in I} M_i = \mathbf{0}$ if $I = \emptyset$), and we abbreviate $a\langle \mathbf{0}\rangle$ to $\overline{a}$.

Free identifiers and free variables. Notions of free identifiers and free variables in croll-π are standard. Constructs with binders are of the following forms: $\nu a.\, P$ binds the name a with scope P; $\nu u.\, M$ binds the identifier u with scope M; and $a(X) \triangleright_\gamma P$ binds the process variable X and the key variable γ with scope P. We denote by $\mathtt{fn}(P)$ and $\mathtt{fn}(M)$ the set of free names and keys of process P and configuration M, respectively. Note in particular that $\mathtt{fn}(k : P) = \{k\} \cup \mathtt{fn}(P)$, $\mathtt{fn}(\mathsf{roll}\ k) = \{k\}$. We say that a process P or a configuration M is *closed* if it has no free (process or key) variable. We denote by $\mathcal{P}_{cl}$ and $\mathcal{C}_{cl}$ the sets of closed processes and configurations, respectively. We abbreviate $a(X) \triangleright_\gamma P$, where X is not free in P, to $a \triangleright_\gamma P$; and $a(X) \triangleright_\gamma P$, where γ is not free in P, to $a(X) \triangleright P$.

Remark 1. We have no construct for replicated processes or internal choice in croll-π: as in the higher-order π-calculus, these can easily be encoded.

Remark 2. In the remainder of the paper, we adopt *Barendregt's Variable Convention*: if terms $t_1, \ldots, t_n$ occur in a certain context (e.g., definition, proof), then in these terms all bound identifiers and variables are chosen to be different from the free ones.

3.2 Reduction Semantics

The reduction semantics of croll-π is defined via a reduction relation $\longrightarrow$, which is a binary relation over closed configurations ($\longrightarrow \subset \mathcal{C}_{cl} \times \mathcal{C}_{cl}$), and a structural congruence relation $\equiv$, which is a binary relation over configurations ($\equiv \subset \mathcal{C} \times \mathcal{C}$). We define *configuration contexts* as "configurations with a hole $\bullet$", given by the grammar: $\mathbb{C} ::= \bullet \mid (M \mid \mathbb{C}) \mid \nu u.\, \mathbb{C}$. *General contexts* $\mathbb{G}$ are just configurations with a hole $\bullet$ in a place where an arbitrary process P can occur. A *congruence* on processes or configurations is an equivalence relation $\mathcal{R}$ that

$$\text{(E.ParC)}\ M \mid N \equiv N \mid M \qquad \text{(E.ParA)}\ M_1 \mid (M_2 \mid M_3) \equiv (M_1 \mid M_2) \mid M_3$$

$$\text{(E.NilM)}\ M \mid \mathbf{0} \equiv M \qquad \text{(E.NewN)}\ \nu u.\, \mathbf{0} \equiv \mathbf{0}$$

$$\text{(E.NewC)}\ \nu u.\, \nu v.\, M \equiv \nu v.\, \nu u.\, M \qquad \text{(E.NewP)}\ (\nu u.\, M) \mid N \equiv \nu u.\, (M \mid N)$$

$$\text{(E.}\alpha\text{)}\ M =_\alpha N \implies M \equiv N \qquad \text{(E.TagC)}\ k \prec (k_1, k_2) \equiv k \prec (k_2, k_1)$$

$$\text{(E.TagA)}\ \nu h.\, k \prec (h, k_3) \mid h \prec (k_1, k_2) \equiv \nu h.\, k \prec (k_1, h) \mid h \prec (k_2, k_3)$$

Fig. 2. Structural congruence for croll-π.

$$\text{(S.Com)}\ \frac{\mu = (k_1 : a\langle P\rangle \div C) \mid (k_2 : a(X) \rhd_\gamma Q_2)}{(k_1 : a\langle P\rangle \div C) \mid (k_2 : a(X) \rhd_\gamma Q_2) \longrightarrow \nu k.\, (k : Q_2\{^{P,k}/_{X,\gamma}\}) \mid [\mu; k]}$$

$$\text{(S.TagN)}\ k : \nu a.\, P \longrightarrow \nu a.\, k : P$$

$$\text{(S.TagP)}\ k : P \mid Q \longrightarrow \nu k_1, k_2.\, k \prec (k_1, k_2) \mid k_1 : P \mid k_2 : Q$$

$$\text{(S.Roll)}\ \frac{k <: N \qquad \texttt{complete}(N \mid [\mu; k] \mid (k_r : \mathsf{roll}\ k)) \qquad \mu' = \texttt{xtr}(\mu)}{N \mid [\mu; k] \mid (k_r : \mathsf{roll}\ k) \longrightarrow \mu' \mid N\!\not\downarrow_k}$$

$$\text{(S.Ctx)}\ \frac{M \longrightarrow N}{\mathbb{C}[M] \longrightarrow \mathbb{C}[N]} \qquad \text{(S.Eqv)}\ \frac{M \equiv M' \qquad M' \longrightarrow N' \qquad N' \equiv N}{M \longrightarrow N}$$

Fig. 3. Reduction rules for croll-π

is closed for general or configuration contexts: $P \mathcal{R} Q \implies \mathbb{G}[P] \mathcal{R} \mathbb{G}[Q]$ and $M \mathcal{R} N \implies \mathbb{C}[M] \mathcal{R} \mathbb{C}[N]$.

Structural congruence $\equiv$ is defined as the smallest congruence on configurations that satisfies the axioms in Figure 2, where $t =_\alpha t'$ denotes equality of t and t' modulo α-conversion. Axioms E.ParC to E.α are standard from the π-calculus. Axioms E.TagC and E.TagA model commutativity and associativity of connectors, in order not to have a rigid tree structure. Thanks to these two axioms we will use n-ary connectors $k \prec (h_1, \ldots, h_n)$ as a shortcut for any tree of connectors with root k and leaves $h_1, \ldots, h_n$, assuming all internal nodes are bound. Also, $\nu\tilde{u}.\, A$ stands for $\nu u_1 \ldots u_n.\, A$ if $\tilde{u} = \{u_1, \ldots, u_n\}$.

Configurations can be written in normal form using structural congruence.

Lemma 1 (Normal form). *Given a configuration M, we have:*

$$M \equiv \nu\tilde{n}.\, \prod_i (k_i : P_i) \mid \prod_j [\mu_i; k_j] \mid \prod_l k_l \prec (k'_l, k''_l)$$

The reduction relation $\longrightarrow$ is defined as the smallest binary relation on closed configurations satisfying the rules of Figure 3. This extends the naïve semantics of

roll-π introduced in [17],[7] and outlined here in Section 2, to manage alternatives. We denote by $\Longrightarrow$ the reflexive and transitive closures of $\longrightarrow$.

Reductions are either forward, given by rules S.Com, S.TagN, and S.TagP, or backward, defined by rule S.Roll. They are closed under configuration contexts (rule S.Ctx) and under structural congruence (rule S.Eqv). The rule for communication S.Com is the standard communication rule of the higher-order π-calculus with the side effects discussed in Section 2. Rule S.TagN allows restrictions in processes to be lifted at the configuration level. Rule S.TagP allows to split parallel processes. Rule S.Roll enacts rollback, canceling all the effects of the interaction identified by the unique key k, and releasing the initial configuration that gave rise to the interaction, where the alternative replaces the original message. This is the only difference between croll-π and roll-π: in the latter, the memory μ was directly released. However, this small modification yields significant changes to the expressive power of the calculus, as we will see later.

The rollback impacts only the causal descendants of k, defined as follows.

Definition 1 (Causal dependence). *Let M be a configuration and let $\mathcal{T}_M$ be the set of keys occurring in M. Causal dependence $<:_M$ is the reflexive and transitive closure of $<_M$, which is defined as the smallest binary relation on $\mathcal{T}_M$ satisfying the following clauses:*

- *$k <_M k'$ if $k \prec (k_1, k_2)$ occurs in M with $k' = k_1$ or $k' = k_2$;*
- *$k <_M k'$ if a thread $k : P$ occurs (inside μ) in a memory $[\mu; k']$ of M.*

If the configuration M is clear from the context, we write $k <: k'$ for $k <:_M k'$.

A backward reduction triggered by roll k involves *all* and *only* the descendants of key k. We ensure they are all selected by requiring that the configuration is *complete*, and that no other term is selected by requiring *k-dependence*.

Definition 2 (Complete configuration). *A configuration M is* complete*, denoted as* $\texttt{complete}(M)$*, if, for each memory $[\mu; k]$ and each connector $k' \prec (k, k_1)$ or $k' \prec (k_1, k)$ that occurs in M there exists in M either a connector $k \prec (h_1, h_2)$ or a tagged process $k : P$ (possibly inside a memory).*

A configuration M is k-dependent if all its components depend on k.

Definition 3 (k-dependence). *Let M be a configuration such that:*
$M \equiv \nu\tilde{u}.\ \prod_{i\in I}(k_i : P_i) \mid \prod_{j\in J}[\mu_j; k_j] \mid \prod_{l\in L} k_l \prec (k'_l, k''_l)$ with $k \notin \tilde{u}$.
Configuration M is k-dependent, written $k <: M$ by overloading the notation for causal dependence among keys, if for every i in $I \cup J \cup L$, we have $k <:_M k_i$.

Rollback should release all the resources consumed by the computation to be rolled-back which were provided by other threads. They are computed as follows.

[7] We extend the naïve semantics instead of the high-level or the low-level semantics (also defined in [17]) for the sake of simplicity. However, reduction semantics corresponding to the high-level and low-level semantics of roll-π can similarly be specified.

Definition 4 (Projection). *Let M be a configuration such that:* $M \equiv \nu\tilde{u}.\, \prod_{i\in I}(k_i : P_i) \mid \prod_{j\in J}[k'_j : R_j \mid k''_j : T_j; k_j] \mid \prod_{l\in L} k_l \prec (k'_l, k''_l)$ *with* $k \notin \tilde{u}$. *Then:*

$$M \not\downarrow_k = \nu\tilde{u}.\, \Big(\prod_{j'\in J'} k'_{j'} : R_{j'}\Big) \mid \Big(\prod_{j''\in J''} k''_{j''} : T_{j''}\Big)$$

where $J' = \{j \in J \mid k \not\prec: k'_j\}$ *and* $J'' = \{j \in J \mid k \not\prec: k''_j\}$.

Intuitively, $M \not\downarrow_k$ consists of the threads inside memories in M which are not dependent on k.

Finally, and this is the main novelty of croll-π, function `xtr` defined below replaces messages from the memory targeted by the roll by their alternatives.

Definition 5 (Extraction function).

$$\mathtt{xtr}(M \mid N) = \mathtt{xtr}(M) \mid \mathtt{xtr}(N) \qquad \mathtt{xtr}(k : a\langle P\rangle \div C) = k : C$$
$$\mathtt{xtr}(k : a(X) \rhd_\gamma Q) = k : a(X) \rhd_\gamma Q$$

No other case needs to be taken into account as `xtr` is only called on the contents of memories.

Remark 3. Not all syntactically licit configurations make sense. In particular, we expect configurations to respect the causal information required for executing croll-π programs. We therefore work only with *coherent* configurations. A configuration is coherent if it is obtained by reduction starting from a configuration of the form $\nu k.\, k : P$ where P is closed and contains no roll h primitive (all the roll primitives should be of the form roll γ).

3.3 Barbed Congruence

We define notions of strong and weak barbed congruence to reason on croll-π processes and configurations. Name a is *observable* in configuration M, denoted as $M\downarrow_a$, if $M \equiv \nu\tilde{u}.\, (k : a\langle P\rangle \div C) \mid N$, with $a \notin \tilde{u}$. We write $M\mathcal{R}\downarrow_a$, where $\mathcal{R}$ is a binary relation on configurations, if there exists N such that $M\mathcal{R}N$ and $N\downarrow_a$. The following definitions are classical.

Definition 6 (Barbed congruences for configurations). *A relation $\mathcal{R} \subseteq \mathcal{C}_{cl} \times \mathcal{C}_{cl}$ on closed configurations is a* strong *(respectively* weak*)* barbed simulation *if whenever $M\,\mathcal{R}\,N$,*

- $M\downarrow_a$ *implies* $N\downarrow_a$ *(respectively* $N \Longrightarrow\downarrow_a$*);*
- $M \longrightarrow M'$ *implies* $N \longrightarrow N'$ *(respectively* $N \Longrightarrow N'$*) with* $M'\mathcal{R}N'$.

A relation $\mathcal{R} \subseteq \mathcal{C}_{cl} \times \mathcal{C}_{cl}$ is a strong (weak) barbed bisimulation *if $\mathcal{R}$ and $\mathcal{R}^{-1}$ are strong (weak) barbed simulations. We call* strong (weak) barbed bisimilarity *and denote by $\sim$ ($\approx$) the largest strong (weak) barbed bisimulation. The largest congruence for configuration contexts included in $\sim$ ($\approx$) is called* strong (weak) barbed congruence*, denoted by $\sim_c$ ($\approx_c$).*

The notion of strong and weak barbed congruence extends to closed and open processes, by considering general contexts that form closed configurations.

Definition 7 (Barbed congruences for processes). *A relation $\mathcal{R} \subseteq \mathcal{P}_{cl} \times \mathcal{P}_{cl}$ on closed processes is a* strong *(resp.* weak*) barbed congruence if whenever $P\mathcal{R}Q$, for all general contexts $\mathbb{G}$ such that $\mathbb{G}[P]$ and $\mathbb{G}[Q]$ are closed configurations, we have $\mathbb{G}[P] \sim_c \mathbb{G}[Q]$ (resp. $\mathbb{G}[P] \approx_c \mathbb{G}[Q]$).*

Two open processes P and Q are said to be strong (resp. weak) barbed congruent, denoted by $P \sim_c^o Q$ (resp. $P \approx_c^o Q$) if for all substitutions σ such that $P\sigma$ and $Q\sigma$ are closed, we have $P\sigma \sim_c Q\sigma$ (resp. $P\sigma \approx_c Q\sigma$).

Working with arbitrary contexts can quickly become unwieldy. We offer the following Context Lemma to simplify the proofs of congruence.

Theorem 1 (Context Lemma). *Two processes P and Q are weak barbed congruent, $P \approx_c^o Q$, if and only if for all substitutions σ such that $P\sigma$ and $Q\sigma$ are closed, all closed configurations M, and all keys k, we have: $M \mid (k : P\sigma) \approx M \mid (k : Q\sigma)$.*

The proof of this Context Lemma is much more involved than the corresponding one in the π-calculus, notably because of the bookkeeping required in dealing with process and thread tags. It is obtained by composing the lemmas below.

The first lemma shows that the only relevant configuration contexts are parallel contexts.

Lemma 2 (Context Lemma for closed configurations). *For any closed configurations M, N, $M \sim_c N$ if and only if, for all closed configurations L, $M \mid L \sim N \mid L$. Likewise, $M \approx_c N$ if and only if, for all L, $M \mid L \approx N \mid L$.*

Proof. The left to right implication is immediate, by definition of $\sim_c$. For the other direction, the proof consists in showing that $\mathcal{R} = \{\langle \mathbb{C}[M], \mathbb{C}[N] \rangle \mid \forall L, M \mid L \sim N \mid L\}$ is included in $\sim$. The weak case is identical to the strong one. □

We can then prove the thesis on closed processes.

Lemma 3 (Context lemma for closed processes). *Let P and Q be closed processes. We have $P \approx_c Q$ if and only if, for all closed configuration contexts $\mathbb{C}$ and $k \notin \mathtt{fn}(P, Q)$, we have $\mathbb{C}[k : P] \approx \mathbb{C}[k : Q]$.*

Proof. The left to right implication is clear. One can prove the right to left direction by induction on the form of general contexts for processes. □

We then deal with open processes.

Lemma 4 (Context lemma for open processes). *Let P and Q be (possibly open) processes. We have $P \approx_c^o Q$ if and only if for all closed configuration contexts $\mathbb{C}$, all substitutions σ such that $P\sigma$ and $Q\sigma$ are closed, and all $k \notin \mathtt{fn}(P, Q)$, we have $\mathbb{C}[k : P\sigma] \approx \mathbb{C}[k : Q\sigma]$.*

$$
\begin{aligned}
\mathcal{R} &= \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \cup \mathcal{R}_5 \cup Id \\
\mathcal{R}_1 &= \{\langle k : a\langle P\rangle \div Q \mid L \,;\, k : (\nu c.\, a\langle P\rangle \div c\langle Q\rangle \div \mathbf{0} \mid c(X) \triangleright X) \mid L\rangle\} \\
\mathcal{R}_2 &= \{\langle k : a\langle P\rangle \div Q \mid L \,;\, \nu c, k_1, k_2.\, k \prec (k_1, k_2) \mid k_1 : a\langle P\rangle \div c\langle Q\rangle \div \mathbf{0} \mid k_2 : c(X) \triangleright X \mid L\rangle\} \\
\mathcal{R}_3 &= \{\langle \nu h.\, [k : a\langle P\rangle \div Q \mid k' : a(X) \triangleright_\gamma R; h] \mid L'' \,; \\
&\qquad \nu c, k_1, k_2, h.\, k \prec (k_1, k_2) \mid [k_1 : a\langle P\rangle \div c\langle Q\rangle \div \mathbf{0} \mid k' : a(X) \triangleright_\gamma R; h] \mid k_2 : c(X) \triangleright X \mid L''\rangle\} \\
\mathcal{R}_4 &= \{\langle k : Q \mid L''' \,;\, \nu c, k_1, k_2.\, k \prec (k_1, k_2) \mid k_1 : c\langle Q\rangle \div \mathbf{0} \mid k_2 : c(X) \triangleright X \mid L'''\rangle\} \\
\mathcal{R}_5 &= \{\langle k : Q \mid L''' \,;\, \nu c, k_1, k_2, h.\, k \prec (k_1, k_2) \mid [k_1 : c\langle Q\rangle \div \mathbf{0} \mid k_2 : c(X) \triangleright X; h] \mid h : Q \mid L'''\rangle\}
\end{aligned}
$$

Fig. 4. Bisimulation relation for arbitrary alternatives.

Proof. For the only if part, one proceeds by induction on the number of bindings in σ. The case for zero bindings follows from Lemma 3. For the inductive case, we write $\mathbb{P}[\bullet]$ for a process where an occurrence of $\mathbf{0}$ has been replaced by $\bullet$, and we show that contexts of the form $\mathbb{P} = a\langle R\rangle \mid a(X) \triangleright \mathbb{P}'[\bullet]$ where a is fresh and $\mathbb{P} = a\langle R\rangle \mid a(X) \triangleright_\gamma \mathbb{P}'[\bullet]$ where a is fresh and X never occurs in the continuation actually enforce the desired binding.

For the if part, the proof is by induction on the number of triggers. If the number of triggers is 0 then the thesis follows from Lemma 3. The inductive case consists in showing that equivalence under substitutions ensures equivalence under a trigger context. □

Proof (of Theorem 1). A direct consequence of Lemma 4 and Lemma 2. □

4 croll-π expressiveness

4.1 Alternative idioms

The message with alternative $a\langle P\rangle \div C$ triggers alternative C upon rollback. We choose to restrict C to be either a message with $\mathbf{0}$ alternative or $\mathbf{0}$ itself in order to have a minimal extension of roll-π. However, this simple form of alternative is enough to encode far more complex alternative policies and constructs, as shown below. We define the semantics of the alternative idioms below by changing function `xtr` in Definition 5. We then encode them in croll-π and prove the encoding correct w.r.t. weak barbed congruence. Since we consider extensions of croll-π, in weak barbed congruence we consider just closure under croll-π contexts. By showing that the extensions have the same expressive power of croll-π, we ensure that allowing them in contexts would not change the result. Every encoding maps unmentioned constructs homomorphically to themselves. After having defined each alternative idiom, we freely use it as an abbreviation.

Arbitrary alternatives. Messages with arbitrary alternative can be defined by allowing C to be any process Q. No changes are required to the definition of

function $\mathtt{xtr}$. We can encode arbitrary alternatives as follows, where c is not free in P, Q.

$$(\!| a\langle P\rangle \div Q |\!)_{aa} = \nu c.\, a\langle (\!| P |\!)_{aa}\rangle \div c\langle (\!| Q |\!)_{aa}\rangle \div \mathbf{0} \mid c(X) \triangleright X$$

Proposition 1. *$P \approx_c (\!| P |\!)_{aa}$ for any closed process with arbitrary alternatives.*

Proof. We consider just one instance of arbitrary alternative, the thesis will follow by transitivity.

Thanks to Lemma 4 and Lemma 2, we only need to prove that for all closed configurations L and $k \notin \mathtt{fn}(P)$, we have $k : a\langle P\rangle \div Q \mid L \approx k : (\nu c.\, a\langle P\rangle \div c\langle Q\rangle \div \mathbf{0} \mid c(X) \triangleright X) \mid L$. We consider the relation $\mathcal{R}$ in Figure 4 and prove that it is a weak barbed bisimulation. In every relation, L is closed and $k \notin \mathtt{fn}(P)$.

In $\mathcal{R}_1$, the right configuration can reduce via rule S.TagN followed by S.TagP. These lead to $\mathcal{R}_2$. Performing these reductions is needed to match the barb and the relevant reductions of the left configuration, thus we consider directly $\mathcal{R}_2$. In $\mathcal{R}_2$ the barbs coincide. Rollbacks lead to the identity. The only possible communication is on a, and requires $L \equiv L' \mid k' : a(X) \triangleright_\gamma R$. It leads to $\mathcal{R}_3$, where $L'' = L' \mid R\{^{P,h}/_{X,\gamma}\}$. In $\mathcal{R}_3$ the barbs coincide too. All the reductions can be matched by staying in $\mathcal{R}_3$ or going to the identity, but for executing a roll with key h. This leads to $\mathcal{R}_4$. From $\mathcal{R}_4$ we can always execute the internal communication at c leading to $\mathcal{R}_5$. The thesis follows from the result below, whose proof requires again to find a suitable bisimulation relation.

Lemma 5. *For each configuration M k-dependent and complete such that $k', t, k_1, k_2 \notin \mathtt{fn}(M)$ we have $M \approx_c \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid [k_1 : t\langle Q\rangle \div C \mid k_2 : t(X) \triangleright R; k'] \mid M\{^{k'}/_{k}\}$.*

□

Proofs concerning other idioms follow similar lines, and can be found in the online technical report [16].

A particular case of arbitrary alternative $a\langle P\rangle \div Q$ is when Q is a message whose alternative is not $\mathbf{0}$. By applying this pattern recursively we can write $a_1\langle P_1\rangle \div \ldots \div a_n\langle P_n\rangle \div Q$. In particular, by choosing $a_1 = \cdots = a_n$ and $P_1 = \cdots = P_n$ we can try n times the alternative P before giving up by executing Q.

Endless retry. We can also retry the same alternative infinitely many times, thus obtaining the behavior of roll-π messages.

$$(\!| a\langle P\rangle |\!)_{er} = \nu t.\, Y \mid a\langle (\!| P |\!)_{er}\rangle \div t\langle Y\rangle \qquad Y = t(Z) \triangleright Z \mid a\langle (\!| P |\!)_{er}\rangle \div t\langle Z\rangle$$

Proposition 2. *$P \approx_c (\!| P |\!)_{er}$ for any closed process with roll-π messages.*

As corollary of Proposition 2 we thus have the following.

Corollary 1. *croll-π is a conservative extension of roll-π.*

Triggers with alternative. Until now we attached alternatives to messages. Symmetrically, one may attach alternatives to triggers. Thus, upon rollback, the message is released and the trigger is replaced by a new process.

The syntax for triggers with alternative is $a(X) \triangleright_\gamma Q \div b\langle Q' \rangle \div \mathbf{0}$, with X not free in Q'. As for messages, we use a single message as alternative, but one can use general processes as described earlier. Triggers with alternative are defined by the extract clause below.

$$\mathtt{xtr}(k : a(X) \triangleright_\gamma Q \div b\langle Q' \rangle \div \mathbf{0}) = k : b\langle Q' \rangle \div \mathbf{0}$$

Interestingly, messages with alternative and triggers with alternative may coexist. The encoding of triggers with alternative is as follows.

$$(\!| a(X) \triangleright_\gamma Q \div b\langle Q' \rangle \div \mathbf{0} |\!)_{at} = \nu c, d.\, \overline{c} \div \overline{d} \div \mathbf{0} \mid (c \triangleright_\gamma a(X) \triangleright (\!| Q |\!)_{at}) \mid (d \triangleright b\langle (\!| Q' |\!)_{at} \rangle \div \mathbf{0})$$

Proposition 3. *$P \approx_c (\!| P |\!)_{at}$ for any closed process with triggers with alternative.*

4.2 Comparing croll-π and roll-π

While Corollary 1 shows that croll-π is at least as expressive as roll-π, a natural question is whether croll-π is actually strictly more expressive than roll-π or not. The theorem below gives a positive answer to this question.

Theorem 2. *There is no encoding $(\!| \bullet |\!)$ from croll-π to roll-π such that for each croll-π configuration M:*

1. *if M has a computation including at least a backward step, then $(\!| M |\!)$ has a computation including at least a backward step;*
2. *if M has only finite computations, then $(\!| M |\!)$ has only finite computations.*

Proof. Consider configuration $M = \nu k.\, k : \overline{a} \div \overline{b} \div \mathbf{0} \mid a \triangleright_\gamma \mathsf{roll}\ \gamma$. This configuration has a unique possible computation, composed by one forward step followed by one backward step. Assume towards a contradiction that an encoding exists and consider $(\!| M |\!)$. $(\!| M |\!)$ should have at least a computation including a backward step. From roll-π loop lemma, if we have a backward step, we are able to go forward again, and then there is a looping computation. This is in contrast with the second condition of the encoding. The thesis follows. □

The main point behind this result is that the Loop Lemma, a cornerstone of roll-π theory [17] capturing the essence of rigid rollback (and similar results in [8, 18, 20, 22]), does not hold in croll-π. Naturally, the result above does not imply that croll-π cannot be encoded in HOπ or in π-calculus. However, these calculi are too low level for us, as hinted by the fact that the encoding of a simple reversible higher order calculus into HOπ is quite complex, as shown in [18].

5 Programming in croll-π

A main goal of croll-π is to make reversibility techniques exploitable for application development. Even if croll-π is not yet a full-fledged language, we have developed a proof-of-concept interpreter for it. To the best of our knowledge, this is the first interpreter for a causal-consistent reversible language. We then put the implementation at work on a few simple, yet interesting, programming problems. We detail below the algorithm we devised to solve the Eight Queens problem [3, p. 165][8].

The interpreter for croll-π is written in Maude [10], a language based on both equational and rewriting logic that allows the programmer to define terms and reduction rules, e.g., to execute reduction semantics of process calculi. Most of croll-π's rules are straightforwardly interpreted, with the exception of rule S.Roll. This rule is quite complex as it involves checks on an unbounded number of interacting components. Such an issue is already present in roll-π [17], where it is addressed by providing an easier to implement, yet equivalent, low-level semantics. This semantics replaces rule S.Roll with a protocol that sends notifications to all the involved components to roll-back, then waits for them to do so. Extending the low-level semantics from roll-π to croll-π simply requires the application of function `xtr` to the memory targeted by the rollback. We do not detail the low-level semantics of croll-π here, and refer the reader to [17] for a detailed description in the setting of roll-π. Our Maude interpreter is based on this low-level semantics, extended with values (integers and pairs) and with the `if-then-else` construct. It is fairly concise (less than 350 lines of code).

The Eight Queens problem can be formulated as follows: how to place 8 queens on an 8×8 chess board such that no queen can directly capture another? We have chosen this problem since its solution involves a state-space exploration and requires frequent backtrack. We defined an algorithm in croll-π where all queens are autonomous entities, numbered from 1 to 8, all executing the code schema shown in Figure 5. We use $\mathbf{x}$ to indicate a pair of variables (x_1, x_2), and replicated messages $!c_i\langle\mathbf{x}\rangle \div \mathbf{0}$ to denote the encoding of a parallel composition of an infinite number of messages $c_i\langle\mathbf{x}\rangle \div \mathbf{0}$ (cf. Remark 1). The queens are activated in numeric order. The i-th queen is activated by messages on channels c_j from its predecessors, instantiating variables $\mathbf{x_j}$ with their position. When a queen is activated it looks for its position by trying sequentially all the positions in the i-th row of the chess board. To try a position, it sends it over channel p_i and then verifies whether there is a conflict or not by computing $err(\mathbf{x_j}, \mathbf{x})$ for each $j < i$. In case of conflict, it rolls-back the choice of the position (with roll γ_i) and tries the next position. If no suitable position is available, the choice of position of the previous queen is rolled-back (possibly recursively) by the communication over f_i. If instead there is no conflict, the queen commits its position on c_i, thus activating the next queen, and waits for potential rollback requests on f_{i+1}.

[8] The interpreter, the code for solving the Eight Queens problem, and other examples are available at `http://proton.inrialpes.fr/~mlienhar/croll-pi/implem`.

$$
\begin{aligned}
Q_i \;\triangleq\;& c_1(\mathbf{x_1}) \triangleright \ldots c_{i-1}(\mathbf{x_{i-1}}) \triangleright\; p_i\langle i,1\rangle \div \ldots \div p_i\langle i,8\rangle \div f_i\langle 0\rangle \div 0 \\
& \mid p_i(\mathbf{x}) \triangleright_{\gamma_i} \texttt{if}\ err(\mathbf{x_1},\mathbf{x})\ \texttt{then}\ \mathsf{roll}\ \gamma_i\ \texttt{else}\ \ldots\ \texttt{if}\ err(\mathbf{x_{i-1}},\mathbf{x})\ \texttt{then}\ \mathsf{roll}\ \gamma_i \\
& \texttt{else}\ !c_i\langle \mathbf{x}\rangle \div 0 \mid f_{i+1}(y) \triangleright \mathsf{roll}\ \gamma_i \\
err((x_1,x_2),(y_1,y_2)) \;\triangleq\;& (x_1 = y_1 \vee x_2 = y_2 \vee |x_1 - y_1| = |x_2 - y_2|)
\end{aligned}
$$

Fig. 5. The i-th queen

On a recent laptop, the interpreter returns almost immediately with the first solution of the problem, namely the set of eight pairs (1,1) (2,5) (3,8) (4,6) (5,3) (6,7) (7,2) (8,4).

6 Asynchronous Interacting Transactions

This section shows how croll-π can model in a precise way interacting transactions with compensations as formalized in TransCCS [13]. Actually, the natural croll-π encoding improves on the semantics in [13], since croll-π causality tracking is more precise than the one in TransCCS, which is based on dynamic embedding of processes into transactions. Thus croll-π avoids some spurious undo of actions, as described below. Before entering the details of TransCCS, let us describe the general idea.

We consider a very general notion of atomic (but not necessarily isolated) transaction, i.e., a process that executes completely or not at all. Informally, a transaction $[P,Q]_\gamma$ with name γ executing process P with compensation Q can be modeled by a process of the form:

$$[P,Q]_\gamma = \nu a, c.\, \overline{a} \div \overline{c} \div \mathbf{0} \mid (a \triangleright_\gamma P) \mid (c \triangleright Q)$$

Intuitively, when $[P,Q]_\gamma$ is executed, it first starts process P under the rollback scope γ. Abortion of the transaction can be triggered in P by executing a roll γ. Whenever P is rolled-back, the rollback does not restart P (since the message on a is substituted by the alternative on c), but instead starts the compensation process Q. In this approach commit is implicit: when there is no reachable roll γ, the transaction is committed. From the explanation above, it should be clear that in the execution of $[P,Q]_\gamma$, either P executes completely, i.e., until it reaches a commit, or not at all, in the sense that it is perfectly rolled-back. If P is ever rolled-back, its failed execution can be compensated by that of process Q. Interestingly, and in contrast with irreversible actions used in [12], our rollback scopes can be nested without compromising this all-or-nothing semantics.

Let us now consider an asynchronous fragment of TransCCS, the calculus in [13], removing choice and recursion. Dealing with the whole calculus would not add new difficulties related to rollback, but only related to the encoding of such operators in higher-order π. The syntax of the fragment of TransCCS we consider is as follows.

$$P ::= \mathbf{0} \mid \nu a.\, P \mid (P \mid Q) \mid \overline{a} \mid a.P \mid \mathsf{co}\ k \mid [\![P \triangleright_k Q]\!]$$

$$\text{(R-Comm)}\quad \overline{a} \mid a.P \longrightarrow P \qquad\qquad \text{(R-Emb)}\quad \frac{k \notin \mathtt{fn}(R)}{[\![P \triangleright_k Q]\!] \mid R \longrightarrow [\![P \mid R \triangleright_k Q \mid R]\!]}$$

$$\text{(R-Co)}\quad [\![P \mid \mathsf{co}\ k \triangleright_k Q]\!] \longrightarrow P \qquad\qquad \text{(R-Ab)}\quad [\![P \triangleright_k Q]\!] \longrightarrow Q$$

and is closed under active contexts $\nu a.\,\bullet$, $\bullet \mid Q$ and $[\![\bullet \triangleright_k Q]\!]$, and structural congruence.

Fig. 6. Reduction rules for TransCCS

Essentially, it extends CCS with a transactional construct $[\![P \triangleright_k Q]\!]$, executing a transaction with body P, name k and compensation Q, and a commit operator $\mathsf{co}\ k$.

The rules defining the semantics of TransCCS are given in Figure 6. Structural congruence contains the usual rules for parallel composition and restriction. Keep in mind that transaction scope is a binder for its name k, thus k does not occur outside the transaction, and there is no name capture in rules R-Co and R-Emb.

A croll-π transaction $[P, Q]_\gamma$ as above has explicit abort, specified by $\mathsf{roll}\ \gamma$, where γ is used as the transaction name, and implicit commit. TransCCS takes different design choices, using non-deterministic abort and programmable commit. Thus we have to instantiate the encoding above.

Definition 8 (TransCCS encoding). *Let P be a TransCCS process. Its encoding $(\!|\bullet|\!)_t$ in croll-π is defined as:*

$$(\!|\nu a.\,P|\!)_t = \nu a.\,(\!|P|\!)_t \qquad (\!|P \mid Q|\!)_t = (\!|P|\!)_t \mid (\!|Q|\!)_t \qquad (\!|\overline{a}|\!)_t = \overline{a}$$
$$(\!|a.P|\!)_t = a \triangleright (\!|P|\!)_t \qquad (\!|\mathsf{co}\ l|\!)_t = l(X) \triangleright \mathbf{0} \qquad (\!|\mathbf{0}|\!)_t = \mathbf{0}$$

$$(\!|[\![P \triangleright_l Q]\!]|\!)_t = [\nu l.\,(\!|P|\!)_t \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \triangleright X, (\!|Q|\!)_t]_\gamma$$

Since in croll-π only configurations can execute, the behavior of P should be compared with $\nu k.\,k : (\!|P|\!)_t$.

In the encoding, abort is always possible since at any time the only occurrence of the roll in the transaction can be activated by a communication on l. On the other hand, executing the encoding of a TransCCS commit disables the roll related to the transaction. This allows to garbage collect the compensation, and thus corresponds to an actual commit. Note, however, that in croll-π the abort operation is not atomic as in TransCCS since the roll related to a transaction first has to be enabled through a communication on l, disabling in this way any possibility to commit, and then it can be executed. Clearly, until the roll is executed, the body of the transaction can continue its execution. To make abort atomic one would need the ability to disable an active roll, as could be done using a (mixed) choice such as $(\mathsf{roll}\ k) + (l \triangleright \mathbf{0})$. In this setting an output on l would commit the transaction. Adding choice would not make the reduction semantics more difficult, but its impact on behavioral equivalence has not been studied yet.

The relation between the behavior of a TransCCS process P and of its translation $(\!|P|\!)_t$ is not immediate, not only because of the comment above on atomicity, but also because of the approximate tracking of causality provided by TransCCS. TransCCS tracks interacting processes using rule (R-Emb): only processes inside the same transaction may interact, and when a process enters the transaction it is saved in the compensation, so that it can be restored in case of abort. However, no check is performed to ensure that the process actually interacts with the transaction code. For instance, a process $\overline{a} \mid a.P$ may enter a transaction $[\![Q \triangleright_k R]\!]$ and then perform the communication at a. Such a communication would be undone in case of abort. This is a spurious undo, since the communication at a is not related to the transaction code. Actually, the same communication could have been performed outside the transaction, and in this case it would not have been undone.

In croll-π encoding, a process is "inside" the transaction with key k if and only if its tag is causally dependent on k. Thus a process enters a transaction only by interacting with a process inside it. For this reason, there is no reduction in croll-π corresponding to rule (R-Emb), and since no process inside the transaction is involved in the reduction at a above, the reduction would not be undone in case of abort, since it actually happens "outside" the transaction. Thus our encoding avoids spurious undo, and computations in croll-π correspond to computations in TransCCS with minimal applications of rule (R-Emb). These computations are however very difficult to characterize because of syntactic constraints. In fact, for two processes inside two parallel transactions k_1 and k_2 to interact, either k_1 should move inside k_2 or vice versa, but in both the cases not only the interacting processes move, as minimality would require, but also all the other processes inside the same transactions have to move. Intuitively, TransCCS approximates the causality relation, which is a dag, using the tree defined by containment. The spurious reductions undone in TransCCS can always be redone so to reach a state corresponding to the croll-π one. In this sense croll-π minimizes the set of interactions undone.

We define a notion of weak barbed bisimilarity ${}_t\!\approx_{c\pi}$ relating a TransCCS process P and a croll-π configuration M. First, we define barbs in TransCCS by the predicate $P{\downarrow_a}$, which is true in the cases below, false otherwise.

$$\begin{array}{ll} \overline{a}{\downarrow_a} & \nu b.\, P{\downarrow_a} \ \text{ if } P{\downarrow_a} \wedge a \neq b \\ P \mid P'{\downarrow_a} \ \text{ if } P{\downarrow_a} \vee P'{\downarrow_a} & [\![P \triangleright_k Q]\!]{\downarrow_a} \ \text{ if } P{\downarrow_a} \wedge a \neq k \end{array}$$

Here, differently from [13], we observe barbs inside the transaction body, to have a natural correspondence with croll-π barbs.

Definition 9. *A relation $\mathcal{R}$ relating TransCCS processes P and croll-π configurations M is a weak barbed bisimulation if and only if for each $(P, M) \in \mathcal{R}$:*

1. *if $P{\downarrow_a}$ then $M \Longrightarrow{\downarrow_a}$;*
2. *if $M{\downarrow_a}$ then $P \Longrightarrow{\downarrow_a}$;*
3. *if $P \longrightarrow P_1$ is derived using rule* (R-Ab) *then $M \Longrightarrow M'$, $P_1 \Longrightarrow P_2$ and $P_2 \mathcal{R} M'$;*

4. *if* $P \longrightarrow P_1$ *is derived without using rule* (R-Ab) *then* $M \Longrightarrow M'$ *and* $P_1 \mathcal{R} M'$*;*
5. *if* $M \longrightarrow M'$ *then either: (i)* $P \mathcal{R} M'$ *or (ii)* $P \longrightarrow P_1$ *and* $P_1 \mathcal{R} M'$ *or (iii)* $M' \longrightarrow M''$, $P \longrightarrow P_1$ *and* $P_1 \mathcal{R} M''$.

Weak barbed bisimilarity ${}_t\!\approx_{c\pi}$ *is the largest weak barbed bisimulation.*

The main peculiarities of the definition above are in condition 3, which captures the need of redoing some reductions that are unduly rolled-back in TransCCS, and in case (iii) of condition 5, which forces atomic abort.

Theorem 3. *For each TransCCS process* P, $P \; {}_t\!\approx_{c\pi} \; \nu k.\, k : (\!| P |\!)_t$.

Proof. The proof has to take into account the fact that different croll-π configurations may correspond to the same TransCCS process. In particular, a TransCCS transaction $[\![P \triangleright_k Q]\!]$ is matched in different ways if Q is the original compensation or if part of it is the result of an application of rule (R-Emb).

Thus, in the proof, we give a syntactic characterization of the set of croll-π configurations $(\!| P |\!)^p$ matching a TransCCS process P. Then we show that $\nu k.\, k : (\!| P |\!)_t \in (\!| P |\!)^p$, and that there is a match between reductions of P and the weak reductions of each configuration in $(\!| P |\!)^p$. The proof, in the two directions, is by induction on the rule applied to derive a single step. □

7 Related work and conclusion

We have presented a concurrent process calculus with explicit rollback and minimal facilities for alternatives built on a reversible substrate analogous to a Lévy labeling [4] for concurrent computations. We have shown by way of examples how to build more complex alternative idioms and how to use rollback and alternatives in conjunction to encode transactional constructs. In particular, we have developed an analysis of communicating transactions proposed in TransCCS [13]. We also developed a proof-of-concept interpreter of our language and used it to solve the Eight Queens problem.

Undo or rollback capabilities in sequential languages have a long history (see [19] for an early survey). In a concurrent setting, interest has developed more recently. Works such as [9] introduce logging and process group primitives as a basis for defining fault-tolerant abstractions, including transactions. Ziarek et al. [25] introduce a checkpoint abstraction for concurrent ML programs. Field et al. [15] extend the actor model with checkpointing constructs. Most of the approaches relying instead on a fully reversible concurrent language have already been discussed in the introduction. Here we just recall that models of reversible computation have also been studied in the context of computational biology, e.g., [8]. Also, the effect of reversibility on Hennessy-Milner logic has been studied in [23]. Several recent works have proposed a formal analysis of transactions, including [13] studied in this paper, as well as several other works such as [21, 5, 7] (see [1] for numerous references to the line of work concentrating on software transactional memories). Note that although reversible calculi can be used to

implement transactions, they offer more flexibility. For instance, transactional events [14] only allow an all or nothing execution of transactions. Moreover, no visible side-effect is allowed during the transaction, as there is no way to specify how to compensate the side-effects of a failed transaction. A reversible calculus with alternatives allows the encoding of such compensations.

With the exception of the seminal work by Danos and Krivine [12] on RCCS, we are not aware of other work exploiting precise causal information as provided by our reversible machinery to analyze recovery-oriented constructs. Yet this precision seems important: as we have seen in Section 6, it allows us to weed out spurious undo of actions that appear in an approach that relies on a cruder transaction "embedding" mechanism. Although we have not developed a formal analysis yet, it seems this precision would be equally important, e.g., to avoid uncontrolled cascading rollbacks (domino effect) in [25] or to ensure that, in contrast to [15], rollback is always possible in failure-free computations. Although [9] introduces primitives able to track down causality information among groups of processes, called conclaves, it does not provide automatic support for undoing the effects of aborted conclaves, while our calculus directly provides a primitive to undo all the effects of a communication.

While encouraging, our results in Section 6 are only preliminary. Our concurrent rollback and minimal facilities for alternatives provide a good basis for understanding the "all-or-nothing" property of transactions. To this end it would be interesting to understand whether we are able to support both strong and weak atomicity of [21]. How to support isolation properties found, e.g., in software transactional memory models, in a way that combines well with these facilities remains to be seen. Further, we would like to study the exact relationships that exist between these facilities and the different notions of compensation that have appeared in formal models of computation for service-oriented computing, such as [5, 7]. It is also interesting to compare with zero-safe Petri nets [6], since tokens in zero places dynamically define transaction scopes as done by communications in croll-π.

From a practical point of view, we want both to refine the interpreter, and to test it against a wider range of more complex case studies. Concerning the interpreter, a main point is to allow for garbage collection of memories which cannot be restored any more, so to improve space efficiency.

References

[1] M. Abadi and T. Harris. Perspectives on transactional memory. In *CONCUR'09*, volume 5710 of *LNCS*. Springer, 2009.

[2] G. Bacci, V. Danos, and O. Kammar. On the statistical thermodynamics of reversible communicating processes. In *CALCO 2011*, volume 6859 of *LNCS*, 2011.

[3] W. W. Rouse Ball. *Mathematical Recreations and Essays (12th ed.)*. Macmillan, New York, 1947.

[4] G. Berry and J.-J. Lévy. Minimal and optimal computations of recursive programs. *J. ACM*, 26(1), 1979.

[5] R. Bruni, H. C. Melgratti, and U. Montanari. Theoretical foundations for compensations in flow composition languages. In *POPL'05*. ACM, 2005.
[6] R. Bruni and U. Montanari. Zero-safe nets: Comparing the collective and individual token approaches. *Information and Computation*, 156(1-2), 2000.
[7] M. J. Butler, C.A.R. Hoare, and C. Ferreira. A trace semantics for long-running transactions. In *25 Years CSP*, number 3525 in LNCS. Springer, 2004.
[8] L. Cardelli and C. Laneve. Reversible structures. In *CMSB 2011*. ACM, 2011.
[9] T. Chothia and D. Duggan. Abstractions for fault-tolerant global computing. *Theor. Comput. Sci.*, 322(3), 2004.
[10] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J.F. Quesada. Maude: specification and programming in rewriting logic. *Theor. Comp. Sci.*, 285(2), 2002.
[11] V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR'04*, volume 3170 of *LNCS*. Springer, 2004.
[12] V. Danos and J. Krivine. Transactions in RCCS. In *CONCUR'05*, volume 3653 of *LNCS*. Springer, 2005.
[13] E. de Vries, V. Koutavas, and M. Hennessy. Communicating transactions. In *CONCUR 2010*, volume 6269 of *LNCS*. Springer, 2010.
[14] K. Donnelly and M. Fluet. Transactional events. *Journal of Functional Programming*, 18(5–6), 2008.
[15] J. Field and C.A. Varela. Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments. In *POPL'05*. ACM, 2005.
[16] I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. Concurrent flexible reversibility (TR). `http://www.cs.unibo.it/~lanese/publications/fulltext/TR-crollpi.pdf.gz`, 2012.
[17] I. Lanese, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. Controlling reversibility in higher-order pi. In *CONCUR 2011*, volume 6901 of *LNCS*. Springer, 2011.
[18] I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order pi. In *CONCUR 2010*, volume 6269 of *LNCS*. Springer, 2010.
[19] G.B. Leeman. A formal approach to undo operations in programming languages. *ACM Trans. Program. Lang. Syst.*, 8(1), 1986.
[20] M. Lienhardt, I. Lanese, C. A. Mezzina, and J.-B. Stefani. A reversible abstract machine and its space overhead. In *FMOODS/FORTE 2012*, volume 7273 of *LNCS*, 2012.
[21] K. F. Moore and D. Grossman. High-level small-step operational semantics for transactions. In *POPL'08*. ACM, 2008.
[22] I. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2), 2007.
[23] I. Phillips and I. Ulidowski. A logic with reverse modalities for history-preserving bisimulations. In *EXPRESS 2011*, volume 64 of *EPTCS*, 2011.
[24] D. Sangiorgi and D. Walker. *The π-calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
[25] L. Ziarek and S. Jagannathan. Lightweight checkpointing for concurrent ML. *J. Funct. Program.*, 20(2), 2010.

Appendix

Additional material

A Appendix outline

We gather in this appendix full proofs of the main results of the paper, as well as additional material needed to support the proofs. We start below by a number of definitions that we have not included in the main text for lack of space. We then proceed to present a number of useful lemmas (Section B). We finally present the proofs of the main results of the paper, section by section.

Coherent configurations. Not all syntactically licit configurations make sense. In particular, we expect configurations to respect the causal information required for executing croll-π programs. We therefore work only with *coherent* closed configurations. In particular, when working with configuration contexts $\mathbb{C}$, we are only considering coherent closed configurations $\mathbb{C}[M]$. A configuration

$$M \equiv \nu\tilde{u}.\, \prod_{i\in I} k_i : P_i \mid \prod_{j\in J} [\mu_j : k'_j] \mid \prod_{z\in Z} k^1_z \prec (k^2_z, k^3_z)$$

where $\mu_j = h'_j : a_j\langle P_j\rangle \div C \mid h''_j : a_j(X) \rhd_\gamma Q_j$, is called *coherent* if the following clauses are verified:

1. all tags k_i, h'_j, $h''_j$and k^1_z are distinct;
2. the relation $<:_M$ among tags is a partial order;
3. for each roll h_i in P_i, $h_i <: k_i$.

Working only with coherent configurations is licit thanks to the following lemma.

Lemma 6. *Assume M is coherent. If $M \longrightarrow M'$ then M' is coherent.*

Proof. By case analysis on the reduction rules. □

General contexts. We did not detail in the main text the exact form of general contexts. We do so now. We define four notions of contexts: *process contexts*, *thread contexts*, *memory contexts*, and *configuration contexts*. Process contexts, noted $\mathbb{P}$, are given by the following grammar:

$$\mathbb{P} ::= \bullet \;\mid\; \nu a.\, \mathbb{P} \;\mid\; (P \mid \mathbb{P}) \;\mid\; a\langle\mathbb{P}\rangle \div C \;\mid\; a\langle P\rangle \div b\langle\mathbb{P}\rangle \div \mathbf{0} \;\mid\; a(X) \rhd_\gamma \mathbb{P}$$

Thread contexts, noted $\mathbb{T}$, are given by the following grammar:

$$\mathbb{T} ::= \bullet \;\mid\; k : a\langle\mathbb{P}\rangle \div C \;\mid\; k : a\langle P\rangle \div c\langle\mathbb{P}\rangle \div \mathbf{0} \;\mid\; k : a(X) \rhd_\gamma \mathbb{P}$$

Memory contexts, noted $\mathbb{M}$, are given by the following grammar:

$$\mathbb{M} ::= [\mathbb{T} \mid k_2 : a(X) \triangleright_\gamma P; k] \ \mid \ [k_1 : a\langle P\rangle \div C \mid \mathbb{T}; k]$$

Configuration contexts, noted $\mathbb{C}$, are given by the following grammar:

$$\mathbb{C} ::= \bullet \ \mid \ \nu u.\, \mathbb{C} \ \mid \ (M \mid \mathbb{C})$$

A (one hole) general context for processes $\mathbb{G}$ is thus either a context of the form $\mathbb{C}[k : \mathbb{P}]$, $\mathbb{C}[\mathbb{T}]$ or $\mathbb{C}[\mathbb{M}]$.

The definition of strong and weak barbed congruence for processes can be detailed as follows.

Definition 10 (Strong and weak barbed congruences for processes). *We say two closed processes P and Q are strong (resp. weak) barbed congruent, noted $P \sim_c Q$ (resp. $P \approx_c Q$) if, for all closed process contexts, all closed configuration contexts $\mathbb{C}$, all closed thread contexts $\mathbb{T}$, and all closed memory contexts $\mathbb{M}$, we have $\mathbb{C}[k : \mathbb{P}[P]] \sim \mathbb{C}[k : \mathbb{P}[Q]]$ (resp. $\mathbb{C}[k : \mathbb{P}[P]] \approx \mathbb{C}[k : \mathbb{P}[Q]]$), $\mathbb{C}[\mathbb{T}[P]] \sim \mathbb{C}[\mathbb{T}[Q]]$ (resp. $\mathbb{C}[\mathbb{T}[P]] \approx \mathbb{C}[\mathbb{T}[Q]]$), and $\mathbb{C}[\mathbb{M}[P]] \sim \mathbb{C}[\mathbb{M}[Q]]$ (resp. $\mathbb{C}[\mathbb{M}[P]] \approx \mathbb{C}[\mathbb{M}[Q]]$).*

B Useful lemmas

Lemma 7. *Let σ be an injective substitution that sends channel names to channel names and keys to keys. If $M \sim N$ (resp. $M \approx N$), then $M\sigma \sim N\sigma$ (resp. $M\sigma \approx N\sigma$).*

Proof. By showing that $\mathcal{S} = \{\langle U, V\rangle \mid U \equiv M\sigma, V \equiv N\sigma\}$ is a strong (resp. weak) barbed bisimulation.

Lemma 8. *For any closed configurations M, N, if $M \sim N$, then $\nu u.\, M \sim \nu u.\, N$ and if $M \approx N$, then $\nu u.\, M \approx \nu u.\, N$.*

Proof. By showing that $\mathcal{R} = \{\langle \nu u.\, M, \nu u.\, N\rangle \mid M \sim N \ (resp. M \approx N)\}$ is a strong (resp. weak) barbed bisimulation: if $M \sim N$ (resp. $M \approx N$) then (weak) observables and reductions of M and N are the same, the same is true of (weak) observables and reductions of $\nu u.\, M$ and $\nu u.\, N$.

The following lemma is the croll-π analog of a well-known result related to the encoding of HOπ into the standard π-calculus. The situation is complicated in our case by the necessity to clearly keep track of how tags are handled in bisimulation candidates. A key point is the choice of the fixpoint combinator (noted Y_P in the proof). Care has to be taken to avoid introducing undue causal dependencies via this combinator, a situation that has no equivalent in HOπ. In particular, the proof below *would not work* if we were to change the fixpoint combinator to $Y'_P = c \triangleright t\langle Y\rangle \triangleright P \mid t\langle Y\rangle \mid Y$. The combinator Y'_P creates copies of the P process only on demand, in contrast to Y_P, but in so doing creates artificial causal dependencies between possibly independent executions of P.

$$
\begin{aligned}
\mathcal{R} =& \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \cup Id \qquad\qquad Y_P = t(Y) \triangleright (c \triangleright P) \mid t\langle Y\rangle \mid Y \\
\mathcal{R}_1 =& \{\langle M\{^P/_X\} \,;\, \nu c, t, k_0, k_0'.\, M\{^{\overline{c}}/_X\} \mid k_0 : t\langle Y_P\rangle \mid k_0' : Y_P\rangle\} \\
\mathcal{R}_2 =& \{\langle M\{^P/_X\} \,;\, \nu c, t, \tilde{k}, \tilde{k}', \tilde{k}'', \tilde{h}.\, M\{^{\overline{c}}/_X\} \mid \prod_{i=0}^{n}([k_i : t\langle Y_P\rangle \mid k_i' : Y_P; k_i''] \mid \\
& \qquad k_i'' \prec (h_{i+1}, k_{i+1}, k_{i+1}') \mid h_{i+1} : c \triangleright P) \mid k_n : t\langle Y_P\rangle \mid k_n' : Y_P\rangle\} \\
\mathcal{R}_3 =& \{\langle M\{^P/_X\} \mid \prod_{j=1}^{m} M_j \,; \\
& \qquad \nu c, t, \tilde{k}, \tilde{k}', \tilde{k}'', \tilde{h}.\, M\{^{\overline{c}}/_X\} \mid \prod_{i\in I}([k_i : t\langle Y_P\rangle \mid k_i' : Y_P; k_i''] \mid k_i'' \prec (h_{i+1}, k_{i+1}, k_{i+1}') \mid \\
& \qquad h_{i+1} : c \triangleright P) \mid \prod_{j\in J}([k_j : \overline{c} \mid h_{j+1} : c \triangleright P; l_j] M_j\{^{l_j}/_{k_j}\}) \mid \\
& \qquad k_n : t\langle Y_P\rangle \mid k_n' : Y_P\rangle \mid \\
& I \cup J = \{0, \ldots, n\}, I \cap J = \emptyset\}
\end{aligned}
$$

Fig. 7. Candidate relation for Lemma 9

Lemma 9. *For all closed processes P, all closed configurations M such that $M\{^P/_X\}$ is closed, and all $c, t, k, k' \notin \mathtt{fn}(M, P)$, we have*

$$M\{^P/_X\} \approx_c \nu c, t, k_0, k_0'.\, M\{^{\overline{c}}/_X\} \mid k_0 : t\langle Y_P\rangle \mid k_0' : Y_P$$

where $Y_P = t(Y) \triangleright (c \triangleright P) \mid t\langle Y\rangle \mid Y$.

Proof. We consider the relation $\mathcal{R}$ in Figure7 and prove that it is a weak barbed congruence. In all the relations, $c, t, k_0, k_0' \notin \mathtt{fn}(M, P)$. From $\mathcal{R}_1$ we immediately move to the generalization in $\mathcal{R}_2$, obtained by unfolding the recursion.

Let us consider $\mathcal{R}_2$. The barbs of M are easily matched. If there is an occurrence of P enabled on the left, on the right we can perform an interaction on c and we match the barb (if no trigger is enabled, we unfold again the recursion, remaining in the same relation). On the right we have no other barbs. Let us consider reductions. Unfolding of recursion leave us in the same relation. The same for interactions inside M. Let us consider an interaction on c. Such interactions are always enabled if on the left there is a P enabled, and we can imagine to perform them as soon as possible. This leads us to $\mathcal{R}_3$, which generalizes $\mathcal{R}_2$. Barbs are matched as before. Similarly unfolding of recursion and enabling of copies of P leaves us in the same relation. The same for computations inside M or P. Let us consider rollbacks. Rollbacks inside P or not involving them are easily matched. Rollbacks involving P are matched by undoing the corresponding communication on c, but not the corresponding unfolding of recursion. Unfolding of recursion is never undone.

C Proofs of Section 3

C.1 Proofs of Section 3.3

(Reminder) Lemma 2. (Context Lemma for closed configurations). For any closed configurations M, N, $M \sim_c N$ if and only if, for all closed configurations L, $M \mid L \sim N \mid L$. Likewise, $M \approx_c N$ if and only if, for all L, $M \mid L \approx N \mid L$.

Proof. The left to right implication is immediate, by definition of $\sim_c$. We prove the other direction. As the weak case is identical to the strong one, we do not detail it.

Let $\mathcal{R} = \{\langle \mathbb{C}[M], \mathbb{C}[N] \rangle \mid \forall L, M \mid L \sim N \mid L\}$ be our candidate relation. By definition, $\mathcal{R}$ is a congruence. We now need to show it is included in $\sim$. To this end, we first show by induction on the context $\mathbb{C}$ that for all L, $\mathbb{C}[M] \mid L \sim \mathbb{C}[N] \mid L$.

The base case is immediate as it is our initial hypothesis.

For the parallel composition case, we assume that $\forall L, \mathbb{C}[M] \mid L \sim \mathbb{C}[N] \mid L$ and show that $\forall L, (\mathbb{C}[M] \mid L') \mid L \sim (\mathbb{C}[N] \mid L') \mid L$. This is also immediate by structural congruence.

For the restriction case, we assume that $\forall L, \mathbb{C}[M] \mid L \sim \mathbb{C}[N] \mid L$ and show that $\forall L, (\nu u.\, \mathbb{C}[M]) \mid L \sim (\nu u.\, \mathbb{C}[N]) \mid L$. For each fixed L, let v be a name fresh in relation to $\mathbb{C}[.]$, M, N, and L. We write σ the permutation between u and v. By α-conversion and structural equivalence, we have $(\nu u.\, \mathbb{C}[M]) \mid L \sim (\nu v.\, \mathbb{C}[M]\sigma) \mid L \sim \nu v.\, (\mathbb{C}[M]\sigma \mid L)$. Let $L' = L\sigma$, where u was renamed to (the fresh) v, we then have $L'\sigma = L$, thus $(\nu u.\, \mathbb{C}[M]) \mid L \sim \nu v.\, (\mathbb{C}[M] \mid L')\sigma$.

By induction, we have $\mathbb{C}[M] \mid L' \sim \mathbb{C}[N] \mid L'$. By Lemma 7 strong and weak bisimilarity are preserved by injective renamings (which is the case here as v was chosen fresh), thus we have $(\mathbb{C}[M] \mid L')\sigma \sim (\mathbb{C}[N] \mid L')\sigma$. By Lemma 8, we then have $\nu v.\, (\mathbb{C}[M] \mid L')\sigma \sim \nu v.\, (\mathbb{C}[N] \mid L')\sigma$.

We now compute as follows:

$$\begin{aligned} \nu v.\, (\mathbb{C}[N] \mid L')\sigma = \nu v.\, (\mathbb{C}[N]\sigma \mid L'\sigma) &= \nu v.\, (\mathbb{C}[N]\sigma \mid L) \\ &\equiv (\nu v.\, \mathbb{C}[N]\sigma) \mid L =_\alpha (\nu u.\, \mathbb{C}[N]) \mid L \end{aligned}$$

By transitivity, we thus have $(\nu u.\, \mathbb{C}[M]) \mid L \sim (\nu u.\, \mathbb{C}[N]) \mid L$, as requested.

To conclude, let $\mathbb{C}[M], \mathbb{C}[N]$ from $\mathcal{R}$. By hypothesis, we have $\forall L, M \mid L \sim N \mid L$. By the just proven property, we thus have $\forall L, \mathbb{C}[M] \mid L \sim \mathbb{C}[N] \mid L$. By taking $L = \mathbf{0}$, we have $\mathbb{C}[M] \sim \mathbb{C}[N]$, hence $\mathcal{R} \subseteq \sim$. □

(Reminder) Lemma 3. (Context lemma for closed processes). Let P and Q be closed processes. We have $P \approx_c Q$ if and only if, for all closed configuration contexts $\mathbb{C}$ and $k \notin \mathtt{fn}(P, Q)$, we have $\mathbb{C}[k : P] \approx \mathbb{C}[k : Q]$.

Proof. The left to right implication is clear. To prove the right to left direction we proceed by induction on the form of general contexts for processes. We deal first with contexts of the form $\mathbb{C}[k : \mathbb{P}]$.

$$
\begin{aligned}
\mathcal{S} =& Id_{\mathcal{C}^2_{cl}} \cup \mathcal{S}_1 \cup \mathcal{S}_2 \\
\mathcal{S}_1 =& \{\langle U, V\rangle \mid U \equiv \mathbb{C}[U_0]\ ,\ V \equiv \mathbb{C}[V_0]\ ,\ U_0 = \nu t, k, k'.\, k : t\langle Y_{\mathbb{P}'[P]}\rangle \mid k' : Y_{\mathbb{P}'[P]}\ , \\
& \quad V_0 = \nu t, k, k'.\, k : t\langle Y_{\mathbb{P}'[Q]}\rangle \mid k' : Y_{\mathbb{P}'[Q]}, t, k, k' \notin \mathtt{fn}(\mathbb{C}, \mathbb{P}'[P], \mathbb{P}'[Q])\ ,\ \mathbb{C} \text{ closed configuration context}\} \\
\mathcal{S}_2 =& \{\langle U, V\rangle \mid\ U \equiv \mathbb{C}[U_0]\ ,\ V \equiv \mathbb{C}[V_0] \\
& \quad U_0 = \nu t, \tilde{k}.\, (\prod_{i=1}^{n} m^1_{iP} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j\in J} k^i_0 : c \triangleright \mathbb{P}'[P]) \mid (\prod_{j\in J'} m^2_{jP}) \mid k_1 : t\langle Y_{\mathbb{P}'[P]}\rangle \mid k_2 : Y_{\mathbb{P}'[P]}\ , \\
& \quad V_0 = \nu t, \tilde{k}.\, (\prod_{i=1}^{n} m^1_{iQ} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j\in J} k^i_0 : c \triangleright \mathbb{P}'[Q]) \mid (\prod_{j\in J'} m^2_{jQ}) \mid k_1 : t\langle Y_{\mathbb{P}'[Q]}\rangle \mid k_2 : Y_{\mathbb{P}'[Q]}\ , \\
& \quad (\{t\} \cup \tilde{k}) \cap \mathtt{fn}(\mathbb{C}, \mathbb{P}'[P], \mathbb{P}'[Q]) = \emptyset\ ,\ J \cap J' = \emptyset\ ,\ J \cup J' = \{1, \ldots, n\}\ , \\
& \quad \tilde{k} = \{k, k'\} \cup \{k_i \mid 1 \le i \le n\}\ ,\ \mathtt{corr}(\tilde{m}^1, \tilde{m}^2, \tilde{k})\ ,\ \mathbb{C} \text{ closed configuration context}\}
\end{aligned}
$$

where condition $\mathtt{corr}(\tilde{m}^1, \tilde{m}^2, \tilde{k})$ stipulates the following:

- $m^1_{iR} = [k^1_{i-1} : t\langle Y_{\mathbb{P}'[R]}\rangle \mid k^2_{i-1} : Y_{\mathbb{P}'[R]}; k_i]$
- $k^1_0 = k$ and $k^1_0 = k'$
- $m^2_{iR} = [\eta_i : \overline{c} \mid k^0_i : c \triangleright \mathbb{P}'[R]; l_i]$ for some η_i, l_i

Fig. 8. Candidate relation for case $\mathbb{P} = a\langle\mathbb{P}'\rangle \div C$ of Lemma 3

- $\mathbb{P} = \bullet$. This is just the lemma's assumption.

- $\mathbb{P} = \nu a.\, \mathbb{P}'$. Reductions involving only the configuration context are easily matched. The only possible reduction involving $\mathbb{P}$ is $\mathbb{C}[k : \mathbb{P}[P]] \longrightarrow \mathbb{C}[\nu a.\, k : \mathbb{P}'[P]]$. By inductive hypothesis we know that for each configuration context $\mathbb{K}$ we have $\mathbb{K}[k : \mathbb{P}'[P]] \approx \mathbb{K}[k : \mathbb{P}'[Q]]$. The thesis follows by choosing $\mathbb{K} = \mathbb{C}[\nu a.\, \bullet]$.
- $\mathbb{P} = R \mid \mathbb{P}'$. Reductions involving only the configuration context are easily matched. The only possible reduction involving $\mathbb{P}$ is $\mathbb{C}[k : \mathbb{P}[P]] = \mathbb{C}[k : R \mid \mathbb{P}'[P]] \longrightarrow \mathbb{C}[\nu h_1, h_2.\, k \prec (h_1, h_2) \mid h_1 : R \mid h_2 : \mathbb{P}'[P]]$. $\mathbb{C}[k : \mathbb{P}[Q]]$ has an analogous transition. By inductive hypothesis we know that for each configuration context $\mathbb{K}$ we have $\mathbb{K}[h_2 : \mathbb{P}'[P]] \approx \mathbb{K}[h_2 : \mathbb{P}'[Q]]$. The thesis follows by choosing $\mathbb{K} = \mathbb{C}[\nu h_1, h_2.\, k \prec (h_1, h_2) \mid h_1 : R \mid \bullet]$.
- $\mathbb{P} = a(X) \triangleright_\gamma \mathbb{P}'$. We prove that the relation $\mathcal{S}$ defined below progresses to $\mathcal{S} \approx$, i.e. if $\langle P, Q\rangle \in \mathcal{S}$, the weak observables of P and Q coincide, if $P \longrightarrow P'$ then there exists Q' such that $Q \Longrightarrow Q'$ and $P'\ \mathcal{S} \approx Q'$, and if $Q \longrightarrow Q'$, then there exists P' such that $P \Longrightarrow P'$ and $P' \approx \mathcal{S}\ Q'$.

$$
\begin{aligned}
\mathcal{S} &= \mathcal{S}_1 \cup \mathcal{S}_2 \\
\mathcal{S}_1 &= \{\langle U, V\rangle \mid U \equiv \mathbb{C}[k : \mathbb{P}[P]], V \equiv \mathbb{C}[k : \mathbb{P}[Q]], \mathbb{C} \text{ conf. context}\} \\
\mathcal{S}_2 &= \{\langle U, V\rangle \mid U \equiv \mathbb{C}[m_P], U \equiv \mathbb{C}[m_Q], \mathbb{C} \text{ conf. context}\}
\end{aligned}
$$

where configurations m_P (resp. m_Q) are of the form $[k' : a\langle R\rangle \mid k : \mathbb{P}[P]; h]$ (resp. $[k' : a\langle R\rangle \mid k : \mathbb{P}[Q]; h]$), for some k', R, h.
We check the clauses for weak barbed simulation for the different pairs of configurations in $\mathcal{S}$ to prove that $\mathcal{S}$ progresses to $\mathcal{S} \approx$. Since $\mathcal{S}$ is symmetric, this will ensure that $\mathcal{S}$ progresses to $\mathcal{S} \approx$, and hence is a weak barbed bisimulation.
Consider first a pair $\langle U, V\rangle \in \mathcal{S}_1$. Let $\mathbb{C} = \nu\tilde{u}.\bullet \mid M$ be the configuration context such that $U = \mathbb{C}[k : \mathbb{P}[P]]$, $V = \mathbb{C}[k : \mathbb{P}[Q]]$. Observables of U and V are observables of $\nu\tilde{u}.\, M$, and thus coincide. Assume $U \longrightarrow U'$. We have two cases to consider:

- $U' \equiv \nu\tilde{u}.\, k : \mathbb{P}[P] \mid M'$ and $M \longrightarrow M'$.
  We then have $V \longrightarrow V' = \nu\tilde{u}.\, k : \mathbb{P}[Q] \mid M'$, and $\langle U', V'\rangle \in \mathcal{S}_1$.
- $M \equiv k' : a\langle R\rangle \mid M'$ and $U' = \nu\tilde{u}.\,(\nu h.\, m_P \mid h : \mathbb{P}'[P]) \mid M'$, with $m_P = [k' : a\langle R\rangle \mid k : \mathbb{P}'\{^R/_X\}[P]; h]$.
  We then have $V \longrightarrow \nu\tilde{u}.\,(\nu h.\, m_Q \mid h : \mathbb{P}'[Q]) \mid M' = V'$, where $m_Q = [k' : a\langle R\rangle \mid k : \mathbb{P}'\{^R/_X\}[Q]; h]$.
  Let $\mathbb{C}' = \nu\tilde{u}, h.\bullet \mid M'$. By induction hypothesis, we have $\mathbb{K}[h : \mathbb{P}'[P]] \approx \mathbb{K}[h : \mathbb{P}'[Q]]$ for all $\mathbb{K}$. We thus have $V' = \mathbb{C}'[m_Q \mid h : \mathbb{P}'[Q]] \approx \mathbb{C}'[m_Q \mid h : \mathbb{P}'[P]]$, $U' = \mathbb{C}'[m_P \mid h : \mathbb{P}'[P]]$, and $\langle U', V'\rangle \in \mathcal{S}_2 \approx$.

Consider now a pair $\langle U, V\rangle \in \mathcal{S}_2$. Observables of $\mathbb{C}[m_P]$ and of $\mathbb{C}[m_Q]$ coincide. Assume now that $U \longrightarrow U'$. We have two cases to consider:

- $U' = \mathbb{C}'[m_P]$, in which case we have $V \longrightarrow V' = \mathbb{C}'[m_Q]$, and $\langle U', V'\rangle \in \mathcal{S}_2$.
- $U' = \mathbb{C}'[k : \mathbb{P}[P]]$, resulting from the application of rule S.Roll with target memory m_P (the case where the target is an ancestor such that the trigger is not a descendant of the key of the roll is similar), in which case we have $V \longrightarrow V' = \mathbb{C}'[k : \mathbb{P}[Q]]$, by applying S.Roll to m_Q, and $\langle U', V'\rangle \in \mathcal{S}_1$.

– $\mathbb{P} = a\langle \mathbb{P}'\rangle \div C$. By Lemma 9, we have for all closed configuration contexts $\mathbb{C}$, $k, k', t, c \notin \mathtt{fn}(\mathbb{C}, \mathbb{P}'[P])$,

$$\mathbb{C}[h : a\langle \mathbb{P}'[P]\rangle \div C] \approx \mathbb{C}[\nu c, t, k, k'.\, h : a\langle \overline{c}\rangle \div C \mid k : t\langle Y_{\mathbb{P}'[P]}\rangle \mid k' : Y_{\mathbb{P}'[P]}]$$

and similarly for $\mathbb{P}'[Q]$. It therefore suffices to prove that for all closed configuration contexts $\mathbb{C}$ and all $k, k', t \notin \mathtt{fn}(\mathbb{C}, \mathbb{P}'[P], \mathbb{P}'[Q])$ we have

$$\mathbb{C}[\nu t, k, k'.\, k : t\langle Y_{\mathbb{P}'[P]}\rangle \mid k' : Y_{\mathbb{P}'[P]}] \approx \mathbb{C}[\nu t, k, k'.\, k : t\langle Y_{\mathbb{P}'[Q]}\rangle \mid k' : Y_{\mathbb{P}'[Q]}]$$

under the induction hypothesis that for all $\mathbb{C}$, and all $h \notin \mathtt{fn}(\mathbb{P}'[P], \mathbb{P}'[Q])$ we have $\mathbb{C}[h : \mathbb{P}'[P]] \approx \mathbb{C}[h : \mathbb{P}'[Q]]$.
To prove this, we show that the relation $\mathcal{S}$ defined below progresses to $\mathcal{S} \approx$, which will ensure that $\mathcal{S}$ is a weak barbed bisimulation. Let $\mathcal{S}$ the relation in Figure 8. We now check the different clauses for weak progress to $\mathcal{S} \approx$. Since $\mathcal{S}$ is symmetric, we only need to check the clauses for weak simulation. Notice first that for all pairs $\langle U, V\rangle \in \mathcal{S}$, observables of U are the same as those of V.
Consider now a pair $\langle U, V\rangle \in \mathcal{S}_2$ (the case $\langle U, V\rangle \in \mathcal{S}_1$ is similar, but simpler). We have the following different cases:

1. $U \longrightarrow U' = \mathbb{C}'[U_0]$, derived from applying a reduction rule with no implication of terms from U_0. In this case, we can apply the same reduction rule to get $V \longrightarrow V' = \mathbb{C}'[V_0]$, and $\langle U', V' \rangle \in \mathcal{S}_2$.
2. $\mathbb{C} = \mathbb{C}'[[\mu; h] \mid k_r : \mathsf{roll}\ h \mid \bullet]$, and $U \longrightarrow U'$ derived using rule S.Roll with $U' = \mathbb{C}'[U'_0]$, and

$$U'_0 = \nu t, \tilde{k}. (\prod_{i=1}^{n} m^1_{iP} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in K} k^i_0 : c \triangleright \mathbb{P}'[P]) \mid (\prod_{j \in K'} m^2_{jP}) \mid k_1 : t\langle Y_{\mathbb{P}'[P]} \rangle \mid k_2 : Y_{\mathbb{P}'[P]}$$

with $J \subseteq K$, $K' \subseteq J'$, $K \cap K' = \emptyset$, and $K \cup K' = \{1, \ldots, n\}$, noting that the receipts on t that led to the creation of memory m^1_i are independent from any other action in the context, and thus are never rolled back. In this case, we can apply S.Roll to get $V \rightsquigarrow V' = \mathbb{C}'[V'_0]$ with

$$V'_0 = \nu t, \tilde{k}. (\prod_{i=1}^{n} m^1_{iQ} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in K} k^i_0 : c \triangleright \mathbb{P}'[Q]) \mid (\prod_{j \in K'} m^2_{jQ}) \mid k_1 : t\langle Y_{\mathbb{P}'[Q]} \rangle \mid k_2 : Y_{\mathbb{P}'[Q]}$$

and $\langle U', V' \rangle \in \mathcal{S}_2$.
3. $\mathbb{C} = \mathbb{C}'[\eta : \overline{c} \mid \bullet]$, with $U' = \mathbb{C}'[U'_0]$

$$U'_0 \equiv \nu t, \tilde{k}, l. U'_1 \mid [\eta : \overline{c} \mid k^e_0 : c \triangleright \mathbb{P}'[P]; l] \mid l : \mathbb{P}'[P]$$

$$U'_1 = (\prod_{i=1}^{n} m^1_{iP} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in J \setminus \{e\}} k^i_0 : c \triangleright \mathbb{P}'[P]) \mid (\prod_{j \in J'} m^2_{jP}) \mid k_1 : t\langle Y_{\mathbb{P}'[P]} \rangle \mid k_2 : Y_{\mathbb{P}'[P]}$$

obtained by applying rule S.Com to $\eta : \overline{c}$ and $k^e_0 : Y_{\mathbb{P}'[P]}$. Noting that $[\eta : \overline{c} \mid k^e_0 : c \triangleright \mathbb{P}'[P]; l]$ is of the form given above in the definition of predicate `corr`, we have:

$$U'_0 \equiv \nu t, \tilde{k}, l. l : \mathbb{P}'[P] \mid (\prod_{i=1}^{n} m^1_{iP} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in J \setminus \{e\}} k^i_0 : c \triangleright \mathbb{P}'[P]) \mid (\prod_{j \in J' \cup \{e\}} m^2_{jP}) \mid k_1 : t\langle Y_{\mathbb{P}'[P]} \rangle \mid k_2 : Y_{\mathbb{P}'[P]}$$

In this case, we can apply S.Com to obtain likewise $V \twoheadrightarrow V' = \mathbb{C}'[V'_0]$ with:

$$V'_0 \equiv \nu t, \tilde{k}, l. l : \mathbb{P}'[Q] \mid (\prod_{i=1}^{n} m^1_{iQ} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in J \setminus \{e\}} k^i_0 : c \triangleright \mathbb{P}'[Q]) \mid (\prod_{j \in J' \cup \{e\}} m^2_{jQ}) \mid k_1 : t\langle Y_{\mathbb{P}'[Q]} \rangle \mid k_2 : Y_{\mathbb{P}'[Q]}$$

Define $\mathbb{K} = \mathbb{C}'[\nu l.\, \bullet]$, $K = J \setminus \{e\}$, $K' = J' \cup \{e\}$. We have

$$
\begin{aligned}
U' &\equiv \mathbb{K}[l : \mathbb{P}'[P] \mid U'_2] \\
U'_2 &\equiv (\prod_{i=1}^{n} m^1_{iP} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in K} k^i_0 : c \triangleright \mathbb{P}'[P]) \mid (\prod_{j \in K'} m^2_{jP}) \mid \\
&\quad k_1 : t\langle Y_{\mathbb{P}'[P]} \rangle \mid k_2 : Y_{\mathbb{P}'[P]} \\
V' &\equiv \mathbb{K}[l : \mathbb{P}'[Q] \mid V'_2] \\
V'_2 &\equiv (\prod_{i=1}^{n} m^1_{iQ} \mid k_i \prec (k^i_1, k^i_2, k^i_3)) \mid (\prod_{j \in K} k^i_0 : c \triangleright \mathbb{P}'[Q]) \mid (\prod_{j \in K'} m^2_{jQ}) \mid \\
&\quad k_1 : t\langle Y_{\mathbb{P}'[Q]} \rangle \mid k_2 : Y_{\mathbb{P}'[Q]}
\end{aligned}
$$

By induction hypothesis, we have $V' \approx V'' = \mathbb{K}[l : \mathbb{P}'[P] \mid V'_2]$. Define now $\mathbb{K}' = \mathbb{C}'[\nu l.\, l : \mathbb{P}'[P] \mid \bullet]$. We have $U' \equiv \mathbb{K}'[U'_2]\ \mathcal{S}_2\ \mathbb{K}'[V'_2] = V'' \approx V'$, and thus $\langle U', V' \rangle \in \mathcal{S}_2 \approx$.

– $\mathbb{P} = a\langle R \rangle \div b\langle \mathbb{P}' \rangle \div \mathbf{0}$. Handled in the same way as the previous case.

We finally deal with contexts of the form $\mathbb{C}[\mathbb{T}]$ (dealing with contexts of the form $\mathbb{C}[\mathbb{M}]$ is similar). By Lemma 9, we have

$$\mathbb{C}[\mathbb{T}[P]] \equiv \nu \tilde{v}.\, \mathbb{T}[P] \mid L \approx_c \nu \tilde{v}, c, t, k, k'.\, \mathbb{T}[\overline{c}] \mid L \mid k : t\langle Y_P \rangle \mid k' : Y_P$$

But we just proved in the previous case that $k : t\langle Y_P \rangle \approx_c k : t\langle Y_Q \rangle$ and that $k' : Y_P \approx_c k' : Y_Q$. Hence

$$\mathbb{C}[\mathbb{T}[P]] \approx_c \nu \tilde{v}, c, t, k, k'.\, \mathbb{T}[\overline{c}] \mid L \mid k : t\langle Y_Q \rangle \mid k' : Y_Q = \mathbb{C}[\mathbb{T}[Q]]$$

for $\approx_c$ is a congruence. □

A process context $\mathbb{P}$ is closing for a set of processes P_i iff $\mathbb{P}[P_i]$ are all closed. A substitution σ is closing for a set of processes P_i iff $P_i\sigma$ are all closed.

(Reminder) Lemma 4. (Context lemma for open processes). Let P and Q be (possibly open) processes. We have $P \approx^o_c Q$ if and only if, for all closed configuration contexts $\mathbb{C}$, for all substitutions σ closing for P and Q, $k \notin \mathtt{fn}(P, Q)$, we have $\mathbb{C}[k : P\sigma] \approx \mathbb{C}[k : Q\sigma]$.

Proof. Let us prove the only if part.

We have to prove that for each closing σ we have $\mathbb{C}[k : P\sigma] = \mathbb{C}[k : Q\sigma]$. We proceed by induction on the number of bindings in σ. The case of zero bindings follows from Lemma 3. We have to consider bindings of the form $\{^R/_X\}$ and bindings of the form $\{^{k'}/_\gamma\}$.

Let us start from bindings of the form $\{^R/_X\}$. Consider the process context $\mathbb{P} = a\langle R \rangle \mid a(X) \triangleright \mathbb{P}'[\bullet]$ where a is fresh. If $\mathbb{P}'$ closes the other variables, this is a closing context. By hypothesis $\mathbb{C}[k : a\langle R \rangle \mid a(X) \triangleright \mathbb{P}'[P]] \approx \mathbb{C}[k : a\langle R \rangle \mid$

$a(X) \triangleright \mathbb{P}'[Q]]$. By performing the communication on both sides we have:
$\mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^R/_X\}[P\{^R/_X\}]] \approx \mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright \mathbb{P}'[Q]; k'] \mid k' : \mathbb{P}'\{^R/_X\}[Q\{^R/_X\}]]$.

Note that replacing the memory in the second configuration by substituting it with the one in the first configuration will not change the behavior since the content of the memory is never liberated since both the processes are dependent on any key k'' such that a $\mathsf{roll}\ k''$ may involve the memory, and there is no $\mathsf{roll}\ k'$. Thus we have:
$\mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^R/_X\}[P\{^R/_X\}]] \approx \mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^R/_X\}[Q\{^R/_X\}]]$. We show now that: $\mathbb{C}[k : \mathbb{P}'\{^R/_X\}[P\{^R/_X\}]] \approx \mathbb{C}[k : \mathbb{P}'\{^R/_X\}[Q\{^R/_X\}]]$. Communications cannot involve the memory, thus they have to be matched by the process. Considering rollbacks, the memory would have been removed in the original process if involved, and the continuation is the same in the old and new process. Thus corresponding actions are done.

Note now that $\mathbb{P}'\{^R/_X\}[\bullet]$ is an arbitrary context closing for $P\{^R/_X\}$ and $Q\{^R/_X\}$, and that we have here one less binding needed for a closing substitution, thus we can apply inductive hypothesis to get that for each closing σ' we have $\mathbb{C}[k : P\{^R/_X\}\sigma'] = \mathbb{C}[k : Q\{^R/_X\}\sigma']$. Since every σ can be expressed as $\{^R/_X\}\sigma'$ the thesis follows.

Let us consider now bindings of the form $\{^{k'}/_\gamma\}$. Consider the process context $\mathbb{P} = a\langle R\rangle \mid a(X) \triangleright_\gamma \mathbb{P}'[\bullet]$ where a is fresh and X never occurs in the continuation. If $\mathbb{P}'$ closes the other variables, this is a closing context. By hypothesis $\mathbb{C}[k : a\langle R\rangle \mid a(X) \triangleright_\gamma \mathbb{P}'[P]] \approx \mathbb{C}[k : a\langle R\rangle \mid a(X) \triangleright_\gamma \mathbb{P}'[Q]]$. By performing the communication on both sides we have:
$\mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright_\gamma \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^{k'}/_\gamma\}[P\{^{k'}/_\gamma\}]] \approx \mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright_\gamma \mathbb{P}'[Q]; k'] \mid k' : \mathbb{P}'\{^{k'}/_\gamma\}[Q\{^{k'}/_\gamma\}]]$.

Note that replacing the memory in the second process with a copy of the memory in the first process does not change the behavior: the only case where the content of the memory is freed is during a rollback, but then we go to the first process which is known to be equivalent to the second. Thus we have:
$\mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright_\gamma \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^{k'}/_\gamma\}[P\{^{k'}/_\gamma\}]] \approx \mathbb{C}[\nu k', h_1, h_2.\, k \prec (h_1, h_2) \mid [h_1 : a\langle R\rangle \mid h_2 : a(X) \triangleright_\gamma \mathbb{P}'[P]; k'] \mid k' : \mathbb{P}'\{^{k'}/_\gamma\}[Q\{^{k'}/_\gamma\}]]$. We show now that: $\mathbb{C}[k' : \mathbb{P}'\{^{k'}/_\gamma\}[P\{^{k'}/_\gamma\}]] \approx \mathbb{C}[k' : \mathbb{P}'\{^{k'}/_\gamma\}[Q\{^{k'}/_\gamma\}]]$. Communications cannot involve the memory, thus they have to be matched by the process. Concerning rollbacks, rollback of k' has to be matched by a rollback of k', since this is the only move that makes the barb at a observable. The rolled-back memory should be in the context, and should be the same in both the cases, thus we go to the identity. Other rollbacks are trivially matched. The thesis follows.

For the if part, we prove that if for each closing σ and each closed context $\mathbb{C}$ we have $\mathbb{C}[k : P\sigma] \approx \mathbb{C}[k : Q\sigma]$ then for each closing $\mathbb{P}$ we have $\mathbb{C}[k : \mathbb{P}[P]] \approx \mathbb{C}[k : \mathbb{P}[Q]]$ (the case for thread contexts $\mathbb{C}[\mathbb{T}[\bullet]]$ and memory contexts $\mathbb{C}[\mathbb{M}[\bullet]]$

$$
\begin{aligned}
\mathcal{R} =& \mathcal{R}_1 \cup \mathcal{R}_2 \cup Id \\
\mathcal{R}_1 =& \{\langle \mathbb{C}[k : a(X) \triangleright_\gamma \mathbb{P}''[P]] \; ; \; \mathbb{C}[k : a(X) \triangleright_\gamma \mathbb{P}''[Q]] \rangle\} \\
\mathcal{R}_2 =& \{\langle \mathbb{C}[\nu k'.\, [k : a(X) \triangleright_\gamma \mathbb{P}''[P] \mid k_1 : a\langle R\rangle; k'] \mid \mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[P\{^R/_X\}\{^{k'}/_\gamma\}]] \; ; \\
& \quad \mathbb{C}[\nu k'.\, [k : a(X) \triangleright_\gamma \mathbb{P}''[Q] \mid k_1 : a\langle R\rangle; k'] \mid \mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[Q\{^R/_X\}\{^{k'}/_\gamma\}]] \rangle\}
\end{aligned}
$$

Fig. 9. Candidate relation of Lemma 4

is handled similarly). Note that only triggers can bind variables. The proof is by induction on the number of triggers. If the number of triggers is 0 then the thesis follows from Lemma 3. For the inductive case consider the outermost trigger around the bullet. We have: $\mathbb{P}[\bullet] = \mathbb{P}'[a(X) \triangleright_\gamma \mathbb{P}''[\bullet]]$.

We would need to consider pairs of the form: $(\mathbb{C}[k : \mathbb{P}'[a(X) \triangleright_\gamma \mathbb{P}''[P]]], \mathbb{C}[k : \mathbb{P}'[a(X) \triangleright_\gamma \mathbb{P}''[Q]]])$.

Since $a(X) \triangleright_\gamma \mathbb{P}''[P]$ and $a(X) \triangleright_\gamma \mathbb{P}''[P]$ are closed we may assume that $\mathbb{P}'$ is empty thanks to Lemma 3.

Let us consider the relation $\mathcal{R}$ defined in Figure 9. Moves involving only $\mathbb{C}$ are trivially matched. Note that $\mathbb{P}''[P]$ and $\mathbb{P}''[Q]$ cannot move since they are not enabled. Moves involving a rollback at k will lead to the identity. The only other possibility is an interaction with a message $k_1 : a\langle R\rangle$. This leads to a pair in $\mathcal{R}_2$.

Let us consider $\mathcal{R}_2$. We know that for each closing substitution σ we have $\mathbb{C}[k : P\sigma] \approx \mathbb{C}[k : Q\sigma]$. This implies that for each closing substitution σ' we have $\mathbb{C}[k : P\{^R/_X\}\{^{k'}/_\gamma\}\sigma'] \approx \mathbb{C}[k : Q\{^R/_X\}\{^{k'}/_\gamma\}\sigma']$. Consider the process context $\mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[\bullet]$ which is closing for $P\{^R/_X\}\{^{k'}/_\gamma\}$ and $Q\{^R/_X\}\{^{k'}/_\gamma\}$. Since this has one less trigger around the bullet by inductive hypothesis we know that $\mathbb{C}[k : \mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[P\{^R/_X\}\{^{k'}/_\gamma\}]] \approx \mathbb{C}[k : \mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[Q\{^R/_X\}\{^{k'}/_\gamma\}]]$. Let us go back to the pair in $\mathcal{R}_2$. Equality of barbs follows from weak barbed congruence of the pair above. Let us consider transitions. Transitions involving only $\mathbb{C}$ are trivially matched. Rollback sends us back to $\mathcal{R}_1$ or to the identity. Communications involving $\mathbb{P}''\{^R/_X\}\{^{k'}/_\gamma\}[P\{^R/_X\}\{^{k'}/_\gamma\}]$ are matched because of the equivalence above, since the only difference between the two contexts, given by the content of the memory, has no effect on communications. This concludes the if part. □

(Reminder) Theorem 1. (Context Lemma). Two processes P and Q are weak barbed congruent, $P \approx_c^o Q$, if and only if for all substitution σ closing P and Q, all closed configurations M, and all keys k we have: $M \mid (k : P\sigma) \approx M \mid (k : Q\sigma)$.

Proof. A direct consequence of Lemma 4 and Lemma 2. □

C.2 Proofs of Section 4.2

(Reminder) Theorem 2. There is no encoding $(\!|\bullet|\!)$ from croll-π to roll-π such that:

1. If M has a computation performing at least a backward step, then $(\!|M|\!)$ has a computation performing at least a backward step;
2. If M has only finite computations, then $(\!|M|\!)$ has only finite computations.

Proof. Take configuration $M = \nu k.\, k : \overline{a} \div \overline{b} \mid a \triangleright_\gamma \mathsf{roll}\ \gamma$. This configuration has a computation composed by one forward step followed by one backward step, which then stops. This is the only possible computation. Assume towards a contradiction that an encoding exists and consider $(\!|M|\!)$. $(\!|M|\!)$ should have at least a computation including a backward step. From roll-π loop lemma, if we have a backward step, we are able to go forward again, and then there is a looping computation. This is in contrast with the second condition of the encoding, then the thesis follows. □

D Proofs of Section 4

D.1 Arbitrary alternatives

(Reminder) Lemma 5. For each configuration M k-dependent and complete such that $k', t, k_1, k_2 \notin \mathtt{fn}(M)$ we have $M \approx_c \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid [k_1 : t\langle Q\rangle \div C \mid k_2 : t(X) \triangleright R; k'] \mid M\{^{k'}/_k\}$.

Proof. Thanks to Lemma 2 from Appendix B, we only need to prove that for all closed configurations L we have $\nu t.\, M \mid L \approx \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid [N; k'] \mid M \mid L$ where M and N are defined as in the statement of the lemma.

We will consider the relation $\mathcal{R}$ below and prove that it is a weak barbed bisimulation.

$$\begin{aligned}\mathcal{R} = \{&\langle \nu t.\, M \mid L\ ;\ \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid [N; k'] \mid M\{^{k'}/_k\} \mid L\rangle \mid \\ &L \text{ closed}, k' \notin \mathtt{fn}(M), k <: M\ , \mathtt{complete}(M)\} \cup Id\end{aligned}$$

The barbs coincide and interactions involving only L are trivially matched. The same for interactions involving only M and communications involving M and L. We thus only have to consider $\mathsf{roll}\ k_1$ for some k_1 ancestor of k or for $k_1 = k$. This leads to the identity, since all the terms in M are involved. Note that terms which are not k-dependent inside memories are equal in both the terms, since the only difference is given by the substitution on key k. Also, the memory $[N; k']$ is completely removed and the restriction on k' and $\tilde{h}$ can be garbage collected. □

$$
\begin{aligned}
\mathcal{R} =& \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \cup Id \\
\mathcal{R}_1 =& \{\langle k : a\langle P\rangle \mid L \,;\, k : (\nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle) \mid L\rangle \mid L \text{ closed}, \, k \notin \mathtt{fn}(P)\} \\
\mathcal{R}_2 =& \{\langle \nu k'.\, [k : a\langle P\rangle \mid k_t : a(X) \rhd_\gamma Q; k'] \mid L \,;\, \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid \\
& \quad k_1 : Y \mid [k_2 : a\langle P\rangle \div t\langle Y\rangle \mid k_t : a(X) \rhd_\gamma Q; k'] \mid L\rangle \mid \text{ closed}, \, k \notin \mathtt{fn}(P)\} \\
\mathcal{R}_3 =& \{\langle \nu k'.\, k : a\langle P\rangle \mid L \,;\, \nu k', t, k_1, k_2.\, k \prec (k_1, k_2) \mid k_1 : Y \mid k_2 : t\langle Y\rangle \mid L\rangle \mid L \text{ closed}, \, k \notin \mathtt{fn}(P)\} \\
\mathcal{R}_4 =& \{\langle \nu k'.\, k : a\langle P\rangle \mid L \,;\, \nu k', t, k'', k_1, k_2.\, k \prec (k_1, k_2) \mid [k_1 : Y \mid k_2 : t\langle Y\rangle; k''] \mid k'' : Y \mid a\langle P\rangle \div t\langle Y\rangle \mid L\rangle \mid \\
& \quad L \text{ closed}, \, k \notin \mathtt{fn}(P)\}
\end{aligned}
$$

Fig. 10. Candidate relation for Proposition 2

D.2 Endless retries

(Reminder) Proposition 2. For any closed process P with roll-π messages, $P \approx_c (\!| P |\!)_{er}$.

Proof. We consider just one instance of roll-π message, the thesis will follow by transitivity.

We prove that for each P, for all closed process contexts $\mathbb{P}$ closing for P, all closed configuration contexts $\mathbb{C}$, and all $k \notin \mathtt{fn}(P)$, we have $\mathbb{C}[k : \mathbb{P}[a\langle P\rangle]] \approx \mathbb{C}[k : \mathbb{P}[\nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle]]$ where $Y = t(Z) \rhd Z \mid a\langle P\rangle \div t\langle Z\rangle$.

Thanks to Lemma 4 and Lemma 2 from Appendix B, we only need to prove that for all closed configurations L and $k \notin \mathtt{fn}(P)$, we have $k : a\langle P\rangle \mid L \approx k : (\nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle) \mid L$.

We will consider the relation $\mathcal{R}$ in Figure 10 and prove that it is a weak barbed congruence. For $\mathcal{R}_1$ it is easy to see that barbs coincide. Moves involving only L are easily matched. Moves involving $k : a\langle P\rangle$ require to apply structural congruence to the second term. We will consider $k : (\nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle) \equiv \nu t.\, k \prec (k_1, k_2) \mid k_1 : Y \mid k_2 : a\langle P\rangle \div t\langle Y\rangle$. The only possible interaction involving $k : a\langle P\rangle$ is a communication on a or a rollback. Communication requires that $L = k_t : a(X) \rhd_\gamma Q \mid L'$. With this assumption the communication leads to a pair in $\mathcal{R}_2$. Note that using structural congruence we can assume that the restriction on k' is at top level. Note also that L has changed. Rollback instead completely removes the parts of the configuration which are different on the two sides, leading to the identity. There is no other possible reduction from $k : (\nu t.\, Y \mid a\langle P\rangle \div t\langle Y\rangle) \mid L)$ thus this concludes the proof for $\mathcal{R}_1$.

Let us consider $\mathcal{R}_2$. Again the barbs are the same and reductions involving only L are trivially matched. The only reductions that may involve the memory are that the memory is part of a rollback concerning an ancestor of k' or k' itself. For the first case we have three possibilities: either the ancestor of k' is an ancestor of k, or of k_t, or both. If it is an ancestor of k (or of both) we go to the identity. If it is an ancestor of k_t but not of k the process is released and we go back to $\mathcal{R}_1$. The second case sends us to a pair in $\mathcal{R}_3$ since the released trigger is the same in both the cases.

$$
\begin{aligned}
\mathcal{R} =& \mathcal{R}_1 \cup \mathcal{R}_2 \cup \mathcal{R}_3 \cup \mathcal{R}_4 \cup \mathcal{R}_5 \cup \mathcal{R}_6 \cup Id \\
\mathcal{R}_1 =& \{\langle k : a(X) \triangleright_\gamma Q \div C \mid L \,;\, k : \nu c, d.\, \overline{c} \div \overline{d} \div \mathbf{0} \mid c \triangleright_\gamma a(X) \triangleright Q \mid d \triangleright C \mid L\rangle\} \\
\mathcal{R}_2 =& \{\langle k : a(X) \triangleright_\gamma Q \div C \mid L \,;\, \nu c, d, k_1, h_1, h_2, h_3.\, k \prec (h_1, h_2, h_3) \mid [h_1 : \overline{c} \div \overline{d} \div \mathbf{0} \mid h_2 : c \triangleright_\gamma a(X) \triangleright Q; k_1] \mid \\
& \quad k_1 : a(X) \triangleright Q\{^{k_1}/_\gamma\} \mid h_3 : d \triangleright C \mid L\rangle\} \\
\mathcal{R}_3 =& \{\langle \nu k_2.\, [k' : a\langle R\rangle \mid k : a(X) \triangleright_\gamma Q \div C; k_2] \mid k_2 : Q\{^{R,k_2}/_{X,\gamma}\} \mid L \,;\, \nu c, d, k_1, k_2, h_1, h_2, h_3.\, k \prec (h_1, h_2, h_3) \mid \\
& \quad [h_1 : \overline{c} \div \overline{d} \div \mathbf{0} \mid h_2 : c \triangleright_\gamma a(X) \triangleright Q; k_1] \mid [k' : a\langle R\rangle \mid k_1 : a(X) \triangleright Q; k_2] \mid k_2 : Q\{^{R,k_1}/_{X,\gamma}\} \mid h_3 : d \triangleright C \mid L\rangle\} \\
\mathcal{R}_4 =& \{\langle \nu k_2.\, [k' : a\langle R\rangle \mid k : a(X) \triangleright_\gamma Q \div C; k_2] \mid M\{^{k_2}/_\gamma\} \mid L \,;\, \nu c, d, k_1, k_2, h_1, h_2, h_3.\, k \prec (h_1, h_2, h_3) \mid \\
& \quad [h_1 : \overline{c} \div \overline{d} \div \mathbf{0} \mid h_2 : c \triangleright_\gamma a(X) \triangleright Q; k_1] \mid [k' : a\langle R\rangle \mid k_1 : a(X) \triangleright Q; k_2] \mid M\{^{k_1}/_\gamma\} \mid h_3 : d \triangleright C \mid L\rangle \mid \\
& \quad k_2 <: M\} \\
\mathcal{R}_5 =& \{\langle k : C \mid L \,;\, \nu c, d, h_1, h_2, h_3.\, k \prec (h_1, h_2, h_3) \mid h_1 : \overline{d} \div \mathbf{0} \mid h_2 : c \triangleright_\gamma a(X) \triangleright Q \mid h_3 : d \triangleright C \mid L\rangle\} \\
\mathcal{R}_6 =& \{\langle M \mid L \,;\, \nu c, d, k_3, h_1, h_2, h_3.\, k \prec (h_1, h_2, h_3) \mid [h_1 : \overline{d} \div \mathbf{0} \mid h_3 : d \triangleright C; k_3] \mid M\{^{k_3}/_k\} \mid \\
& \quad h_2 : c \triangleright_\gamma a(X) \triangleright Q \mid L\rangle \mid k <: M\}
\end{aligned}
$$

Fig. 11. Candidate relation for Proposition 3

Let us consider $\mathcal{R}_3$. We have an internal reduction on the right leading us to $\mathcal{R}_4$, thus to match the barb at a we refer to $\mathcal{R}_4$.

Let us consider $\mathcal{R}_4$. Thanks to Lemma 5 the right configuration is equivalent to the one in $\mathcal{R}_1$. □

D.3 Triggers with Alternative

(Reminder) Proposition 3. For any closed process P with triggers with alternative, $P \approx_c (\!|P|\!)_{at}$.

Proof. We can consider just one instance of the trigger with alternative, and the thesis in the general case will follow from transitivity. Thanks to Lemma 4 and Lemma 2 we only have to prove that for each closed configuration L and each $k \notin \mathtt{fn}(Q, C)$ we have $k : a(X) \triangleright_\gamma Q \div C \mid L \approx k : \nu c, d.\, \overline{c} \div \overline{d} \div \mathbf{0} \mid c \triangleright_\gamma a(X) \triangleright Q \mid d \triangleright C$.

We will consider the relation $\mathcal{R}$ in Figure 11 and prove that it is a weak barbed congruence. In all the relations, L is closed and $k \notin \mathtt{fn}(Q, C)$.

For $\mathcal{R}_1$, barbs coincide and moves involving just L are trivially matched. For any other move, we assume to execute first the extrusion of c and d, the split of the parallel composition and the communication on c on the right side, moving to $\mathcal{R}_2$.

For $\mathcal{R}_2$, the most interesting case is a communication on a with a message $a\langle R\rangle$ inside L, leading to $\mathcal{R}_3$. The other moves lead to the identity or to staying in the relation.

For $\mathcal{R}_3$, we generalize the tuple to the one in $\mathcal{R}_4$.

For $\mathcal{R}_4$, barbs and moves of L are easily matched. Communications of M (possibly with L) are easily matched. Execution of rolls of k_2 are matched by execution of rolls of k_1. The corresponding rollbacks lead to $\mathcal{R}_5$. Other rollbacks lead to the identity.

For $\mathcal{R}_5$, we execute the communication at d going to $\mathcal{R}_6$ to match all the challenges. Actually $\mathcal{R}_6$ is a generalization of the resulting term.

For $\mathcal{R}_6$ note that the trigger at c can never execute. Communications are easily matched. Rollbacks on k or its ancestors lead to the identity. $\square$

E Proofs of Section 6

To prove a behavioral correspondence between TransCCS and the corresponding croll-π configurations, we have to consider the different shapes that a running transaction can take at runtime. In fact, the possible evolutions of computation in croll-π allow different croll-π configurations to represent the same TransCCS process. For instance, $[\![P \triangleright_k Q]\!]$ can be represented in different ways according to whether components in P and Q were inside the transaction since the very beginning or entered later, and different memories may track different past interactions leading to the same process.

For this reason we generalize the concept of translation to the one of possible translation. Given a TransCCS process P, its set of possible translations $(\!|P|\!)^p$ contains all the configurations corresponding to P. Writing $(\!|P|\!)^p$ inside a context is a shortcut for writing all the configurations obtained by putting the different configurations in $(\!|P|\!)^p$ inside the same context.

Definition 11 (Possible translations). *The set of possible translations $(\!|P|\!)^p$ of a TransCCS process P is defined by structural induction on P, and then closed using some closure operations. For inductive definition, all the clauses are defined as in Definition 8, but the one for transaction which includes, in addition to:*

$$(\!|[\![P \triangleright_l Q]\!]|\!)^p = \nu a, c.\, \overline{a} \div \overline{c} \mid a \triangleright_\gamma (\nu l.\, (\!|P|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \triangleright X) \mid c \triangleright (\!|Q|\!)^p$$

also, whenever the transaction can be written as $[\![P \mid R_1 \triangleright_l Q \mid R_2]\!]$, configurations of the form:

$$\begin{aligned}(\!|[\![P \mid R_1 \triangleright_l Q \mid R_2]\!]|\!)^p = \nu a, c, k', l, \tilde{h}, \tilde{h}'.\, & k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid k' \prec (h'_1, h'_4) \mid \\ & h'_4 \prec (h'_2, h'_3) \mid [h_1 : \overline{a} \div \overline{c} \mid h_2 : a \triangleright_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \triangleright X); k'] \mid M_P \mid \\ & h'_2 : l\langle \mathsf{roll}\ k'\rangle \mid h'_3 : l(X) \triangleright X \mid M_R \mid h_3 : c \triangleright (\!|Q|\!)^p\end{aligned}$$

In the last clause we have $\tilde{h} = \{h_1, h_2, h_3, h_4\}$, and $\tilde{h}' = \{h'_1, h'_2, h'_3, h'_4\}$. Also M_P is h'_1-dependent and complete and $M_P \in (\!|P|\!)^p$. M_R instead is not h'_1 dependent, and $M_R \in (\!|R_1|\!)^p$. Furthermore, $R_2 \Longrightarrow R_3$ for some R_3 such that $M_P \not\downarrow_{k'} \mid M_R \in (\!|R_3|\!)^p$.

In possible translations any process R can be replaced by a message $\overline{c}$ and a trigger $c \triangleright R$, modeling the fact that the process may be a compensation not released yet.

Keys and memories and connectors are mostly undefined in possible translations. This means that we have different possible translations for each assignment of keys to toplevel processes and for each set of connectors compatible with the constraints above and such that the obtained configuration is coherent, and that we can freely add memories tracking interactions occurring outside transactions. This may include adding toplevel restrictions for keys.

Possible translations are also closed under structural congruence.

Note that the translation $\nu k.\, k : (\!| P |\!)$ of the TransCCS process P belongs to $(\!| P |\!)^p$. We now discuss the relation between a process P and its possible translations $(\!| P |\!)^p$ from both a static and a dynamic point of view.

From the static point of view, given a process P, all the configurations in $(\!| P |\!)^p$ have the same set of weak barbs of P.

Lemma 10. *Let P be a TransCCS process and let $M \in (\!| P |\!)^p$. Then:*

- *if $P{\downarrow_a}$ then $M \Longrightarrow{\downarrow_a}$;*
- *if $M{\downarrow_a}$ then $P \Longrightarrow{\downarrow_a}$.*

Proof. The proof is by structural induction on P. All the cases are easy. Just note that barbs in $[\![P \triangleright_k Q]\!]$, if not yet visible in a possible translation $(\!| [\![P \triangleright_k Q]\!] |\!)^p$, become visible as soon as the communication at a starting the transaction is executed. □

From a dynamic point of view we show that for each TransCCS process P, P and all its possible translations $(\!| P |\!)^p$ have related evolutions.

Lemma 11. *Let P be a TransCCS process and $(\!| P |\!)^p$ the set of its possible translations. If $P \longrightarrow P_1$ then for each possible translation $(\!| P |\!)^p$ we have:*

1. *either $(\!| P |\!)^p \Longrightarrow\approx M$ and M is a possible translation of P_1 or;*
2. *$P \longrightarrow P_1$ is derived using rule (R-Ab), $(\!| P |\!)^p \Longrightarrow\approx M$, $P_1 \Longrightarrow P_2$ and M is a possible translation of P_2.*

Proof. The proof is by case analysis on the rule applied to derive $P \longrightarrow P_1$. For matching the challenge we can always assume that all the communications starting the relevant transactions have already been executed, thus considering only possible transactions of the second form in Definition 11. Similarly, we can assume all the compensations have been released, thus there is no need to consider closure under communications at c.

(R-Comm): a possible translation of $\overline{a} \mid a.P$ has the form $k_1 : \overline{a} \mid k_2 : a \triangleright (\!| P |\!)^p$. The transition can be matched by $k_1 : \overline{a} \mid k_2 : a \triangleright (\!| P |\!)^p \longrightarrow \nu k'.\, [k_1 : \overline{a} \mid k_2 : a \triangleright (\!| P |\!)^p; k'] \mid k' : (\!| P |\!)^p$. If the interaction occurs at top-level, i.e. outside of any transaction, nothing else has to be proved since memories and top-level restrictions can be freely added. If instead the interaction happened inside a transaction, we have to show that the constraints imposed by transactions on keys and on memories are preserved. In this case, both the interacting

processes have to be inside the body of the same transaction. We have different cases according to whether they belong to M_P or to M_R. If they both belong to M_P the resulting configuration is still k'-dependent and complete. If both the interacting processes are in M_R then the resulting process is again in M_R, and R_2 could evolve to a R_3 matching the transition. If they are one in M_P and the other one in M_R then the continuation will belong to M_P, and the process in R_3 is retrieved by projection since it is not k'-dependent, thus R_3 is unchanged.

(R-Ab): a possible translation of the left-hand side is

$$\begin{array}{l}\nu a, c, k', l, \tilde{h}, \tilde{h}'.\, k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid k' \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid \\ \quad [h_1 : \overline{a} \div \overline{c} \mid h_2 : a \rhd_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \rhd X); k'] \mid \\ \quad M_P \mid h'_2 : l\langle \mathsf{roll}\ k'\rangle \mid h'_3 : l(X) \rhd X \mid M_R \mid h_3 : c \rhd (\!|Q|\!)^p\end{array}$$

By executing the communication at l we get

$$\begin{array}{l}\nu a, c, k', l, \tilde{h}, \tilde{h}', k''.\, k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid k' \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid \\ \quad [h_1 : \overline{a} \div \overline{c} \mid h_2 : a \rhd_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \rhd X); k'] \mid \\ \quad M_P \mid [h'_2 : l\langle \mathsf{roll}\ k'\rangle \mid h'_3 : l(X) \rhd X; k''] \mid k'' : \mathsf{roll}\ k' \mid M_R \mid h_3 : c \rhd (\!|Q|\!)^p\end{array}$$

Now by executing $\mathsf{roll}\ k'$ we get:

$$\begin{array}{l}\nu a, c, \tilde{h}.\, k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid h_1 : \overline{c} \mid h_2 : a \rhd_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \rhd X) \mid \\ \quad M_P \lightning_{k'} \mid M_R \mid h_3 : c \rhd (\!|Q|\!)^p\end{array}$$

Note that the trigger at a cannot be activated any more. It is easy to show that this process is thus strong barbed equivalent to:

$$\nu c, \tilde{h}.\, k \prec (h_1, h_2) \mid h_1 : \overline{c} \mid M_P \lightning_{k'} \mid M_R \mid h_2 : c \rhd (\!|Q|\!)^p$$

where now $\tilde{h}$ contains just $\{h_1, h_2\}$. By executing communication at c we get:

$$\nu c, \tilde{h}, k''.\, k \prec (h_1, h_2) \mid [h_1 : \overline{c} \mid h_2 : c \rhd (\!|Q|\!)^p; k''] \mid k'' : (\!|Q|\!)^p \mid M_P \lightning_{k'} \mid M_R$$

Thanks to Lemma 5 and by garbage collecting restriction of c this is strong barbed equivalent to:

$$k : (\!|Q|\!)^p \mid M_P \lightning_{k'} \mid M_R$$

The thesis follows since R_2 can evolve to R_3 such that $M_P \lightning_{k'} \mid M_R \in (\!|R_3|\!)^p$. This is the only case where the second case of the thesis applies.

(R-Co): a possible translation of the left-hand side is

$$\begin{array}{l}\nu a, c, k', l, \tilde{h}, \tilde{h}'.\, k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid k' \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid \\ \quad [h_1 : \overline{a} \div \overline{c} \mid h_2 : a \rhd_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma\rangle \mid l(X) \rhd X); k'] \mid \\ \quad M_P \mid h'_2 : l\langle \mathsf{roll}\ k'\rangle \mid h'_3 : l(X) \rhd X \mid M_R \mid h_3 : c \rhd (\!|Q|\!)^p\end{array}$$

where $M_P \equiv M'_P \mid k_c : (\!|\mathsf{co}\ l|\!)^p = M'_P \mid k_c : \mathtt{1}(X) \triangleright \mathbf{0}$. This trigger can take $l\langle \mathsf{roll}\ k' \rangle$ leading to:

$$\begin{aligned}
&\nu a, c, k', l, \tilde{h}, \tilde{h}', k''.\, k \prec (h_1, h_4) \mid h_4 \prec (h_2, h_3) \mid k' \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid \\
&\quad [h_1 : \overline{a} \div \overline{c} \mid h_2 : a \triangleright_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma \rangle \mid l(X) \triangleright X); k'] \mid M'_P \mid \\
&\quad [h'_2 : l\langle \mathsf{roll}\ k' \rangle \mid k_c : \mathtt{1}(X) \triangleright \mathbf{0}; k''] \mid k'' : \mathbf{0} \mid h'_3 : l(X) \triangleright X \mid M_R \mid h_3 : c \triangleright (\!|Q|\!)^p
\end{aligned}$$

Message $l\langle \mathsf{roll}\ k' \rangle$ will never be released by a rollback, since there is no $\mathsf{roll}\ k''$ and other rolls target an ancestor transaction, from whose key both k_c and h'_2 are descendants. Since there is no other $\mathsf{roll}\ k'$, no such rollback will ever be executed. We can thus garbage collect the compensation $\overline{c}$, and, as a consequence, the trigger at c and c itself getting:

$$\begin{aligned}
&\nu a, k', l, \tilde{h}, \tilde{h}', k''.\, k \prec (h_1, h_2) \mid k' \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid \\
&\quad [h_1 : \overline{a} \mid h_2 : a \triangleright_\gamma (\nu l.\, (\!|P'|\!)^p \mid l\langle \mathsf{roll}\ \gamma \rangle \mid l(X) \triangleright X); k'] \mid M'_P \mid \\
&\quad [h'_2 : l\langle \mathsf{roll}\ k' \rangle \mid k_c : \mathtt{1}(X) \triangleright \mathbf{0}; k''] \mid k'' : \mathbf{0} \mid h'_3 : l(X) \triangleright X \mid M_R
\end{aligned}$$

Note that $M'_P \mid [h'_2 : l\langle \mathsf{roll}\ k' \rangle; k'' \mid k_c : \mathtt{1}(X) \triangleright \mathbf{0}] \mid k'' : \mathbf{0} \mid h'_3 : l(X) \triangleright X$ is k'-dependent and complete thus thanks to Lemma 5 the configuration above is equivalent to:

$$\begin{aligned}
&\nu a, l, \tilde{h}, \tilde{h}', k''.\, k \prec (h'_1, h'_4) \mid h'_4 \prec (h'_2, h'_3) \mid M'_P \mid \\
&\quad [k_c : \mathtt{1}(X) \triangleright \mathbf{0} \mid h'_2 : l\langle \mathsf{roll}\ k' \rangle; k''] \mid k'' : \mathbf{0} \mid h'_3 : l(X) \triangleright X \mid M_R
\end{aligned}$$

which is a possible translation of $P \mid R_1$ as desired.

(R-Emb): the process R is simply inserted into M_R. No computation is performed in croll-π. □

We show now that all the reductions of $(\!|P|\!)^p$ in croll-π are related to the reductions of P. A main difference is that in croll-π all the processes can interact, while in TransCCS only processes inside the same transaction can. The lemma below shows that this constraint can always be verified by a suitable sequence of reductions derived using rule (R-Emb).

Lemma 12. *Let P be a TransCCS process. Given two process inside P there always is a sequence of reductions derived using rule (R-Emb) moving them inside the same transaction.*

Proof. If the two processes are already inside the same transaction then the thesis follows. Otherwise, take their common ancestor A in the hierarchy of transaction nesting. If A directly contains one of the two processes, move this process using (R-Emb) down in the hierarchy to the transaction containing the other process.

Otherwise take one of the child transactions of A that contains one of the processes, say P, and move this transaction inside the one containing the other process, say Q, via a sequence of applications of (R-Emb). Now we are back to the first case and we can embed Q into the transaction containing P. □

We can now prove the other direction of the relation between TransCCS process P and croll-π configurations in $(\!|P|\!)^p$.

Lemma 13. *Let P be a TransCCS process and $(\!|P|\!)^p$ one of its possible translations. If $(\!|P|\!)^p \longrightarrow M$ then $M \in (\!|P_1|\!)^p$ for some P_1 such that $P \longrightarrow P_1$ or $P = P_1$.*

Proof. The proof is by case analysis on the applied rule.

Let us consider rule (S.COM). We have two cases to consider: auxiliary communications on channels a, c or l according to the notation of Definition 11, or normal communications on channels that correspond to TransCCS channels.

If the communication is on a private channel a of some transaction then it is matched by P by staying idle, and we go from a possible translation of the first form in Definition 11 to one of the second form. If the communication is on a private channel c of some transaction then it is matched by P by staying idle, using closure under communications on c. If the communication is on channel l, it has to be followed by the corresponding roll, and the pair of reductions is matched by rule (R-Ab), using closure under communications on c. The behavioral correspondence is as described in the proof of Lemma 11.

The other possibility is that the communication is on a channel that corresponds to a channel in TransCCS. In this case the communication is matched by rule (R-Comm). For the rule to be applicable the two interacting terms should belong to the same transaction. We can always make this condition satisfied by executing a sequence of reductions derived from rule (R-Emb) as shown by Lemma 12. Note that applying these reductions does not change the set $(\!|P|\!)^p$, thus the behavioral correspondence is as described in the proof of Lemma 11.

Let us consider the case of rule (H-ROLL). This is never directly enabled in a possible translation, but it is always preceded by the corresponding communication at l, which has already been discussed above. □

We can now put the results together to prove Theorem 3.

(Reminder) Theorem 3. For each TransCCS process P, $P \; {}_t\!\approx_{c\pi} \nu k.\, k : (\!|P|\!)$.

Proof. We show that the relation:

$$\mathcal{R} = \{(P, M) \mid M \in (\!|P|\!)^p\}$$

is a weak barbed bisimulation.

Condition on barbs follows from Lemma 10. Conditions dealing with challenges from P are matched thanks to Lemma 11 while challenges from M are matched thanks to Lemma 13. □