Forward-Reverse Observational Equivalences in CCSK

Ivan Lanese, Iain Phillips

Forward-Reverse Observational Equivalences in CCSK*

Ivan Lanese[1][0000−0003−2527−9995] and Iain Phillips[2][0000−0001−5013−5876]

[1] Focus Team, University of Bologna/INRIA
[2] Imperial College London

Abstract. In the context of CCSK, a reversible extension of CCS, we study observational equivalences that distinguish forward moves from backward ones. We present a refinement of the notion of forward-reverse bisimilarity and show that it coincides with a notion of forward-reverse barbed congruence. We also show a set of sound axioms allowing one to reason equationally on process equivalences.

Keywords: CCSK, forward-reverse bisimilarity, forward-reverse barbed congruence, axiomatization

1 Introduction

Building concurrent systems is difficult and error-prone, since one has to reason on a large number of possible interleavings; yet concurrency is a must in current systems, such as the Internet, the Cloud, or parallel processors.

Reversible computing, allowing a system to execute both in the standard, forward direction, as well as backwards, recovering past states, has a number of interesting applications, including low-energy computing [18], simulation [6], biological modelling [5,33] and program debugging [11,24,21]. Many of these applications involve concurrent systems. Thus, a number of works have proposed reversible extensions of concurrent formalisms, including CCS [9,30], π-calculus [8], higher-order π [20], Petri nets [25], and the Erlang [22] programming language. Given the relevance of analysis techniques for concurrent systems, also a number of analysis techniques have been considered, e.g., following the session types approach [27,3,7]. Notions of observational equivalence have also been used in a few works, such as [30,26,1,2], yet the question of which notions of observational equivalence are suitable for reversible processes, and how they can be exploited to actually reason about them, has seldom been considered.

In this paper we tackle this issue. In particular, we consider a setting where reversibility is observable, that is forward actions are observationally distinguishable from their undo. This means that we are interested in systems where reversibility is relevant, e.g., recovery protocols, reversible debuggers, and so on.

* The first author has also been partially supported by French ANR project DCore ANR-18-CE25-0007 and by INdAM – GNCS 2020 project *Sistemi Reversibili Concorrenti: dai Modelli ai Linguaggi*. We thank reviewers for their helpful comments.

If instead one considers reversibility as an implementation detail only, then also more abstract notions where forward moves can be matched by backward moves and vice versa could be of interest. We leave this analysis for future work. Also, we consider *causal-consistent reversibility* [9], which is currently the most used notion of reversibility for concurrent systems. It states that any action can be undone, provided that its consequences, if any, are undone beforehand. Among its properties, it ensures that every reachable state is also reachable by forward computation. This contrasts with other approaches, used, e.g., in biological modelling [33], where causes can be undone without undoing their consequences, thus leading to new states. This setting will be left for future work as well. Finally, we will focus on strong equivalences, where internal steps need to be matched by other internal steps. Weak equivalences, which abstract away from internal steps, are of course also relevant, but before tackling them the strong setting needs to be well understood, hence we leave also this analysis for future work.

In the setting of causal-consistent reversibility, we define and motivate a notion of (strong) revised forward-reverse bisimilarity, also comparing it with alternative formulations (Section 4). We support our proposal with two contributions: (i) we show that it admits an equivalent formulation in terms of forward-reverse barbed congruence (Section 5); (ii) we prove sound a number of axioms which can be used to reason equationally about reversible systems (Theorem 4.10).

From a technical point of view, we work on CCSK, which is the instance on CCS of a general approach to reverse process calculi presented in [30]. The reason for this design choice is that CCSK history representation as keys ensures, differently from alternative approaches such as the one of RCCS [9], no redundancy in process representation, and this is fundamental to have simple axioms. We will come back to this point in the related work (Section 6), where we will also contrast our proposal with other proposals in the literature.

For space reasons, most proofs are available only in the companion technical report [23].

2 CCSK

As anticipated in the Introduction, we base our work on CCSK as defined in [30], with a minor change that we will discuss in Remark 4.2. We recall below the syntax and semantics of CCSK, referring to [30] for more details. CCSK is built on top of CCS, adding a mechanism of keys to keep track of which part of the process has already been executed. Hence, during forward computation, executed actions are not discarded, but just labelled as already been executed.

We define the actions of CCS, representing communications over named channels, much as usual. Let $\mathcal{A}$ be a set of *names*, ranged over by $a, b, \ldots$. Let $\overline{a}$ be the *complement* of $a \in \mathcal{A}$, let $\overline{\mathcal{A}} = \{\overline{a} : a \in \mathcal{A}\}$, and let $\mathtt{Act}$ be the disjoint union of $\mathcal{A}$, $\overline{\mathcal{A}}$ and $\{\tau\}$. Also, let $\overline{\overline{a}} = a$ for $a \in \mathcal{A} \cup \overline{\mathcal{A}}$. Standard prefixes, ranged over by $\alpha, \beta, \ldots$, are drawn from $\mathtt{Act}$. Intuitively, names represent input actions, co-names represent output actions, and τ is an internal synchronisation.

CCS processes, which we shall also call *standard* processes, are given by:

$$P, Q := 0 \mid \alpha.P \mid P + Q \mid P \,|\, Q \mid (\nu a)P$$

Intuitively, 0 is the inactive process, $\alpha.P$ is a process that performs action α and continues as P, $P + Q$ is nondeterministic choice, $P \,|\, Q$ is parallel composition and restriction $(\nu a)P$ binds channel a (both occurrences in input actions and in output actions) inside P. We do not consider renaming, adding it will not change the results of the paper. A channel is bound if inside the scope of a restriction operator, free otherwise. Function $\texttt{fn}(P)$ computes the set of free names in process P.

CCSK adds to CCS the possibility of going backwards. In order to remember which input interacted with which output while going forward, at each forward step fresh keys are created, and the same key is used to label an input and the corresponding output.

We denote the set of keys by $\texttt{Keys}$, ranged over by $m, n, k, \ldots$. Prefixes, ranged over by π, are of the form $\alpha[m]$ or α. The former denotes that α has already been executed, the latter that it has not.

CCSK processes are given by:

$$X, Y := 0 \mid \pi.X \mid X + Y \mid X \,|\, Y \mid (\nu a)X$$

hence they are like CCS processes but for the fact that some prefixes may be labelled with a key. In the following, we will drop trailing 0s.

We use predicate $\texttt{std}(X)$ to mean that X is standard, that is none of its actions has been executed, hence it has no keys. We assume function $\texttt{toStd}(X)$ that takes a CCSK process X and gives back the standard process obtained by removing from X all keys. Finally, we consider function $\texttt{keys}(X)$ which computes the set of keys in CCSK process X.

In a CCSK process we distinguish free from bound keys.

Definition 2.1 (Free and bound keys). *A key k is* bound *in a process X iff it occurs either twice, attached to complementary prefixes, or once, attached to a τ prefix. A key k is* free *if it occurs once, attached to a non-τ prefix.*

Intuitively, all the occurrences of a bound key are inside the process, while a free key has another occurrence in the environment.

Figure 1 shows the forward rules of CCSK. Backward rules in Figure 2 are obtained from forward rules by reversing the direction of transitions.

Rule (TOP) allows a prefix to execute. The rule generates a key m. Freshness of m is guaranteed by the side conditions of the other rules. Rule (PREFIX) states that an executed prefix does not block execution. The two rules for (CHOICE) and the two for (PAR) allow processes to execute inside a choice or a parallel composition. The side condition of rule (CHOICE) ensures that at most one branch can execute. Rule (SYNCH) allows two complementary actions to synchronise producing a τ. The key of the two actions needs to be the same. Rule (RES) allows an action which does not involve the restricted channel to propagate through restriction.

$$\text{(TOP)}\ \frac{\mathtt{std}(X)}{\alpha.X \xrightarrow{\alpha[m]}_f \alpha[m].X} \qquad \text{(PREFIX)}\ \frac{X \xrightarrow{\beta[n]}_f X'}{\alpha[m].X \xrightarrow{\beta[n]}_f \alpha[m].X'}\ m \neq n$$

$$\text{(CHOICE)}\ \frac{X \xrightarrow{\alpha[m]}_f X' \quad \mathtt{std}(Y)}{X + Y \xrightarrow{\alpha[m]}_f X' + Y} \qquad \frac{Y \xrightarrow{\alpha[m]}_f Y' \quad \mathtt{std}(X)}{X + Y \xrightarrow{\alpha[m]}_f X + Y'}$$

$$\text{(PAR)}\ \frac{X \xrightarrow{\alpha[m]}_f X' \quad m \notin \mathtt{keys}(Y)}{X \,|\, Y \xrightarrow{\alpha[m]}_f X' \,|\, Y} \qquad \frac{Y \xrightarrow{\alpha[m]}_f Y' \quad m \notin \mathtt{keys}(X)}{X \,|\, Y \xrightarrow{\alpha[m]}_f X \,|\, Y'}$$

$$\text{(SYNCH)}\ \frac{X \xrightarrow{\alpha[m]}_f X' \quad Y \xrightarrow{\overline{\alpha}[m]}_f Y'}{X \,|\, Y \xrightarrow{\tau[m]}_f X' \,|\, Y'}\ (\alpha \neq \tau)$$

$$\text{(RES)}\ \frac{X \xrightarrow{\alpha[m]}_f X'}{(\nu a)X \xrightarrow{\alpha[m]}_f (\nu a)X'}\ \alpha \notin \{a, \overline{a}\}$$

Fig. 1. Forward SOS rules for CCSK

The forward semantics of a CCSK process is the smallest relation $\rightarrow_f$ closed under the rules in Figure 1. The backward semantics of a CCSK process is the smallest relation $\rightarrow_r$ closed under the rules in Figure 2. The semantics is the union of the two relations. We shall let μ range over transition labels $\alpha[m]$.

As standard in reversible computing (see, e.g., [30] or the notion of coherent process in [9]), all the developments consider only processes reachable from a standard process.

Definition 2.2 (Reachable process). *A process X is* reachable *iff there exists a standard process P and a finite sequence of transitions from P to X.*

We recall here a main result of reversible computing, useful for our development: the Loop Lemma states that any action can be undone, and any undone action can be redone.

Lemma 2.3 (Loop Lemma [30, Prop. 5.1]). *$X \xrightarrow{\alpha[m]}_f X'$ iff $X' \xrightarrow{\alpha[m]}_r X$.*

3 Syntactic Characterisation of Reachable Processes

In this paper we discuss relevant notions of observational equivalence for CCSK. However, since, as described above, only reachable processes are of interest, we need to understand the structure of reachable processes. Hence, as a preliminary step, we propose in this section a sound and complete syntactic characterisation of reachable processes. This result is interesting in itself, since a similar preliminary step is needed for most of the reasonings on a calculus such as CCSK. Notably, many works in the literature study properties that reachable processes

$$\text{(BK-TOP)}\quad \frac{\mathtt{std}(X)}{\alpha[m].X \xrightarrow{\alpha[m]}_r \alpha.X} \qquad \text{(BK-PREFIX)}\quad \frac{X \xrightarrow{\beta[n]}_r X'}{\alpha[m].X \xrightarrow{\beta[n]}_r \alpha[m].X'}\ m \neq n$$

$$\text{(BK-CHOICE)}\quad \frac{X \xrightarrow{\alpha[m]}_r X' \quad \mathtt{std}(Y)}{X + Y \xrightarrow{\alpha[m]}_r X' + Y} \qquad \frac{Y \xrightarrow{\alpha[m]}_r Y' \quad \mathtt{std}(X)}{X + Y \xrightarrow{\alpha[m]}_r X + Y'}$$

$$\text{(BK-PAR)}\quad \frac{X \xrightarrow{\alpha[m]}_r X' \quad m \notin \mathtt{keys}(Y)}{X \,|\, Y \xrightarrow{\alpha[m]}_r X' \,|\, Y} \qquad \frac{Y \xrightarrow{\alpha[m]}_r Y' \quad m \notin \mathtt{keys}(X)}{X \,|\, Y \xrightarrow{\alpha[m]}_r X \,|\, Y'}$$

$$\text{(BK-SYNCH)}\quad \frac{X \xrightarrow{\alpha[m]}_r X' \quad Y \xrightarrow{\overline{\alpha}[m]}_r Y'}{X \,|\, Y \xrightarrow{\tau[m]}_r X' \,|\, Y'}\ (\alpha \neq \tau)$$

$$\text{(BK-RES)}\quad \frac{X \xrightarrow{\alpha[m]}_r X'}{(\nu a)X \xrightarrow{\alpha[m]}_r (\nu a)X'}\ \alpha \notin \{a, \overline{a}\}$$

Fig. 2. Reverse SOS rules for CCSK

satisfy, but they do not provide a complete characterisation, see, e.g., [20, Lemma 3]. We are not aware of any such characterisation for reversible CCS.

A main item in the characterisation of reachability is that keys should define a partial order, thus given a process X we define a partial order $\leq_X$ on $\mathtt{keys}(X)$ as follows.

Definition 3.1 (Partial order on keys). *We first define a function* $\mathtt{ord}(\cdot)$ *that given a process computes a set of relations among its keys. The definition is by structural induction on* X*.*

$$\begin{aligned} \mathtt{ord}(0) &= \emptyset & \mathtt{ord}(\alpha.X) &= \mathtt{ord}(X) \\ \mathtt{ord}(X + Y) &= \mathtt{ord}(X) \cup \mathtt{ord}(Y) & \mathtt{ord}(X \,|\, Y) &= \mathtt{ord}(X) \cup \mathtt{ord}(Y) \\ \mathtt{ord}(\alpha[n].X) &= \mathtt{ord}(X) \cup \{n < k | k \in \mathtt{keys}(X)\} \\ \mathtt{ord}((\nu a)X) &= \mathtt{ord}(X) \end{aligned}$$

The partial order $\leq_X$ *on* $\mathtt{keys}(X)$ *is the reflexive and transitive closure of* $\mathtt{ord}(X)$*.*

The definition above captures structural dependencies on keys. Combined with the property that prefixes that interact have the same key (see rule (SYNCH) in Figure 1), it captures a form of Lamport's happened-before relation [17] adapted to synchronous communication.

In order to characterise reachable processes we need to define contexts.

Definition 3.2 (Context). *A CCSK* context *is a process with a hole, as generated by the grammar below:*

$$C := \bullet \mid \pi.C \mid C + Y \mid X + C \mid C \,|\, Y \mid X \,|\, C \mid (\nu a)C$$

We denote with $C[X]$ *the process obtained by replacing* $\bullet$ *with* X *inside* C*.*

We can now present our characterisation of reachable processes.

Definition 3.3 (Well-formed process). *A process X is* well-formed *if all the following conditions hold:*

1. *if $X = C[\alpha.Y]$ then $\mathtt{std}(Y)$, that is standard prefixes have standard continuations;*
2. *if $X = C[Y + Y']$ then $\mathtt{std}(Y) \vee \mathtt{std}(Y')$, that is in choices at most one branch is non-standard;*
3. *each key occurs at most twice;*
4. *if a key occurs twice, then the two occurrences are attached to complementary prefixes;*
5. *if a key n occurs twice, then there are C, Y, Y' such that $X = C[Y \mid Y']$, and both Y and Y' contain exactly one occurrence of n;*
6. *if $X = C[(\nu a)Y]$ and a key n occurs in a prefix on channel a inside Y, then n occurs twice inside Y;*
7. *$\leq_X$ is acyclic.*

Thanks to conditions 3 and 4 each key is either free or bound.
Each well-formed process X is reachable from $\mathtt{toStd}(X)$.

Lemma 3.4. *Let X be a well-formed process. Let $k_1, \ldots, k_n$ be a fixed total order on $\mathtt{keys}(X)$ compatible with $\leq_X$. Then there is a computation $\mathtt{toStd}(X) \xrightarrow{\alpha[k_1]}_f \ldots \xrightarrow{\alpha[k_n]}_f X$.*

The classes of reachable and well-formed processes coincide.

Proposition 3.5 (Reachable coincides with well-formed). *X is reachable iff it is well-formed.*

The results above, enabled by the syntactic characterisation of reachable processes, are needed for our development, but can also help in general in the study of the theory of CCSK. Indeed, we can for instance derive as a corollary the Parabolic Lemma [30, Lemma 5.12], which can be rephrased as follows.

Corollary 3.6 (Parabolic Lemma [30, Lemma 5.12]). *Each reachable process is forward reachable.*

Proof. Thanks to Proposition 3.5 a reachable process is also well-formed. Hence, it is forward reachable thanks to Lemma 3.4.

4 Revised Forward-Reverse Bisimilarity

In this section we move to the main aim of this work, namely the definition of a strong observational equivalence for CCSK able to distinguish forward steps from backward ones. We consider as starting point for our analysis the notion of forward-reverse bisimulation introduced in the original CCSK paper [30, Definition 6.5], and rephrased below.

Definition 4.1 (Forward-reverse bisimulation). *A symmetric relation $\mathcal{R}$ is a* forward-reverse bisimulation *if whenever $X \mathcal{R} Y$:*

1. $\texttt{keys}(X) = \texttt{keys}(Y)$;
2. *if* $X \xrightarrow{\mu}_f X'$ *then there is* Y' *such that* $Y \xrightarrow{\mu}_f Y'$ *and* $X' \mathcal{R} Y'$;
3. *if* $X \xrightarrow{\mu}_r X'$ *then there is* Y' *such that* $Y \xrightarrow{\mu}_r Y'$ *and* $X' \mathcal{R} Y'$.

We first notice that clause 1 is redundant. Indeed, given a reachable process X we know that there is a computation $\texttt{toStd}(X) \rightarrow_f^* X$. Thanks to the Loop Lemma, we also have a computation $X \rightarrow_r^* \texttt{toStd}(X)$, whose labels exhibit all the keys in $\texttt{keys}(X)$. Thus, any process Y related to X by the bisimulation should match these labels because of clause 3, hence it should have at least the same keys. By symmetry, it should actually have exactly the same keys.

However, we claim that requiring to match all keys in the labels, as done implicitly by clause 3, is too demanding. The intuition under this claim is that keys are only a technical means to:

1. distinguishing executed from non-executed prefixes, and the choice of the key is irrelevant for this purpose;
2. coupling together prefixes that synchronised (see rule (SYNCH) in Figure 1): in this case the choice of the key is irrelevant as well, but it is important whether two occurrences refer to the same key (highlighting a performed synchronisation) or to different keys (denoting independent actions).

As for item 2, keys can be safely α-renamed provided that *all* the occurrences of a given key are renamed in the same way. Now, keys are linear in the sense of [15], that is each key can occur at most twice because of condition 3 in the characterisation of reachable processes (Definition 3.3, see also Proposition 3.5). One can think of an occurrence attached to a τ prefix as two occurrences attached to complementary actions. Now, a key k is bound (cfr. Definition 2.1) if both its occurrences are in the considered process, hence k can be safely α-renamed without affecting the context. For instance, we want to equate $\overline{a}[n] \,|\, a[n]$ and $\overline{a}[m] \,|\, a[m]$. If instead a key k is free in a process, then the other occurrence can appear in the context: here α-equivalence needs to be disallowed to ensure compositionality. Indeed, we cannot equate $\overline{a}[n]$ and $\overline{a}[m]$, since in a context such as $\cdot \,|\, a[n]$ the former needs to synchronise to go back, while the latter does not.

We decided to embed α-conversion of bound keys in the semantics. Given our choice of not observing bound keys, this is needed for compositionality reasons. Indeed, otherwise one could distinguish processes such as $\overline{a}[n] \,|\, a[n] \,|\, b$ and $\overline{a}[m] \,|\, a[m] \,|\, b$ since the former can take a forward transition with label $b[m]$, while the latter can not, due to the side condition of rule (PAR). Thanks to α-conversion, the two processes can both take a forward transition with label $b[m]$, provided that the latter first α-converts m to a different key.

We will come back to this issue in the discussion after Proposition 4.9, showing that dropping α-conversion of bound keys would break congruence of bisimilarity. Note that such an issue does not occur in [15], since they never create new linear prefixes, while we do create new keys.

In the light of the discussion above, we extend the semantics of CCSK with a structural equivalence, defined as the smallest equivalence relation (that is, reflexive, symmetric, and transitive relation) closed under the following rule:

$$X \equiv X[n/m] \qquad m \text{ bound in } X, n \notin \mathtt{keys}(X)$$

where $[n/m]$ denotes the substitution of all the occurrences of key m with key n. Notice that structural equivalence preserves well-formedness (this can be checked by inspection of the conditions in Definition 3.3), but it is not a congruence, since, e.g., $\tau[n] \equiv \tau[m]$ but $\tau[n].\tau[n] \not\equiv \tau[n].\tau[m]$. To avoid this issue we would need explicit binders for keys, but we prefer to avoid them to stay as close as possible to the original CCSK.

We also need to introduce rules enabling the use of structural equivalence in the derivation of transitions:

$$\text{(EQUIV)} \quad \frac{Y \equiv X \quad X \xrightarrow{\alpha[m]}_f X' \quad X' \equiv Y'}{Y \xrightarrow{\alpha[m]}_f Y'}$$

$$\text{(BK-EQUIV)} \quad \frac{Y \equiv X \quad X \xrightarrow{\alpha[m]}_r X' \quad X' \equiv Y'}{Y \xrightarrow{\alpha[m]}_r Y'}$$

Due to the problem above, these rules can only be used *as last rules in a derivation.*

Remark 4.2. The introduction of structural equivalence and of the related rules is the only difference between our semantics of CCSK and the one in [30]. Note that with a little abuse of notation from now on arrows $\xrightarrow{\mu}_f$, $\xrightarrow{\mu}_r$ and $\xrightarrow{\mu}$ refer to the semantics which includes structural equivalence.

Notice that the change to the semantics has no impact on well-formedness and on Proposition 3.5.

Thanks to structural equivalence we can show that keys attached to τ labels are irrelevant.

Proposition 4.3. *For each $X \xrightarrow{\tau[m]} X'$ and n not free in X we have $X \xrightarrow{\tau[n]} X''$ with $X'' \equiv X'$.*

Proof. The proof is by induction on the derivation of $X \xrightarrow{\tau[m]} X'$. The induction is trivial if $n \notin \mathtt{keys}(X)$. If n is bound then one can apply rule (EQUIV) or (BK-EQUIV) to first convert n to any fresh key. □

Actually, by applying rule (EQUIV) or (BK-EQUIV) one can also obtain $X'' = X'$.

Given the result above, one could even use just label τ instead of $\tau[m]$. We prefer however not to do it so that all the labels feature a key. This is handy when writing rules such as (RES).

We can revise the notion of forward-reverse bisimulation (Definition 4.1) as follows:

Definition 4.4 (Revised forward-reverse bisimulation). *A symmetric relation $\mathcal{R}$ is a* revised forward-reverse bisimulation *(revised FR bisimulation for short) if whenever $X\ \mathcal{R}\ Y$:*

1. *if $X \xrightarrow{\mu}_f X'$ then there is Y' such that $Y \xrightarrow{\mu}_f Y'$ and $X'\ \mathcal{R}\ Y'$;*
2. *if $X \xrightarrow{\mu}_r X'$ then there is Y' such that $Y \xrightarrow{\mu}_r Y'$ and $X'\ \mathcal{R}\ Y'$.*

The revised forward-reverse (FR) bisimilarity*, denoted as $\sim_b$, is the largest revised FR bisimulation.*

For uniformity, such a definition requires to match bound keys in labels, however thanks to structural equivalence and Proposition 4.3 this does not allow one to distinguish processes that only differ on bound keys. Indeed, structural equivalence is a bisimulation.

Proposition 4.5 (Structural equivalence is a bisimulation). *$X \equiv Y$ implies $X \sim_b Y$.*

Proof. The thesis follows by coinduction, using Proposition 4.3 to match labels whose keys have been α-converted (and, as such, are bound, and can only be attached to τ labels). □

We now use revised FR bisimilarity to show on an example that, as expected, bound keys are not observable.

Example 4.6. We show that $\overline{a}[n] \,|\, a[n] \sim_b \overline{a}[m] \,|\, a[m]$. To this end we need to show that the relation:

$$R = \{(\overline{a}[n] \,|\, a[n], \overline{a}[m] \,|\, a[m])\} \cup \mathtt{Id}$$

where $\mathtt{Id}$ is the identity relation on reachable processes is a bisimulation.

This is trivial since the only possible transition is a backward $\tau[k]$ for any key k (using rule (BK-EQUIV)), leading to $\overline{a} \,|\, a$ on both the sides. Any further transition can be matched so to remain in the identity relation $\mathtt{Id}$. Notice that one can obtain the same result directly from Proposition 4.5.

We now show some properties of revised FR bisimilarity. They will also be useful in the next section, to prove a characterisation of revised FR bisimilarity as a barbed congruence.

First, two equivalent processes are either both standard or both non-standard.

Lemma 4.7. *If $X \sim_b Y$ then $\mathtt{std}(X)$ iff $\mathtt{std}(Y)$.*

Proof. A process is non-standard iff it can perform backward transitions. The thesis follows. □

Also, equivalent processes have the same set of free keys.

Lemma 4.8. *If $X \sim_b Y$ and key n is free in X then key n is free in Y as well.*

$$
\begin{array}{rll}
X \mid Y = Y \mid X & & \text{(PAR-COMM)} \\
X \mid (Y \mid Z) = (X \mid Y) \mid Z & & \text{(PAR-ASS)} \\
X \mid 0 = X & & \text{(PAR-UNIT)} \\
\\
X + Y = Y + X & & \text{(CH-COMM)} \\
X + (Y + Z) = (X + Y) + X & & \text{(CH-ASS)} \\
X + 0 = X & & \text{(CH-UNIT)} \\
X + P = X & \text{iff } \mathtt{toStd}(X) = P & \text{(CH-IDEM)} \\
\\
(\nu a)(\nu b)X = (\nu b)(\nu a)X & & \text{(RES-COMM)} \\
(\nu a)(X \mid Y) = X \mid (\nu a)Y & \text{iff } a \notin \mathtt{fn}(X) & \text{(RES-PAR)} \\
(\nu a)(X + Y) = ((\nu a)X) + ((\nu a)Y) & & \text{(RES-CH)} \\
(\nu a)\pi.X = \pi.(\nu a)X & \text{iff } a \notin \mathtt{fn}(\pi) & \text{(RES-PREF)} \\
(\nu a)0 = 0 & & \text{(RES-DROP)} \\
(\nu a)\alpha.P = 0 & \text{iff } a \in \mathtt{fn}(\alpha) & \text{(RES-LOCK)} \\
(\nu a)X = (\nu b)X[b/a] & \text{iff } b \notin \mathtt{fn}(X) & \text{(RES-ALPH)}
\end{array}
$$

Fig. 3. CCSK axiomatisation

We now show that revised FR bisimilarity is a congruence.

Proposition 4.9 (Revised FR bisimilarity is a congruence). *$X \sim_b Y$ implies $C[X] \sim_b C[Y]$ for each C such that $C[X]$ and $C[Y]$ are both reachable.*

Note that α-conversion of bound keys is needed to prove congruence w.r.t. parallel composition. Otherwise, processes $\overline{a}[n] \mid a[n]$ and $\overline{a}[m] \mid a[m]$ would be distinguished by a parallel context performing a transition creating a new key m, since this would be allowed only in the first case. Thanks to α-conversion, this is possible in both the cases, by first α-converting m in $\overline{a}[m] \mid a[m]$ to a different key.

We now study the axiomatisation of revised FR bisimilarity. While we are not able to provide a sound and complete axiomatisation, we can prove sound a number of relevant axioms. These allow one to reason equationally on CCSK processes.

We consider the list of axioms in Figure 3. Most axioms are standard CCS axioms [28], extended to deal with non-standard prefixes. There are however a few interesting differences. E.g., we notice the non-standard axiom (CH-IDEM). Indeed, the left-hand side of the standard axiom $X + X = X$ is not reachable if X is not standard, and $X + X = X$ is an instance of (CH-IDEM) if X is standard. We also note that in rule (RES-LOCK) replacing α with π would be useless since the resulting process would not be reachable.

Theorem 4.10. *The axioms in Figure 3 are sound w.r.t. revised FR bisimilarity.*

$$(\nu a)(\overline{a}.P \mid a.Q) = \tau.(\nu a)(P \mid Q) \qquad \text{(RES-EXP)}$$
$$(\nu a)(\overline{a}[n].X \mid a[n].Y) = \tau[n].(\nu a)(X \mid Y) \qquad \text{(RES-EXP-BK)}$$
$$\tau \mid \tau = \tau.\tau \qquad \text{(TAU-EXP)}$$
$$\tau[n] \mid \tau = \tau[n].\tau \qquad \text{(TAU-EXP-BF)}$$
$$\tau[n] \mid \tau[m] = \tau[n].\tau[m] \qquad \text{(TAU-EXP-BK)}$$

Fig. 4. Sample instances of the Expansion Law

Proof. For each axiom, instances of the axiom form a bisimulation, hence the thesis follows. □

Probably the most relevant axiom in CCS theory is the Expansion Law [28], which can be written as follows:

$$P_1 \mid P_2 = \sum\{\alpha.(P_1' \mid P_2) : P_1 \xrightarrow{\alpha} P_1'\} + \sum\{\alpha.(P_1 \mid P_2') : P_2 \xrightarrow{\alpha} P_2'\} + \sum\{\tau.(P_1' \mid P_2') : P_1 \xrightarrow{\alpha} P_1', P_2 \xrightarrow{\overline{\alpha}} P_2'\}$$

where $\sum$ is n-ary choice.

It is well-known [30] that the Expansion Law does not hold for reversible calculi, and indeed we can falsify it on an example using revised FR bisimilarity.

Example 4.11. We show here that $a \mid b \not\sim_b a.b + b.a$. Indeed, $a \mid b$ can take forward actions $a[n]$, $b[m]$, and then undo $a[n]$, while $a.b + b.a$ cannot. Notably, also $a \mid a \not\sim_b a.a$, since $a \mid a$ can take forward actions $a[n]$, $a[m]$, and then undo $a[n]$, while $a.a$ cannot.

This shows that classical CCS bisimilarity [28] is not as distinguishing, on CCS processes, as revised FR bisimilarity. Indeed, as expected, revised FR bisimilarity is strictly finer.

Proposition 4.12. *$P \sim_b Q$ implies $P \sim Q$ where $\sim$ is classical CCS bisimilarity [28], while the opposite implication does not hold.*

Proof. Classical CCS bisimilarity corresponds to clause 1 in the definition of revised FR bisimilarity. The failure of the other inclusion follows from Example 4.11. □

We show below that even if the full Expansion Law does not hold, a few instances indeed hold.

Proposition 4.13. *The instances of the Expansion Law in Figure 4 are sound w.r.t. revised FR bisimilarity.*

Proof. Instances of axioms (RES-EXP) and (RES-EXP-BK) form a revised FR bisimulation. The last three axioms form a revised FR bisimulation as well. □

5 Forward-Reverse Barbed Congruence

In order to further justify the definition of revised FR bisimilarity, we show it to be equivalent to a form of forward-reverse barbed congruence.

As classically [14,34], an (open) barbed congruence is obtained by defining a set of basic observables, called *barbs*, and then taking the smallest equivalence on processes closed under contexts and under reduction requiring equivalent processes to have the same barbs. In our setting of course reductions can be both forwards and backwards. We start by defining barbs.

Definition 5.1 (Barbs). *A process X has a* forward output barb *at a, written $\downarrow_{\overline{a}}$, iff $X \xrightarrow{\overline{a}[n]}_f X'$ for some n and X'. A process X has a* backward (input or output) barb *at $\alpha[n]$ ($\alpha \neq \tau$), written $\uparrow_{\alpha[n]}$, iff $X \xrightarrow{\alpha[n]}_r X'$ for some X'.*

We notice some asymmetry in the definition. The issue here is that we want to distinguish processes such as $a[n]$ and $b[n]$ (or $\overline{a}[n]$ and $\overline{b}[n]$), but there is no context undoing any of the prefixes such that both the compositions are reachable. Hence, we need (backward) barbs distinguishing them. This issue does not occur in forward transitions. Indeed, allowing one to observe also forward input barbs or keys in forward barbs would not change the observational power.

We can now formalise the notion of forward-reverse barbed congruence.

Definition 5.2 (Forward-reverse barbed congruence). *A symmetric relation $\mathcal{R}$ is a* forward-reverse (FR) barbed bisimulation *if whenever $X \mathcal{R} Y$:*

- *$X \downarrow_{\overline{a}}$ implies $Y \downarrow_{\overline{a}}$;*
- *$X \uparrow_{\alpha[n]}$ implies $Y \uparrow_{\alpha[n]}$;*
- *if $X \xrightarrow{\tau[n]}_f X'$ then there is Y' such that $Y \xrightarrow{\tau[n]}_f Y'$ and $X' \mathcal{R} Y'$;*
- *if $X \xrightarrow{\tau[n]}_r X'$ then there is Y' such that $Y \xrightarrow{\tau[n]}_r Y'$ and $X' \mathcal{R} Y'$.*

Forward-reverse (FR) barbed bisimilarity*, denoted as $\sim_{bb}$, is the largest FR barbed bisimulation. A* forward-reverse (FR) barbed congruence *is a FR barbed bisimulation such that $X \mathcal{R} Y$ implies $C[X] \mathcal{R} C[Y]$ for each C such that $C[X]$ and $C[Y]$ are both reachable. We denote as $\sim_c$ the largest FR barbed congruence.*

As discussed above, we sometimes need barbs to require to match free keys, e.g., to distinguish $a[n]$ from $b[n]$. Indeed, in this case there exists no context able to force the match such that both the compositions are reachable. Such a context would need to include occurrences of both $\overline{a}[n]$ and $\overline{b}[n]$, but then any composition involving such a context would not be reachable.

However, when such a context exists, processes which differ for free keys only can be distinguished without the need for barbs, as shown by the example below.

Example 5.3. We show that $\overline{a}[n] \not\sim_c \overline{a}[m]$ without relying on barbs. Indeed, if we put the two processes in the context $\bullet \mid a[n]$ they behave differently:

$$\overline{a}[n] \mid a[n] \xrightarrow{\tau[n]}_r \overline{a} \mid a \qquad \text{while} \qquad \overline{a}[m] \mid a[n]$$

cannot perform any backward τ move. We remark that both the processes above are reachable.

However, bound keys are not observable, as shown in Example 4.6 for revised FR bisimilarity. We will show that this holds as well for FR barbed congruence, since actually revised FR bisimilarity coincides with FR barbed congruence (Corollary 5.12). This can be seen as a justification of our choice of revised FR bisimilarity. Revised FR bisimilarity, as usual for bisimilarities, is easier to work with since it has no universal quantification over contexts. For instance, it allowed us to easily prove the equivalence in Example 4.6.

First, revised FR barbed bisimilarity implies FR barbed congruence.

Theorem 5.4. *$X \sim_b Y$ implies $X \sim_c Y$.*

Proof. It is trivial to show that $\sim_b$ is a FR barbed bisimulation. Congruence follows from Proposition 4.9. □

We now move towards the proof of the other inclusion in the equivalence between revised FR bisimilarity and FR barbed congruence.

To this end, we introduce below the notion of fresh process.

Definition 5.5 (Fresh process). *Given a process X, the corresponding* fresh process $\texttt{toFresh}(X)$ *is inductively defined as follows:*

$$\begin{aligned}
\texttt{toFresh}(0) &= 0\\
\texttt{toFresh}(\pi.X) &= f_i^p.0 + \pi.(f_i^s.0 + X)\\
\texttt{toFresh}(X + Y) &= \texttt{toFresh}(X) + \texttt{toFresh}(Y)\\
\texttt{toFresh}(X \mid Y) &= \texttt{toFresh}(X) \mid \texttt{toFresh}(Y)\\
\texttt{toFresh}((\nu a)X) &= (\nu a)\texttt{toFresh}(X)
\end{aligned}$$

where, in the clause for prefix, i is the number of prefixes before π in a pre-visit of the syntax tree of X. We call f_i^p (where p stands for previous) and f_i^s (where s stands for subsequent) fresh names and the corresponding prefixes fresh prefixes.

A process is fresh if it is obtained by applying function $\texttt{toFresh}(\cdot)$ to some reachable process X.

Note that all fresh names are pairwise different. Indeed the use of a pre-visit of the syntax tree is just a way to generate fixed, pairwise different fresh names for each prefix; any other algorithm with the same property would be fine as well.

Example 5.6 (Function $\texttt{toFresh}(\cdot)$). Consider the process $X = a.b.0 \mid \overline{a}.b.0$. We have $\texttt{toFresh}(X) = f_0^p.0 + a.(f_0^s.0 + f_1^p.0 + b.(f_1^s.0 + 0)) \mid f_2^p.0 + \overline{a}.(f_2^s.0 + f_3^p.0 + b.(f_3^s.0 + 0))$. Note that, e.g., executing a will disable the forward barb at f_0^p and enable the one at f_0^s.

Fresh processes are closed under reductions not involving fresh names.

Lemma 5.7. *$X \xrightarrow{\mu} Y$ iff $\texttt{toFresh}(X) \xrightarrow{\mu} \texttt{toFresh}(Y)$ and μ does not contain a fresh name.*

Fresh processes obtained from reachable processes are reachable.

Lemma 5.8. *Let X be a reachable process. Then $\texttt{toFresh}(X)$ is reachable.*

Proof. By inspection of the conditions in Definition 3.3. □

Intuitively, in a fresh process one is always able to distinguish which prefix of the original process has been done or undone by checking which barbs have been enabled or disabled. Indeed, using the names in the definition, performing the i-th prefix enables the barb at f_i^s and disables the one at f_i^p, while undoing π does the opposite.

This is formalised by the following lemma.

Lemma 5.9. *If W is fresh, $W \xrightarrow{\mu'} W'$ and $W \xrightarrow{\mu''} W''$, and W' and W'' have the same barbs then there exists a substitution σ on keys such that $\mu' = \mu''\sigma$ and $W' = W''\sigma$. The substitution is the identity if the transition is backwards.*

Next proposition shows that to check barbed congruence it is enough to consider contexts which are parallel compositions with fresh processes.

Proposition 5.10. *The relation*

$$\mathcal{R} = \{(C[X], C[Y]) \mid W \mid X \sim_{bb} W \mid Y \;\; \forall\, W \text{ fresh, } C \text{ context such that } C[X], C[Y], W \mid X, W \mid Y \text{ are all reachable}\}$$

is a barbed congruence.

We can now show that FR barbed congruence implies revised FR bisimilarity, hence the two notions coincide.

Theorem 5.11. *The relation $\mathcal{R} = \{(X, Y) \mid X \sim_c Y\}$ is a revised FR bisimulation.*

Corollary 5.12. *$\sim_c$ and $\sim_b$ coincide.*

Proof. By composing Theorem 5.4 and Theorem 5.11. □

6 Related Work

In [19], CCSK has been proved equivalent to RCCS [9], the first reversible process calculus, thus, in principle, our results apply to RCCS as well. However, the mechanism of memories used in RCCS introduces much more redundancy than the key mechanism used in CCSK, hence a direct application of our results to RCCS is not easy. Let us take as example [9, page 299] the process

$$R = \langle\langle 2\rangle, a, 0\rangle \cdot \langle 1\rangle \triangleright 0 \mid \langle\langle 1\rangle, \overline{a}, 0\rangle \cdot \langle 2\rangle \triangleright 0$$

obtained by performing the synchronisation from $\langle\rangle \triangleright (a.0 \mid \overline{a}.0)$. To apply commutativity of parallel composition to R we need not only to swap the two subprocesses, but also to exchange all the occurrences of 1 with 2 in the memories, otherwise the obtained process would not be reachable. This particular issue has been solved in successive works on RCCS [16], yet similar issues remain since, e.g., the history of a thread is duplicated every time a process forks, hence the application of an axiom may impact many subprocesses which are possibly syntactically far away. In such a setting it is not clear how to write axioms.

We now discuss various works which focus on observational equivalences with backward transitions. The seminal paper [10] discusses bisimulations with reverse transitions much before causal-consistent reversibility and CCSK or RCCS were introduced, yet it briefly anticipates the notion of causal-consistent reversibility but discards it in favour of a total order of actions. In this case the bisimilarity coincides with CCS bisimilarity. A total order of actions is also considered in in [26]. They study testing equivalences in a CCS with rollback and focus on finding easier characterisations of testing semantics. As such, their results are not directly related to ours.

It is shown in [29] that the process graphs of CCSK processes are so-called prime graphs, which correspond to prime event structures. Due to the keys these structures are 'non-repeating', i.e. no two events in a configuration can have the same label. It is further shown [29, Theorem 5.4] that FR bisimilarity corresponds to hereditary history-preserving bisimilarity [4,12] on non-repeating prime event structures.

Equivalences for reversible CCS are also studied in [1,2] via encodings in configuration structures. Their main aim is to show that the induced equivalence on CCS coincides with hereditary history-preserving bisimilarity. Differently from us, they work on RCCS instead of CCSK, and, unsurprisingly given the discussion above, they do not consider axiomatisations. In [2] no notion of barbed congruence is considered, and their notions of bisimulations are triples instead of pairs as in our case, since they additionally need to keep track of a bijection among identifiers (which are close to our keys). Barbed congruence is considered in [1], but they only allow for contexts which are coherent on their own, thus disallowing interactions between the context and the process in the hole when going backwards. This has a relevant impact on the theory, as discussed in [1] itself (see [1, Example 3]). Also, they consider only processes without auto-concurrency and without auto-conflict [1, Remark 1].

A hierarchy of equivalences with backward transitions, including hereditary history-preserving bisimilarity, was studied in the context of stable configuration structures in [31]. A logic with reverse as well as forward modalities which characterises hereditary history-preserving bisimilarity was introduced in [32].

CCSK has been given a denotational semantics using reversible bundle event structures in [13]. Although equivalences are not discussed there, this opens the way to defining an equivalence on CCSK processes based on an equivalence between their denotations as reversible event structures.

7 Conclusion and Future Work

We have discussed strong observational equivalences able to distinguish forward and backward moves in CCSK. As shown on many occasions, a main difference w.r.t. the theory of CCS is that in CCSK not all processes are reachable, and this limits what contexts can observe. This motivates, e.g., the use of backward barbs which are more fine-grained than forward ones in the definition of FR barbed congruence. As anticipated in the Introduction, other forms of observational equivalences, such as weak equivalences, are also of interest.

Other interesting directions for future work include comparing the equivalence that our definitions induce on standard processes with known CCS equivalences. We have shown that revised FR bisimilarity is finer than standard CCS bisimilarity, and we conjecture it to be equivalent to hereditary history-preserving bisimilarity [12], in line with the results in [29,1,2].

Finally, we have discussed how axiomatisation is easier in calculi such as CCSK which have no redundancy in the history information. However, this approach right now does not scale to more complex languages such as the π-calculus or Erlang. Hence, it would be interesting to study alternative technical means to represent their reversible extensions with no redundancy (while current versions have high redundancy), so to allow for simple equational reasoning.

References

1. C. Aubert and I. Cristescu. Contextual equivalences in configuration structures and reversibility. *J. Log. Algebr. Meth. Program.*, 86(1):77–106, 2017.
2. C. Aubert and I. Cristescu. How reversibility can solve traditional questions: The example of hereditary history-preserving bisimulation. In *CONCUR*, volume 171 of *LIPIcs*, pages 7:1–7:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.
3. F. Barbanera, I. Lanese, and U. de'Liguoro. A theory of retractable and speculative contracts. *Sci. Comput. Program.*, 167:25–50, 2018.
4. M. Bednarczyk. Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical report, Institute of Computer Science, Polish Academy of Sciences, Gdańsk, 1991.
5. L. Cardelli and C. Laneve. Reversibility in massive concurrent systems. *Sci. Ann. Comput. Sci.*, 21(2):175–198, 2011.
6. C. D. Carothers, K. S. Perumalla, and R. Fujimoto. Efficient optimistic parallel simulations using reverse computation. *ACM Transactions on Modeling and Computer Simulation*, 9(3):224–253, 1999.
7. I. Castellani, M. Dezani-Ciancaglini, and P. Giannini. Reversible sessions with flexible choices. *Acta Informatica*, 56(7-8):553–583, 2019.
8. I. Cristescu, J. Krivine, and D. Varacca. A compositional semantics for the reversible π-calculus. In *LICS*, pages 388–397. IEEE Computer Society, 2013.
9. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.
10. R. De Nicola, U. Montanari, and F. W. Vaandrager. Back and forth bisimulations. In *CONCUR*, volume 458 of *LNCS*, pages 152–165. Springer, 1990.
11. J. Engblom. A review of reverse debugging. In *Proceedings of the 2012 System, Software, SoC and Silicon Debug Conference*, pages 1–6, 2012.

12. S. B. Fröschle and T. T. Hildebrandt. On plain and hereditary history-preserving bisimulation. In *MFCS*, volume 1672 of *LNCS*, pages 354–365. Springer, 1999.
13. E. Graversen, I. Phillips, and N. Yoshida. Event structure semantics of (controlled) reversible CCS. *J. Log. Algebr. Meth. Program.*, 121:100686, 2021.
14. K. Honda and N. Yoshida. On reduction-based process semantics. *Theor. Comput. Sci.*, 151(2):437–486, 1995.
15. N. Kobayashi, B. C. Pierce, and D. N. Turner. Linearity and the pi-calculus. *ACM Trans. Program. Lang. Syst.*, 21(5):914–947, 1999.
16. J. Krivine. A verification technique for reversible process algebra. In *RC*, volume 7581 of *LNCS*, pages 204–217. Springer, 2012.
17. L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.
18. R. Landauer. Irreversibility and heat generated in the computing process. *IBM Journal of Research and Development*, 5:183 –191, 1961.
19. I. Lanese, D. Medic, and C. A. Mezzina. Static versus dynamic reversibility in CCS. *Acta Informatica*, 2019.
20. I. Lanese, C. A. Mezzina, and J. Stefani. Reversibility in the higher-order π-calculus. *Theor. Comput. Sci.*, 625:25–84, 2016.
21. I. Lanese, N. Nishida, A. Palacios, and G. Vidal. CauDEr: A causal-consistent reversible debugger for Erlang (system description). In *FLOPS*, volume 10818 of *LNCS*, pages 247–263. Springer, 2018.
22. I. Lanese, N. Nishida, A. Palacios, and G. Vidal. A theory of reversibility for Erlang. *J. Log. Algebraic Methods Program.*, 100:71–97, 2018.
23. I. Lanese and I. Phillips. Forward-reverse observational equivalences in CCSK (TR). http://www.cs.unibo.it/~lanese/work/CCSKequivTR.pdf.
24. J. McNellis, J. Mola, and K. Sykes. Time travel debugging: Root causing bugs in commercial scale software. CppCon talk, https://www.youtube.com/watch?v=l1YJTg_A914, 2017.
25. H. C. Melgratti, C. A. Mezzina, and I. Ulidowski. Reversing place transition nets. *Log. Methods Comput. Sci.*, 16(4), 2020.
26. C. A. Mezzina and V. Koutavas. A safety and liveness theory for total reversibility. In *TASE*, pages 1–8. IEEE Computer Society, 2017.
27. C. A. Mezzina and J. A. Pérez. Causally consistent reversible choreographies: a monitors-as-memories approach. In *PPDP*, pages 127–138. ACM, 2017.
28. R. Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.
29. I. Phillips and I. Ulidowski. Reversibility and models for concurrency. In *SOS*, volume 192(1) of *ENTCS*, pages 93–108. Elsevier, 2007.
30. I. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2):70–96, 2007.
31. I. Phillips and I. Ulidowski. A hierarchy of reverse bisimulations on stable configuration structures. *Math. Struct. Comput. Sci.*, 22:333–372, 2012.
32. I. Phillips and I. Ulidowski. Event identifier logic. *Math. Struct. Comput. Sci.*, 24(2), 2014.
33. I. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *RC*, volume 7581 of *LNCS*, pages 218–232. Springer, 2012.
34. D. Sangiorgi and D. Walker. On barbed equivalences in pi-calculus. In *CONCUR*, volume 2154 of *LNCS*, pages 292–304. Springer, 2001.