Static versus Dynamic Reversibility in CCS

Ivan Lanese, Doriana Medić, Claudio Antares Mezzina

Noname manuscript No.
(will be inserted by the editor)

Static vs Dynamic Reversibility in CCS

Ivan Lanese · Doriana Medić · Claudio Antares Mezzina

the date of receipt and acceptance should be inserted later

Abstract The notion of reversible computing is attracting interest because of its applications in diverse fields, in particular the study of programming abstractions for fault tolerant systems. Most computational models are not naturally reversible since computation causes loss of information, and history information must be stored to enable reversibility. In the literature, two approaches to reverse the CCS process calculus exist, differing on how history information is kept. Reversible CCS (RCCS), proposed by Danos and Krivine, exploits dedicated stacks of memories attached to each thread. CCS with Keys (CCSK), proposed by Phillips and Ulidowski, makes CCS operators static so that computation does not cause information loss. In this paper we show that RCCS and CCSK are equivalent in terms of LTS isomorphism.

Keywords reversibility · encoding · RCCS · CCSK · isomorphism

1 Introduction

The interest in reversibility dates back to the 60's, with Landauer [29] observing that only irreversible computations need to consume energy, fostering applications of reversibility in scenarios of low-energy computing. Landauer's principle has only been shown empirically in 2012 [3]. Nowadays reversible computing is attracting

This work was partially supported by the COST Action IC1405 on "Reversible computation - extending horizons of computing" and French ANR project DCore ANR-18-CE25-0007.

Doriana Medić
IMT School for Advanced Studies Lucca, Italy / Focus Team, University of Bologna/Inria, France

Claudio Antares Mezzina
Dipartimento di Scienze Pure e Applicate, Università di Urbino, Italy

Ivan Lanese
Focus Team, University of Bologna/Inria, Italy

interest because of its applications in diverse fields: biological modeling [41,10], since many biochemical reactions are by nature reversible; program debugging [4, 20,21,34] and testing [43], allowing during debugging time to bring the program state back to a previous execution point in which certain conditions are met; and parallel discrete event simulation [39]. Of particular interest is the application of reversible computation notions to the study of programming abstractions for dependable systems. Several techniques used to build dependable systems such as transactions [24], system-recovery schemes [19] and checkpoint-rollback protocols [28], rely in one way or another on some forms of undo. The ability to undo any single action provides us with an ideal setting to study, revisit, or imagine alternatives to standard techniques for building dependable systems and to debug them. Indeed distributed reversible actions can be seen as defeasible partial agreements: the building blocks for different transactional models and recovery techniques. Examples on how reversibility can be used to model transactions exist in CCS [16] and higher-order π [30].

Reversibility in a concurrent setting has been first studied in the context of the CCS process calculus [37]. The first reversible variant of CCS, called RCCS, was introduced by Danos and Krivine [15]. In RCCS each process is monitored by a *memory*, that is a stack of past actions. When a process splits because of a parallel composition, its memory is duplicated. The two memories then evolve independently.

A general method for reversing any process calculus formalized using SOS rules in a subset of the path format [2] has been proposed by Phillips and Ulidowski in [42]. The main idea of their approach is to use communication keys to uniquely identify communications, and to make each operator of the calculus *static*, so that no information is lost during computation. CCSK is obtained by instantiating this method on CCS. In CCSK, history information is embedded in the process structure, hence it is never duplicated.

We dub *static* the approach to reversibility of CCSK, while we call *dynamic* the one used by RCCS, since the memories grow as far as the computation progresses.

Hence a natural question arises:

Are RCCS and CCSK equivalent?

The question is relevant, since the two approaches have evolved independently for more than 10 years, giving rise to many extensions and results, e.g., [40,41,26,27, 23] for CCSK and [16–18,25,13,14,1] for RCCS.

In this paper we give a positive answer to the question above, in a very strong sense. Indeed, we show that the labeled transition systems of CCSK and RCCS (up to a few structural transformations on processes) are isomorphic and, as a consequence, they are equated by any behavioral equivalence, such as bisimilarity or trace equivalence. The proof of isomorphism relies on two encodings, one from CCSK to RCCS and the other in the opposite direction. Interestingly, none of the encodings is compositional, but no compositional encoding can exist since reachable[1] RCCS processes are not closed under parallel composition.

The rest of the paper is organized as follows: Section 2 briefly recalls the syntax of CCS and introduces the syntax and the semantics for both RCCS and CCSK. In Section 3 and Section 4 we present, respectively, the encoding from CCSK to

[1] Reachable processes are processes which are well formed in RCCS.

$$\text{(CCS Processes)} \quad P, Q ::= \mathbf{0} \mid \sum_{i \in I} \alpha_i.P_i \mid (P \parallel Q) \mid P\backslash a$$
$$\text{(Actions)} \quad \alpha ::= a \mid \overline{a} \mid \tau$$

Fig. 1 CCS Syntax

RCCS and from RCCS to CCSK, and prove operational correspondence results. In Section 5 we prove the isomorphism between the labeled transition systems of RCCS and CCSK and discuss the issue of non compositionality of the encodings. Section 6 shows some cross-fertilization results, while Section 7 concludes the paper with a discussion on related and future work.

This paper is a revised and enhanced version of [35]. In particular, in [35] the equivalence result was in terms of back and forth bisimilarity, while now we use a much stronger notion, namely LTS isomorphism. Moreover, the encoding from RCCS to CCSK (Section 4) was only sketched in [35], while the discussion about non compositionality of the encodings (Section 5) as well as the cross-fertilization results (Section 6) are new. Finally, the whole presentation has been carefully refined.

2 CCS and its Reversible Extensions

In this section we briefly present the syntax of CCS [37], and then we describe its two reversible extensions, namely RCCS [15] and CCSK [42].

Let $\mathcal{A}$ be a set of actions ranged over by a, and $\overline{\mathcal{A}}$ the corresponding set of co-actions, that is $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$. We let μ, λ and their decorated versions to range over the set $\mathtt{Act} = \mathcal{A} \cup \overline{\mathcal{A}}$, while we let α, β and their decorated versions to range over the set $\mathtt{Act}_\tau = \mathtt{Act} \cup \{\tau\}$, where $\tau \notin \mathtt{Act}$ is the *silent* action.

The syntax of CCS is in Figure 1. Here, $\mathbf{0}$ represents the process that does nothing. A prefix (or action) can be an input a, an output $\overline{a}$ or the silent action τ. $\sum_{i \in I} \alpha_i.P_i$ represents a choice between actions α_i, where P_i is the continuation to be executed after α_i. We write $\alpha.P$ to represent unary choice, and $\alpha_j.P_j + Q$ where $Q = \sum_{i \in I \setminus \{j\}} \alpha_i.P_i$ to emphasize a specific alternative. We assume $\sum_{i \in I} \alpha_i.P_i = \mathbf{0}$ if $I = \emptyset$. $P \parallel Q$ represents the parallel composition of processes P and Q. An action a can be restricted so to be visible only inside process P, written $P\backslash a$. Restriction is the only binder in CCS: a is bound in $P\backslash a$. We write $\mathtt{n}(P)$ for the set of names of a process P, and, respectively, $\mathtt{fn}(P)$ and $\mathtt{bn}(P)$ for the sets of free and bound names. The set $\mathcal{P}$ denotes the set of all CCS processes.

2.1 Reversible CCS

In this section we present RCCS [15,25]. RCCS stores information on past actions inside memories attached to processes. Hence, a memory records every action that the process has performed in the past.

The syntax of RCCS is in Figure 2. RCCS processes are built on top of CCS processes. A term of the form $m \triangleright P$, where m is a memory and P is a CCS process,

$$\begin{array}{rl} \text{(CCS Processes)} & P, Q ::= \mathbf{0} \mid \sum_{i \in I} \alpha_i.P_i \mid (P \parallel Q) \mid P\backslash a \\ \text{(RCCS Processes)} & R, S ::= m \triangleright P \mid (R \parallel S) \mid R\backslash a \\ \text{(Memories)} & m ::= \langle\rangle \mid \langle k, \alpha, Q\rangle \cdot m \mid \langle\uparrow\rangle \cdot m \end{array}$$

Fig. 2 RCCS syntax

is called a *monitored* process. Two RCCS processes R and S can be composed in parallel via $R \parallel S$, and an action a of a RCCS process R can be restricted via $R\backslash a$. *Memories* are stacks of events, with the top of the memory (on the left in the textual representation) storing the very last action of the monitored process. The empty memory is represented by $\langle\rangle$, while $\langle k, \alpha, Q\rangle$ represents an action event meaning that the monitored process did the action α identified by *key* k and, in doing α, alternatives described by Q were discarded. Event $\langle\uparrow\rangle$ represents the split of a process in two parallel ones. We use e to range over events. The cons operator is written as $\cdot$, hence $e \cdot \langle\rangle$ is the stack containing a unique event e. Given two memories m_1 and m_2 we write $m_1@m_2$ for the memory obtained by pushing all the elements in m_1 on top of m_2 (preserving their order). The @ operator can be formally defined as follows.

Definition 1 We define $m_1@m_2$ by structural induction on m_1:

$$\begin{array}{c} \langle\rangle@m_2 = m_2 \\ (e \cdot m_1)@m_2 = e \cdot (m_1@m_2) \end{array}$$

We assume that the cons operator $\cdot$ has precedence over @. We denote by $\mathtt{Mem}$ the set of all memories.

Formally, we assume the existence of an infinite denumerable set of action identifiers, called *keys*, $\mathcal{K}$ such that $\mathcal{K} \cap \mathtt{Act} = \emptyset$. Let $\mathtt{ActK} = \mathtt{Act} \times \mathcal{K}$ be the set of pairs formed by an action μ and a key k. In the same way we define $\mathtt{ActK}_\tau = \mathtt{Act}_\tau \times \mathcal{K}$.

We define below a function computing the set of keys in a given memory or in a given process.

Definition 2 The set of keys in a memory m, or in a RCCS process R, written respectively $\mathtt{key}(m)$ and $\mathtt{key}(R)$, is inductively defined as follows:

$$\begin{array}{rlrl} \mathtt{key}(\langle\uparrow\rangle \cdot m) &= \mathtt{key}(m) & \mathtt{key}(\langle k, \alpha, Q\rangle \cdot m) &= \{k\} \cup \mathtt{key}(m) \\ \mathtt{key}(\langle\rangle) &= \emptyset & \mathtt{key}(m \triangleright P) &= \mathtt{key}(m) \\ \mathtt{key}(R\backslash a) &= \mathtt{key}(R) & \mathtt{key}(R \parallel S) &= \mathtt{key}(R) \cup \mathtt{key}(S) \end{array}$$

As for CCS, the only binder in RCCS is restriction (at the level of both CCS processes and RCCS processes). We extend functions $\mathtt{n}$, $\mathtt{fn}$ and $\mathtt{bn}$ to RCCS processes and memories accordingly.

We now introduce the concept of labeled transition system (LTS), and use it to define the semantics of RCCS. Later on we will use LTSs also to define the semantics of CCSK.

$$\text{(R-ACT)}\quad \frac{k \notin \mathtt{key}(m)}{m \triangleright \alpha.P + Q \xrightarrow{k,\alpha} \langle k, \alpha, Q\rangle \cdot m \triangleright P} \qquad \text{(R-ACT}^\bullet\text{)}\quad \frac{k \notin \mathtt{key}(m)}{\langle k, \alpha, Q\rangle \cdot m \triangleright P \overset{k,\alpha}{\rightsquigarrow} m \triangleright \alpha.P + Q}$$

$$\text{(R-PAR-L)}\quad \frac{R \xrightarrow{k,\alpha} R' \qquad k \notin \mathtt{key}(S)}{R \parallel S \xrightarrow{k,\alpha} R' \parallel S} \qquad \text{(R-PAR-L}^\bullet\text{)}\quad \frac{R \overset{k,\alpha}{\rightsquigarrow} R' \qquad k \notin \mathtt{key}(S)}{R \parallel S \overset{k,\alpha}{\rightsquigarrow} R' \parallel S}$$

$$\text{(R-PAR-R)}\quad \frac{S \xrightarrow{k,\alpha} S' \qquad k \notin \mathtt{key}(R)}{R \parallel S \xrightarrow{k,\alpha} R \parallel S'} \qquad \text{(R-PAR-R}^\bullet\text{)}\quad \frac{S \overset{k,\alpha}{\rightsquigarrow} S' \qquad k \notin \mathtt{key}(R)}{R \parallel S \overset{k,\alpha}{\rightsquigarrow} R \parallel S'}$$

$$\text{(R-SYN)}\quad \frac{R \xrightarrow{k,\alpha} R' \qquad S \xrightarrow{k,\bar{\alpha}} S'}{R \parallel S \xrightarrow{k,\tau} R' \parallel S'} \qquad \text{(R-SYN}^\bullet\text{)}\quad \frac{R \overset{k,\alpha}{\rightsquigarrow} R' \qquad S \overset{k,\bar{\alpha}}{\rightsquigarrow} S'}{R \parallel S \overset{k,\tau}{\rightsquigarrow} R' \parallel S'}$$

$$\text{(R-RES)}\quad \frac{R \xrightarrow{k,\alpha} R' \qquad \alpha \notin \{a, \bar{a}\}}{R\backslash a \xrightarrow{k,\alpha} R'\backslash a} \qquad \text{(R-RES}^\bullet\text{)}\quad \frac{R \overset{k,\alpha}{\rightsquigarrow} R' \qquad \alpha \notin \{a, \bar{a}\}}{R\backslash a \overset{k,\alpha}{\rightsquigarrow} R'\backslash a}$$

$$\text{(R-EQUIV)}\quad \frac{R \equiv R' \qquad R' \xrightarrow{k,\alpha} S' \qquad S' \equiv S}{R \xrightarrow{k,\alpha} S} \qquad \text{(R-EQUIV}^\bullet\text{)}\quad \frac{R \equiv R' \qquad R' \overset{k,\alpha}{\rightsquigarrow} S' \qquad S' \equiv S}{R \overset{k,\alpha}{\rightsquigarrow} S}$$

Fig. 3 RCCS semantics

Definition 3 (LTS) A *labeled transition system* (LTS) is a triple $\langle \mathcal{P}, \mathcal{L}, \rightarrow\rangle$ where $\mathcal{P}$ is a set of states, $\mathcal{L}$ a set of labels, and $\rightarrow \subseteq \mathcal{P} \times \mathcal{L} \times \mathcal{P}$ a transition relation. We write $P \xrightarrow{l} P'$ when $\langle P, l, P'\rangle \in \rightarrow$.

We can now define the operational semantics of RCCS.

Definition 4 (RCCS Semantics) The operational semantics of RCCS is defined as a pair of LTSs on the same set of states and set of labels: a forward LTS $(\mathcal{P}_R, \mathtt{Act}\mathtt{K}_\tau, \rightarrow)$ and a backward LTS $(\mathcal{P}_R, \mathtt{Act}\mathtt{K}_\tau, \rightsquigarrow)$, where $\mathcal{P}_R$ is the set of RCCS processes. We define $\rightleftharpoons = \rightarrow \cup \rightsquigarrow$. Transition relations $\rightarrow$ and $\rightsquigarrow$ are the smallest relations induced by the rules in Figure 3 (left and right columns, respectively). Both relations exploit the structural congruence relation $\equiv$, which is the smallest congruence on RCCS processes containing the rules in Figure 4.

Let us comment on the forward rules (Figure 3, left column). Rule R-ACT allows a monitored process to perform a forward action. The action is associated to a fresh key k. Moreover, a new action event is stored on top of the memory, containing the key k, the action α, and the part of the process which has been discarded by the action, that is Q. Rules R-PAR-L and R-PAR-R propagate an

$$
\begin{array}{lll}
(\textsc{Split}) & m \triangleright (P \parallel Q) \equiv \langle\uparrow\rangle \cdot m \triangleright P \parallel \langle\uparrow\rangle \cdot m \triangleright Q & \\
(\textsc{Res}) & m \triangleright P\backslash a \equiv (m \triangleright P)\backslash a & \text{if } a \notin \mathtt{fn}(m) \\
(\alpha) & R \equiv S & \text{if } R =_\alpha S
\end{array}
$$

Fig. 4 RCCS Structural laws

action through a parallel composition, under the condition that the key k of the action is not used by the parallel process. This check guarantees that all the keys are unique. Rule R-SYN allows two parallel processes to synchronize. To do so, they have to match both the action α and the key k. Rule R-RES propagates actions through restriction provided that the action is not on the restricted name. Rule R-EQUIV allows one to exploit structural congruence. Structural rule (Split) allows a monitored process with a toplevel parallel composition to split, duplicating the memory. Structural rule (Res) allows one to push restriction outside monitored processes. Structural rule (α) allows one to exploit α-conversion, denoted by $=_\alpha$.

Backward rules are in Figure 3, right column. For each forward rule there exists a symmetric backward one. Rule R-ACT$^\bullet$ allows a monitored process to revert its last action. To do so, the event on top of the memory is taken and the information contained in it is used to restore the process as it was before performing the action. Rules R-PAR-L$^\bullet$ and R-PAR-R$^\bullet$ propagate a backward action through a parallel composition, only if the identifier k of the action does not belong to monitored processes in parallel. This check is crucial to avoid partial undo of some synchronizations, as shown by the following example.

Example 1 (Partial undo counterexample) Consider the following RCCS transition:

$$R = m_1 \triangleright a.P \parallel m_2 \triangleright \overline{a}.Q \xrightarrow{k,\tau} \langle k, a, \mathbf{0}\rangle \cdot m_1 \triangleright P \parallel \langle k, \overline{a}, \mathbf{0}\rangle \cdot m_2 \triangleright Q = R_1$$

Only the side condition in rule R-PAR-L$^\bullet$ forbids the left parallel component in R_1 to undo its action with key k unless the right component also undoes its action with the same key. That is:

$$\langle k, a, \mathbf{0}\rangle \cdot m_1 \triangleright P \parallel \langle k, \overline{a}, \mathbf{0}\rangle \cdot m_2 \triangleright Q \overset{k,a}{\not\rightsquigarrow} m_1 \triangleright a.P \parallel \langle k, \overline{a}, \mathbf{0}\rangle \cdot m_2 \triangleright Q = R_2$$

Indeed, R_2 is not a legal past state of R_1, even if it is a legal future for R. In fact, R_2 is obtained if the right parallel component in R synchronizes with the environment, while R_1 is obtained when the two parallel components synchronize with each other.

The remaining rules are similar to the forward ones.

Remark 1 The first presentation of RCCS [15] used events of the form $\langle m_*, \alpha, Q\rangle$, $\langle 1\rangle$ and $\langle 2\rangle$, where m_* is either a memory m, if the process synchronized with another process monitored by m, or $*$ if the process synchronized with its environment. Events $\langle 1\rangle$ and $\langle 2\rangle$ were used to split a process along a parallel composition according to the following rule:

$$m \triangleright (P \parallel Q) \equiv \langle 1\rangle \cdot m \triangleright P \parallel \langle 2\rangle \cdot m \triangleright Q$$

The presentation of RCCS we use, appeared in [25], simplifies the handling of memories and makes the splitting through the parallel composition commutative. However, as remarked in [25], the two presentations are conceptually the same. We have chosen the one in [25] since it simplifies our technical development.

Neither [15] nor [25] exploit α-conversion. However, this is needed, otherwise one could write a deadlocked processes of the form:

$$\langle k, \overline{a}, \mathbf{0} \rangle \cdot \langle\rangle \triangleright (b \backslash a)$$

The process above cannot execute. According to RCCS semantics, the only possibility would be to take the restriction to the top level using structural rule (Res), but this is forbidden by the side condition. Clearly, α-conversion solves the problem, allowing:

$$\langle k, \overline{a}, \mathbf{0} \rangle \cdot \langle\rangle \triangleright (b \backslash a) \equiv \langle k, \overline{a}, \mathbf{0} \rangle \cdot \langle\rangle \triangleright (b \backslash c) \equiv (\langle k, \overline{a}, \mathbf{0} \rangle \cdot \langle\rangle \triangleright b) \backslash c \xrightarrow{w,b}$$

hence we add it. As far as we know, α-conversion first appears in RCCS literature in [1].

Not all RCCS processes are actually meaningful, in the following we will restrict our attention to reachable processes, as standard in the RCCS literature.

Definition 5 (Reachable Process) A RCCS process R is *initial* if it has the form $\langle\rangle \triangleright P$. A RCCS process R is *reachable* if it can be derived from an initial process by using the rules in Figure 3.

A main property of RCCS, and in general of reversible calculi, is the Loop Lemma [15, Lemma 6], which states that each action can be undone.

Lemma 1 (RCCS Loop Lemma) *For each pair of reachable RCCS processes R and R', $R \xrightarrow{k,\alpha} R'$ iff $R' \overset{k,\alpha}{\rightsquigarrow} R$.*

The proof follows from the symmetry between the forward and the backward rules.

2.2 CCS with Communication Keys

In this section we present a reversible CCS obtained by applying the general approach in [42]. Indeed, [42] presents a general approach to derive reversible extensions of calculi formalized using SOS rules in a subset of the path format [2]. It also presents an instance of the application of the approach to CCS. However, the CCS used in [42] is slightly different from the one used to define RCCS [15, 25], since it features no τ prefix, but it allows for unguarded choice and restriction on sets of names. Here, to have a direct comparison with RCCS, concentrating on the reversibility mechanisms and not on the underlying calculus, we apply the general approach in [42] to the CCS version underlying RCCS. We will call CCSK the resulting calculus since it is indeed a CCS with keys, and it is very close to the one in [42].

The main idea behind the approach in [42] is to make all the operators of CCS static. When subprocesses composed using a static operator evolve, both the static operator and the subprocesses that do not evolve are preserved. In

$$\text{(CCSK Processes)} \quad X, Y ::= \mathbf{0} \mid \sum_{i \in I} \pi_i.X_i \mid (X \parallel Y) \mid X \backslash a$$
$$\text{(CCSK Prefixes)} \quad \pi ::= \alpha \mid \alpha[k]$$

Fig. 5 CCSK syntax

CCS, for instance, parallel composition is static, while choice is not. In order to make operators like choice static, performed actions are annotated as having been performed instead of being discarded. When all operators are static, computation causes no loss of information, hence there is no need of using external memory.

The syntax of CCSK is in Figure 5. The only difference w.r.t. CCS processes is that now prefixes may be annotated by identifiers, called keys. Keys have the same role both in CCSK and in RCCS, hence we use the same set of keys in both settings. We now define a few notations for key management in CCSK. We start by defining the function $\mathtt{key}(\cdot)$ computing the set of keys in a given process also for CCSK processes.

Definition 6 The set of keys of a process X, written $\mathtt{key}(X)$, is inductively defined as follows:

$$\mathtt{key}(\alpha.P) = \mathtt{key}(\mathbf{0}) = \emptyset \qquad \mathtt{key}(\alpha[k].X) = \{k\} \cup \mathtt{key}(X)$$
$$\mathtt{key}(X \parallel Y) = \mathtt{key}(X) \cup \mathtt{key}(Y) \qquad \mathtt{key}(\sum_{i \in I} \pi_i.X_i) = \bigcup_{i \in I} \mathtt{key}(\pi_i.X_i)$$
$$\mathtt{key}(X \backslash a) = \mathtt{key}(X)$$

Definition 7 A key k is *fresh* in a process X, written $\mathtt{fresh}(k, X)$ if $k \notin \mathtt{key}(X)$.

Functions `n`, `fn` and `bn` extend from CCS to CCSK as expected.

Definition 8 (Standard Process) A CCSK process X is standard, written $\mathtt{std}(X)$, if $\mathtt{key}(X) = \emptyset$.

Standard CCSK processes coincide with CCS processes.

Notation In the following, we may denote standard CCSK processes using P instead of X, to highlight the fact that they do not contain keys, and are indeed CCS processes. In this case we drop the predicate $\mathtt{std}(X)$.

We can now define the operational semantics of CCSK processes.

Definition 9 (CCSK Semantics) The operational semantics of CCSK is defined as a pair of LTSs on the same set of states and set of labels: a forward LTS $(\mathcal{P}_K, \mathtt{ActK}_\tau, \rightarrow)$ and a backward LTS $(\mathcal{P}_K, \mathtt{ActK}_\tau, \rightsquigarrow)$, where $\mathcal{P}_K$ is the set of CCSK processes. We define $\rightleftharpoons = \rightarrow \cup \rightsquigarrow$. Transition relations $\rightarrow$ and $\rightsquigarrow$ are the smallest relations induced by the rules in Figure 6 (left and right columns, respectively) and closed under α-conversion $=_\alpha$.

Differently from RCCS, CCSK semantics does not exploit any structural congruence but α-conversion.

$$\text{(K-ACT1)}\quad \alpha.P \xrightarrow{\alpha[k]} \alpha[k].P \qquad \text{(K-ACT1}^\bullet\text{)}\quad \alpha[k].P \overset{\alpha[k]}{\rightsquigarrow} \alpha.P$$

$$\text{(K-ACT2)}\quad \frac{X \xrightarrow{\beta[h]} X' \qquad k \neq h}{\alpha[k].X \xrightarrow{\beta[h]} \alpha[k].X'} \qquad \text{(K-ACT2}^\bullet\text{)}\quad \frac{X \overset{\beta[h]}{\rightsquigarrow} X' \qquad k \neq h}{\alpha[k].X \overset{\beta[h]}{\rightsquigarrow} \alpha[k].X'}$$

$$\text{(K-SUM)}\quad \frac{X_j \xrightarrow{\alpha[k]} X'_j \qquad \forall i \neq j.X_i = X'_i \wedge \mathtt{std}(X_i)}{\sum_{i\in I} X_i \xrightarrow{\alpha[k]} \sum_{i\in I} X'_i} \qquad \text{(K-SUM}^\bullet\text{)}\quad \frac{X_j \overset{\alpha[k]}{\rightsquigarrow} X'_j \qquad \forall i \neq j.X_i = X'_i \wedge \mathtt{std}(X_i)}{\sum_{i\in I} X_i \overset{\alpha[k]}{\rightsquigarrow} \sum_{i\in I} X'_i}$$

$$\text{(K-PAR-L)}\quad \frac{X \xrightarrow{\alpha[k]} X' \qquad \mathtt{fresh}(k, Y)}{X \parallel Y \xrightarrow{\alpha[k]} X' \parallel Y} \qquad \text{(K-PAR-L}^\bullet\text{)}\quad \frac{X \overset{\alpha[k]}{\rightsquigarrow} X' \qquad \mathtt{fresh}(k, Y)}{X \parallel Y \overset{\alpha[k]}{\rightsquigarrow} X' \parallel Y}$$

$$\text{(K-PAR-R)}\quad \frac{Y \xrightarrow{\alpha[k]} Y' \qquad \mathtt{fresh}(k, X)}{X \parallel Y \xrightarrow{\alpha[k]} X \parallel Y'} \qquad \text{(K-PAR-R}^\bullet\text{)}\quad \frac{Y \overset{\alpha[k]}{\rightsquigarrow} Y' \qquad \mathtt{fresh}(k, X)}{X \parallel Y \overset{\alpha[k]}{\rightsquigarrow} X \parallel Y'}$$

$$\text{(K-SYN)}\quad \frac{X \xrightarrow{\alpha[k]} X' \qquad Y \xrightarrow{\bar{\alpha}[k]} Y' \qquad \alpha \neq \tau}{X \parallel Y \xrightarrow{\tau[k]} X' \parallel Y'} \qquad \text{(K-SYN}^\bullet\text{)}\quad \frac{X \overset{\alpha[k]}{\rightsquigarrow} X' \qquad Y \xrightarrow{\bar{\alpha}[k]} Y' \qquad \alpha \neq \tau}{X \parallel Y \overset{\tau[k]}{\rightsquigarrow} X' \parallel Y'}$$

$$\text{(K-RES)}\quad \frac{X \xrightarrow{\alpha[k]} X' \qquad \alpha \notin \{a, \bar{a}\}}{X\backslash a \xrightarrow{\alpha[k]} X'\backslash a} \qquad \text{(K-RES}^\bullet\text{)}\quad \frac{X \overset{\alpha[k]}{\rightsquigarrow} X' \qquad \alpha \notin \{a, \bar{a}\}}{X\backslash a \overset{\alpha[k]}{\rightsquigarrow} X'\backslash a}$$

Fig. 6 CCSK semantics

Remark 2 The original semantics of CCSK, and in general the rule format considered in [42], do not allow for α-conversion. However, α-conversion can be added following the guidelines given in [42] to deal with structural congruence. The addition is straightforward. Hence, we exploit α-conversion to have a direct match with RCCS α-conversion.

Rules for forward transitions are in Figure 6, left column. Rule K-ACT1 deals with prefixed processes $\alpha.P$ (note that $\alpha.P$ is a standard process). It just executes a prefix, putting it into the label. Differently from the normal CCS rule for prefix, it also generates a fresh new key k which is associated to the action α, which thus becomes $\alpha[k]$. Notably, the prefix is not discarded during the transition. Rule K-ACT2 allows a prefixed process $\alpha[k].X$ to execute if X can execute. The actions that X can do are forced to use keys different from k. Rule K-SUM deals with choice. Let us note that no subprocess is discarded when applying this rule. In more

detail, if one of the alternatives, say X_j, does an action, say $\alpha[k]$, and becomes X'_j, then the whole process performs the same action. The other alternatives, that is X_i with $i \neq j$, are unchanged. Furthermore, they need to be standard processes. Indeed, if they were not standard then they would have already executed, hence X_j would not be a valid alternative any more. Rules K-PAR-L and K-PAR-R propagate an action $\alpha[k]$ through a parallel composition, provided that the key k is not used by the other processes in parallel (use of $\texttt{fresh}(\cdot)$ predicate in the premises). Rule K-SYN allows two processes in parallel to synchronize. To do so, they have to match both the action and the key. Rule K-RES deals with restriction in the canonical CCS way. Backward rules (right column) are symmetric w.r.t. the forward ones.

Also in CCSK we will restrict the attention to reachable processes, as standard in the CCSK literature.

Definition 10 (Reachable Process) A CCSK process X is *initial* if it is standard, hence a CCS process. A CCSK process X is *reachable* if it can be derived from an initial process P by using the rules in Figure 6.

As for RCCS, also in CCSK the Loop Lemma [42, Proposition 5.1] holds.

Lemma 2 (CCSK Loop Lemma) *For each pair of reachable CCSK processes X and X', $X \xrightarrow{\alpha[k]} X'$ iff $X' \overset{\alpha[k]}{\rightsquigarrow} X$.*

The proof follows from the symmetry between forward and backward rules.

3 Encoding CCSK into RCCS

In this section we present the encoding of CCSK into RCCS and prove the operational correspondence result.

In order to simplify the presentation of the encoding we show a structural property of reachable CCSK processes.

Property 1 (Sum form) If X is a reachable CCSK process and $X = \sum_{i \in I} \pi_i.X_i$, then there exists at most one index $j \in I$ such that $\pi_j = \alpha_j[k]$. Furthermore, for each $i \neq j$, $\texttt{std}(X_i)$ holds, that is X_i is a CCS process.

Proof By induction on the length of the derivation that led an initial process to X, and by case analysis on the last applied rule. □

We can now present the encoding $[\![\cdot]\!]$ of CCSK into RCCS. We recall that P denotes a standard process, while X a general CCSK process. Let $\mathcal{P}_K$ and $\mathcal{P}_R$ be the sets of CCSK and RCCS processes, respectively.

The encoding function $[\![\cdot]\!] : \mathcal{P}_K \to \mathcal{P}_R$ is defined in terms of an auxiliary encoding function, $[\![\cdot]\!] : \mathcal{P}_K \times \texttt{Mem} \to \mathcal{P}_R$, which takes one more argument: a RCCS memory. Both functions are defined in Figure 7. We use the same notation for the two functions, since they can always be easily distinguished because of the different number of arguments.

Let us comment on the encoding. The main difference between CCSK and RCCS is on how they keep track of the history. In RCCS the history information related to each process is stored in the corresponding memory, while in CCSK the

$$
\begin{array}{rcl}
[\![X]\!] & = & [\![X, \langle\rangle]\!] \\
[\![P, m]\!] & = & m \triangleright P \\
[\![\alpha[k].X + \sum_{j\in J} \alpha_j.P_j, m]\!] & = & [\![X, \langle k, \alpha, \sum_{j\in J} \alpha_j.P_j\rangle \cdot m]\!] \\
[\![X \parallel Y, m]\!] & = & [\![X, \langle\uparrow\rangle \cdot m]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot m]\!] \\
[\![X\backslash a, m]\!] & = & [\![X\{^b/_a\}, m]\!]\backslash b \quad \text{if } b \notin \mathtt{fn}(m) \wedge (b = a \vee b \notin \mathtt{fn}(X))
\end{array}
$$

Fig. 7 Encoding of CCSK into RCCS

same information is spread along the structure of the whole process. Moreover, a CCSK process may correspond to the composition of several monitored processes, since, in CCSK, memories are not duplicated when a parallel composition is reached. So the encoding has to inductively drill the structure of a CCSK process X, in order to build the final memory of each process and to find the action event corresponding to each labeled action $\alpha[k]$ present inside X. This explains why the encoding takes a RCCS memory m as additional argument.

The encoding of a standard process P with some memory m is just the monitored process $m \triangleright P$. If the process is not standard, then the encoding is by structural induction. In the case of choice, since the process is not standard and because of Property 1, exactly one alternative corresponds to a non standard process. This alternative refers to the action α which has been performed, hence it is used to build an action event. The parallel and the restriction operators of CCSK instead are mapped to the corresponding operators of RCCS. Let us note that, in the case of parallel composition, the memory m is duplicated into two identical memories $\langle\uparrow\rangle \cdot m$. The rule for restriction needs to avoid to capture free occurrences of a inside m: name capture is avoided by renaming a into a fresh name b. If $a \notin \mathtt{fn}(m)$ then one can choose $b = a$. Note that this makes the encoding nondeterministic in the choice of bound names. This is not an issue since both the calculi feature α-conversion, hence from now on we will consider the encoding as deterministic. Also, to simplify the technicalities, we make the following assumption.

Assumption 1 We always choose $[\![X]\!]$ in such a way that $\mathtt{bn}([\![X]\!]) \cap \mathtt{fn}([\![X]\!]) = \emptyset$.

In order to understand how the encoding works let us consider the following example.

Example 2 Let $X = a + b + c[k].(d[h] \parallel P)$. The encoding of X can be computed as in Figure 8.

We now show an example highlighting the need for renaming in the rule for restriction.

Example 3 Let $X = a[k].(P\backslash a)$. The encoding of X can be computed as in Figure 9. Note that the renaming avoids the capture of the occurrence of name a in the memory, and it makes Assumption 1 satisfied.

Before stating the correctness of the encoding, we need some auxiliary results. First, we can see any RCCS process R as a context C^R composed by parallel and restriction operators, containing numbered holes filled by monitored processes. Hence, we will use the following notation.

$$\begin{aligned}
[\![X]\!] &= [\![X, \langle\rangle]\!] \\
&= [\![a + b + c[k].(d[h] \parallel P), \langle\rangle]\!] \\
&= [\![d[h] \parallel P, \langle k, c, a + b\rangle \cdot \langle\rangle]\!] \\
&= [\![d[h], \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle]\!] \parallel [\![P, \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle]\!] \\
&= [\![\mathbf{0}, \langle h, d, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle]\!] \parallel \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle \triangleright P \\
&= \langle h, d, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle \triangleright \mathbf{0} \parallel \langle\uparrow\rangle \cdot \langle k, c, a + b\rangle \cdot \langle\rangle \triangleright P
\end{aligned}$$

Fig. 8 Encoding of $X = a + b + c[k].(d[h] \parallel P)$

$$\begin{aligned}
[\![X]\!] &= [\![X, \langle\rangle]\!] \\
&= [\![a[k].(P\backslash a), \langle\rangle]\!] \\
&= [\![P\backslash a, \langle a, k, \mathbf{0}\rangle \cdot \langle\rangle]\!] \\
&= [\![P\{^b/_a\}, \langle a, k, \mathbf{0}\rangle \cdot \langle\rangle]\!]\backslash b \\
&= (\langle a, k, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P\{^b/_a\})\backslash b
\end{aligned}$$

Fig. 9 Encoding of $X = a[k].(P\backslash a)$

Notation We write a RCCS process R as $C^R[1 \mapsto m_1 \triangleright P_1, \ldots, n \mapsto m_n \triangleright P_n]$, or, more compactly, as $C^R_{i\in\{1,\ldots,n\}}[i \mapsto m_i \triangleright P_i]$. We may drop R when not relevant. If $R = C^R[1 \mapsto m_1 \triangleright P_1, \ldots, n \mapsto m_n \triangleright P_n]$ then we denote by $R@m$ the process $C^R[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]$.

We now give an example to illustrate the notation above.

Example 4 Let us consider the RCCS process

$$R = (m_1 \triangleright P_1 \parallel m_2 \triangleright P_2)\backslash a \parallel m_3 \triangleright P_3$$

By using the notation above, we can write it as:

$$R = C^R[1 \mapsto m_1 \triangleright P_1, 2 \mapsto m_2 \triangleright P_2, 3 \mapsto m_3 \triangleright P_3]$$

or as

$$R = C^R_{i\in\{1,2,3\}}[i \mapsto m_i \triangleright P_i]$$

where $C^R[\bullet_1, \bullet_2, \bullet_3] = (\bullet_1 \parallel \bullet_2)\backslash a \parallel \bullet_3$.

We can use the notation above to establish useful properties of the translation of CCSK processes.

Lemma 3 *Let X be a CCSK process. There exist $C^{[\![X,\langle\rangle]\!]}$, n, m_1, ..., m_n, P_1, ..., P_n such that, for each RCCS memory m, we have*

$$[\![X, m]\!] = C^{[\![X,\langle\rangle]\!]}[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]$$

Proof Intuitively, the thesis follows by noticing that m is only inserted into monitored processes, and has no impact on the other parts of the term. The proof is by induction on the derivation of $[\![X, m]\!]$:

– if $X = P$ then $[\![X, m]\!] = m \triangleright P$ as desired (with $C^{[\![X,\langle\rangle]\!]}[\bullet] = \bullet$, $n = 1$, $m_1 = \langle\rangle$ and $P_1 = P$);
– if $X = \alpha[k].X' + \sum_{j\in J} \alpha_j.Q_j$ then $[\![X, m]\!] = [\![X', \langle k, \alpha, \sum_{j\in J} \alpha_j.Q_j\rangle \cdot m]\!]$, and by inductive hypothesis

$$[\![X', \langle k, \alpha, \sum_{j\in J} \alpha_j.Q_j\rangle \cdot m]\!] =$$
$$C^{[\![X',\langle\rangle]\!]}[1 \mapsto m_1@\langle k, \alpha, \sum_{j\in J} \alpha_j.Q_j\rangle \cdot m \triangleright P_1, \ldots, n \mapsto m_n@\langle k, \alpha, \sum_{j\in J} \alpha_j.Q_j\rangle \cdot m \triangleright P_n]$$

as desired, by selecting $C^{[\![X,\langle\rangle]\!]} = C^{[\![X',\langle\rangle]\!]}$;
– if $X = X' \parallel X''$ then $[\![X, m]\!] = [\![X', \langle\uparrow\rangle \cdot m]\!] \parallel [\![X'', \langle\uparrow\rangle \cdot m]\!]$, and by inductive hypotheses:

$$[\![X', \langle\uparrow\rangle \cdot m]\!] = C'^{[\![X',\langle\rangle]\!]}[1 \mapsto m'_1@\langle\uparrow\rangle \cdot m \triangleright P_1, \ldots, n_1 \mapsto m'_{n_1}@\langle\uparrow\rangle \cdot m \triangleright P_{n_1}]$$
$$[\![X'', \langle\uparrow\rangle \cdot m]\!] = C''^{[\![X'',\langle\rangle]\!]}[1 \mapsto m''_1@\langle\uparrow\rangle \cdot m \triangleright P'_1, \ldots, n_2 \mapsto m''_{n_2}@\langle\uparrow\rangle \cdot m \triangleright P'_{n_2}]$$

The thesis follows since

$$[\![X, m]\!] = C^{[\![X,\langle\rangle]\!]}[1 \mapsto m'_1@\langle\uparrow\rangle \cdot m \triangleright P_1, \ldots, n_1 \mapsto m'_{n_1}@\langle\uparrow\rangle \cdot m \triangleright P_{n_1},$$
$$n_1 + 1 \mapsto m''_1@\langle\uparrow\rangle \cdot m \triangleright P'_1, \ldots, n_1 + n_2 \mapsto m''_{n_2}@\langle\uparrow\rangle \cdot m \triangleright P'_{n_2}]$$

where $C^{[\![X,\langle\rangle]\!]} = C'^{[\![X',\langle\rangle]\!]} \parallel C'''^{[\![X'',\langle\rangle]\!]}$ with $C'''^{[\![X'',\langle\rangle]\!]}$ equal to $C''^{[\![X'',\langle\rangle]\!]}$ but for having hole numbers increased by n_1.
– if $X = X'\backslash a$ then $[\![X'\backslash a, m]\!] = [\![X'\{^b/_a\}, m]\!]\backslash b$ with $b \notin \mathtt{fn}(m) \wedge (b = a \vee b \notin \mathtt{fn}(X))$.
By inductive hypothesis we have:

$$[\![X'\{^b/_a\}, m]\!] = C^{[\![X'\{^b/_a\},\langle\rangle]\!]}[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]$$

Hence:

$$[\![X'\{^b/_a\}, m]\!]\backslash b = C^{[\![X'\{^b/_a\},\langle\rangle]\!]}[1 \mapsto m_1@m \triangleright P_1, \ldots, n \mapsto m_n@m \triangleright P_n]\backslash b$$

as desired, by selecting $C^{[\![X,\langle\rangle]\!]} = C^{[\![X'\{^b/_a\},\langle\rangle]\!]}\backslash b$.

□

We now show a decomposition property for RCCS transitions, which allows us to isolate the impact of structural congruence inside transitions.

Definition 11 The relation $\twoheadrightarrow$ is the smallest relation induced by the rules in Figure 3, left column, except rule R-EQUIV.

Note that by definition $\twoheadrightarrow \subset \rightarrow$.

Lemma 4 *If there is a RCCS transition $R \xrightarrow{k,\alpha} S$ then there exist $R' \equiv R$ and $S' \equiv S$ such that $R' \xrightarrow{k,\alpha} S'$.*

Proof The proof is by induction on the derivation of the transition $R \xrightarrow{k,\alpha} S$, with a case analysis on the last applied rule:

- Rule R-ACT: the thesis holds trivially, since R-ACT is an axiom, by choosing $R' = R$, $S' = S$.
- Rule R-PAR-L: we have that $R = R_1 \parallel R_2$ and $S = S_1 \parallel R_2$, with premise $R_1 \xrightarrow{k,\alpha} S_1$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1$. By congruence we have $R \equiv R'_1 \parallel R_2$ and $S \equiv S'_1 \parallel R_2$, and, by applying rule R-PAR-L with premise $R'_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1$, we obtain $R'_1 \parallel R_2 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1 \parallel R_2$ as desired.
- Rule R-PAR-R: similar to the case above.
- Rule R-SYN: similar to the case above.
- Rule R-RES: we have that $R = R_1\backslash a$ and $S = S_1\backslash a$, with premise $R_1 \xrightarrow{k,\alpha} S_1$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1$. By congruence $R \equiv R'_1\backslash a$ and $S \equiv S'_1\backslash a$, and, by applying rule R-RES with premise $R'_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1$, we obtain $R'_1\backslash a \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1\backslash a$ as desired.
- Rule R-EQUIV: we have as premises $R \equiv R_1$, $R_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S_1$, and $S_1 \equiv S$. By inductive hypothesis there exist $R'_1 \equiv R_1$ and $S'_1 \equiv S_1$ such that $R'_1 \xrightarrow{k,\alpha}\!\!\triangleright\, S'_1$. We can conclude by noticing that $R \equiv R_1 \equiv R'_1$ and $S \equiv S_1 \equiv S'_1$, as desired.

□

Next lemma shows that only structural congruence changes the context.

Lemma 5 *For each forward RCCS transition derived without using rule* R-EQUIV*:*

$$C^R[1 \mapsto m_1 \triangleright P_1, \ldots, n \mapsto m_n \triangleright P_n] \xrightarrow{k,\alpha}\!\!\triangleright\, S$$

we have:

$$S = C^R[1 \mapsto m''_1@m_1 \triangleright P'_1, \ldots, n \mapsto m''_n@m_n \triangleright P'_n]$$

where $\langle\uparrow\rangle \notin m''_i$ *for each* $i \in \{1, \ldots, n\}$.

Proof By induction on the derivation of the transition, by noticing that RCCS derivation rules only add memories to their monitored processes (only structural congruence may change the context). □

Lemma 8 shows that memory has essentially no impact on forward transitions. We prove this result by first considering transitions derived without structural congruence, then we consider structural congruence in isolation (this requires a condition to rule out some name capture), and finally we combine the two results.

Lemma 6 *There is a forward transition:*

$$C^R[1 \mapsto m_1 \triangleright P_1, \ldots, n \mapsto m_n \triangleright P_n] \xrightarrow{k,\alpha}\!\!\triangleright\, C^R[1 \mapsto m'_1@m_1 \triangleright P'_1, \ldots, n \mapsto m'_n@m_n \triangleright P'_n]$$

iff there is a forward transition

$$C^R[1 \mapsto \langle\rangle \triangleright P_1, \ldots, n \mapsto \langle\rangle \triangleright P_n] \xrightarrow{k,\alpha}\!\!\triangleright\, C^R[1 \mapsto m'_1 \triangleright P'_1, \ldots, n \mapsto m'_n \triangleright P'_n]$$

Proof By rule inspection. □

We now show that memories have no impact on structural congruence.

Lemma 7 *If $R \equiv S$ then, for each memory m such that $(\mathtt{bn}(R) \cup \mathtt{bn}(S)) \cap \mathtt{fn}(m) = \emptyset$, $R@m \equiv S@m$.*

Proof By induction on the derivation of $R \equiv S$. The only interesting case is the base one, corresponding to the application of an axiom. We have a case analysis on the applied axiom.

- SPLIT: we have $R = m' \triangleright (P \parallel Q)$ and $S = \langle\uparrow\rangle \cdot m' \triangleright P \parallel \langle\uparrow\rangle \cdot m' \triangleright Q$. By adding the same memory m to the processes R and S we get $R@m = m'@m \triangleright (P \parallel Q)$ and $S@m = \langle\uparrow\rangle \cdot m'@m \triangleright P \parallel \langle\uparrow\rangle \cdot m'@m \triangleright Q$. By applying the SPLIT axiom to the process $R@m$ we get $m'@m \triangleright (P \parallel Q) \equiv \langle\uparrow\rangle \cdot m'@m \triangleright P \parallel \langle\uparrow\rangle \cdot m'@m \triangleright Q$ as desired.
- RES: we have $R = m' \triangleright (P \backslash a)$ and $S = (m' \triangleright P) \backslash a$ with $a \notin \mathtt{fn}(m')$. By adding the same memory m to the processes R and S we get $R@m = m'@m \triangleright (P \backslash a)$ and $S@m = (m'@m \triangleright P) \backslash a$. By applying the axiom RES to the process $R@m$ we get $m'@m \triangleright (P \backslash a) \equiv (m'@m \triangleright P) \backslash a$ as desired. Note that $a \notin \mathtt{fn}(m')$ from the side condition of the inductive hypothesis and $a \notin \mathtt{fn}(m)$ from the statement of the lemma.
- α: we have $R \equiv S$ since $R =_\alpha S$. By adding the same memory m to both R and S we still have $R@m =_\alpha S@m$ (note that since $(\mathtt{bn}(R) \cup \mathtt{bn}(S)) \cap \mathtt{fn}(m) = \emptyset$ then m is not changed by α-conversion), which implies $R@m \equiv S@m$, as desired.

□

We now combine the results above to show that memory has no impact on forward transitions.

Lemma 8 *If we have a RCCS forward transition $R \xrightarrow{k,\alpha} R'$ then, for each memory m such that $(\mathtt{bn}(R) \cup \mathtt{bn}(R')) \cap \mathtt{fn}(m) = \emptyset$, we have $R@m \xrightarrow{k,\alpha} R'@m$.*

Proof By applying Lemma 4 we have that there exist S and S' such that $R \equiv S \xrightarrow{k,\alpha} S' \equiv R'$. By α-conversion we can choose S and S' such that $(\mathtt{bn}(S) \cup \mathtt{bn}(S')) \cap \mathtt{fn}(m) = \emptyset$.

Since memories have no impact on structural congruence (Lemma 7) from $R \equiv S$ we can derive $R@m \equiv S@m$. Thanks to Lemma 5, $S \xrightarrow{k,\alpha} S'$ has the form needed by Lemma 6, which can thus by applied twice obtaining $S@m \xrightarrow{k,\alpha} S'@m$. Using again Lemma 7, we can derive $S'@m \equiv R'@m$. By applying rule R-EQUIV we obtain $R@m \xrightarrow{k,\alpha} R'@m$, as desired. □

We can now prove the operational correspondence for forward transitions.

Proposition 1 (Forward Correctness) *Let X be a reachable CCSK process and $R = [\![X, \langle\rangle]\!]$. For each CCSK transition $X \xrightarrow{\alpha[k]} X'$ there exists a corresponding RCCS transition $R \xrightarrow{k,\alpha} R'$ with $[\![X', \langle\rangle]\!] = R'$.*

Proof By induction on the derivation of $X \xrightarrow{\alpha[k]} X'$ and by case analysis on the last applied rule.

K-ACT1: We have $\alpha.P \xrightarrow{\alpha[k]} \alpha[k].P$ with $\alpha \in \{a, \bar{a}, \tau\}$. By applying the encoding

$$[\![\alpha.P, \langle\rangle]\!] = \langle\rangle \triangleright \alpha.P$$

Then, by using RCCS rule R-ACT we get $\langle\rangle \triangleright \alpha.P \xrightarrow{k,\alpha} \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P$, with $[\![\alpha[k].P, \langle\rangle]\!] = \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P$.

K-ACT2: We have $\alpha[k].X \xrightarrow{\beta[h]} \alpha[k].X'$ with premise $X \xrightarrow{\beta[h]} X'$. Let $R = [\![X, \langle\rangle]\!]$. By applying the inductive hypothesis we have that $R \xrightarrow{h,\beta} R'$, with $R' = [\![X', \langle\rangle]\!]$. Since $[\![\alpha[k].X, \langle\rangle]\!] = [\![X, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!]$ thanks to Lemma 3 we have that $[\![\alpha[k].X, \langle\rangle]\!] = R@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$. Let a be the name in α. By Assumption 1 $a \notin \mathtt{bn}(R) \cup \mathtt{bn}(R')$. By applying Lemma 8 we have that $R@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \xrightarrow{h,\beta} R'@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$. The thesis follows since $[\![\alpha[k].X, \langle\rangle]\!] = R@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$ and, thanks to Lemma 3, $[\![\alpha[k].X', \langle\rangle]\!] = R'@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$.

K-SUM: We have $\sum_{i\in I} X_i \xrightarrow{\alpha[k]} \sum_{i\in I} X'_i$ with premise $X_j \xrightarrow{\alpha[k]} X'_j$.
We have two cases depending on whether X_j is a CCS process or not. If it is a CCS process then the proof is analogue to the case K-ACT1. If not, it is analogue to the case K-ACT2, with the only difference that the additional memory element has the choice of discarded processes in the last field instead of $\mathbf{0}$.

K-PAR-L: We have $X \parallel Y \xrightarrow{\alpha[k]} X' \parallel Y$ with premise $X \xrightarrow{\alpha[k]} X'$. Thanks to the definition of the encoding we have that $[\![X \parallel Y, \langle\rangle]\!] = [\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!]$. Thanks to Lemma 3 we have that $[\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] = [\![X, \langle\rangle]\!]@\langle\uparrow\rangle \cdot \langle\rangle$. By inductive hypothesis we have that $[\![X, \langle\rangle]\!] = R \xrightarrow{k,\alpha} R' = [\![X', \langle\rangle]\!]$. Thanks to Lemma 8 (the condition is trivially satisfied since $\mathtt{fn}(\langle\uparrow\rangle \cdot \langle\rangle) = \emptyset$) we have that $R@\langle\uparrow\rangle \cdot \langle\rangle \xrightarrow{k,\alpha} R'@\langle\uparrow\rangle \cdot \langle\rangle$. Thanks to Lemma 3 we have $R'@\langle\uparrow\rangle \cdot \langle\rangle = [\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!]$ and $[\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] = [\![Y, \langle\rangle]\!]@\langle\uparrow\rangle \cdot \langle\rangle$. By applying RCCS rule R-PAR-L we have that

$$[\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] \xrightarrow{k,\alpha} R'@\langle\uparrow\rangle \cdot \langle\rangle \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!]$$

We can conclude by noticing that

$$R'@\langle\uparrow\rangle \cdot \langle\rangle \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] = [\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] = [\![X' \parallel Y, \langle\rangle]\!]$$

K-PAR-R: This case is symmetric to the previous one.

K-SYN: We have $X \parallel Y \xrightarrow{\tau[k]} X' \parallel Y'$ with premises $X \xrightarrow{\alpha[k]} X'$ and $Y \xrightarrow{\overline{\alpha}[k]} Y'$. Thanks to the definition of the encoding we have that $[\![X \parallel Y, \langle\rangle]\!] = [\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!]$. By inductive hypothesis we have that $[\![X, \langle\rangle]\!] \xrightarrow{k,\alpha} R' = [\![X', \langle\rangle]\!]$ and $[\![Y, \langle\rangle]\!] \xrightarrow{k,\overline{\alpha}} S' = [\![Y', \langle\rangle]\!]$. Thanks to Lemma 3 and Lemma 8 (the condition is trivially satisfied since $\mathtt{fn}(\langle\uparrow\rangle \cdot \langle\rangle) = \emptyset$) we have that:

$$[\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] \xrightarrow{k,\alpha} R'@\langle\uparrow\rangle \cdot \langle\rangle = [\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!]$$

Similarly, we have that:

$$[\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] \xrightarrow{k,\overline{\alpha}} S'@\langle\uparrow\rangle \cdot \langle\rangle = [\![Y', \langle\uparrow\rangle \cdot \langle\rangle]\!]$$

By applying RCCS rule R-SYN we have that

$$[\![X, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot \langle\rangle]\!] \xrightarrow{k,\tau} [\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y', \langle\uparrow\rangle \cdot \langle\rangle]\!]$$

We can conclude by noticing that

$$[\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![Y', \langle\uparrow\rangle \cdot \langle\rangle]\!] = [\![X' \parallel Y', \langle\rangle]\!]$$

K-RES: We have $X\backslash a \xrightarrow{\alpha[k]} X'\backslash a$ with premise $X \xrightarrow{\alpha[k]} X'$. From the definition of the encoding we have that $[\![X\backslash a, \langle\rangle]\!] = [\![X\{^b/_a\}, \langle\rangle]\!]\backslash b$ with $b = a \vee b \notin \mathtt{fn}(X)$. From inductive hypothesis we have that $[\![X, \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X', \langle\rangle]\!]$. Since $\alpha \notin \{a, \bar{a}\}$ we have $[\![X, \langle\rangle]\!]\{^b/_a\} \xrightarrow{k,\alpha} [\![X'\{^b/_a\}, \langle\rangle]\!]$. By applying RCCS rule R-RES we get $[\![X\{^b/_a\}, \langle\rangle]\!]\backslash b \xrightarrow{k,\alpha} [\![X'\{^b/_a\}, \langle\rangle]\!]\backslash b = [\![X'\backslash a, \langle\rangle]\!]$ as desired.

α-conversion: We have $X \xrightarrow{\alpha[k]} X'$ with premise $X'' \xrightarrow{\alpha[k]} X'''$, $X =_\alpha X''$ and $X' =_\alpha X'''$. From inductive hypothesis we have that $[\![X, \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X', \langle\rangle]\!]$. By applying RCCS α-conversion we get $[\![X'', \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X''', \langle\rangle]\!]$ as desired. □

Proposition 2 (Backward Correctness) *Let X be a reachable CCSK process and $R = [\![X, \langle\rangle]\!]$. For each CCSK transition $X \overset{\alpha[k]}{\rightsquigarrow} X'$ there exists a corresponding RCCS transition $R \overset{k,\alpha}{\rightsquigarrow} R'$ with $[\![X', \langle\rangle]\!] = R'$.*

Proof From CCSK Loop Lemma (Lemma 2) we have that $X \overset{\alpha[k]}{\rightsquigarrow} X'$ implies $X' \xrightarrow{\alpha[k]} X$. By Proposition 1 we have that there exists a corresponding RCCS transition $R' \xrightarrow{k,\alpha} R$ with $[\![X, \langle\rangle]\!] = R$ and $[\![X', \langle\rangle]\!] = R'$. By applying RCCS Loop Lemma (Lemma 1) we have that $R \overset{k,\alpha}{\rightsquigarrow} R'$, as desired. □

The two propositions above prove that if we have a reachable CCSK process X, and if X does an action α in CCSK, then its encoding $[\![X, \langle\rangle]\!]$ does the same action in RCCS. The resulting process $[\![X', \langle\rangle]\!]$ is the encoding of process X'. Now we show the opposite direction.

Proposition 3 (Forward Completeness) *Let X be a reachable CCSK process and $R = [\![X, \langle\rangle]\!]$. For each RCCS transition $R \xrightarrow{k,\alpha} R'$ there exists a corresponding CCSK transition $X \xrightarrow{\alpha[k]} X'$ with $R' \equiv [\![X', \langle\rangle]\!]$.*

Proof Thanks to Lemma 4 we can equivalently write the statement as follows: for each reachable CCSK process X and RCCS processes R and R'', such that $R = [\![X, \langle\rangle]\!]$ and $R'' \equiv R$, if there exists a RCCS transition $R'' \xrightarrow{k,\alpha} R'$, then there exists a corresponding CCSK transition $X \xrightarrow{\alpha[k]} X'$, with $R' \equiv [\![X', \langle\rangle]\!]$. When considering $R'' \equiv R$ we will not consider α-conversion, since it can be trivially matched by CCSK α-conversion.

Now the proof is by structural induction on X with a case analysis on the last applied rule in the derivation of $R'' \xrightarrow{k,\alpha} R'$. We have two main cases, depending on whether X is a standard process P or not.

In the first case $X = P$, and we perform a structural induction on P.

$P = \alpha.P_1$: we have that $R = [\![\alpha.P_1, \langle\rangle]\!]$ and by applying the encoding

$$[\![\alpha.P_1, \langle\rangle]\!] = \langle\rangle \triangleright \alpha.P_1$$

Since we cannot apply any structural rule (beyond α-conversion), then $R'' = R$. Then, the only applicable rule is R-ACT, and we get

$$\langle\rangle \triangleright \alpha.P_1 \xrightarrow{k,\alpha} \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P_1 = R'$$

In CCSK, process $\alpha.P_1$ can do the same action α by applying the rule K-ACT1 and we get

$$\alpha.P_1 \xrightarrow{\alpha[k]} \alpha[k].P_1$$

Since $[\![\alpha[k].P_1, \langle\rangle]\!] = R'$ we are done.

$P = \sum_{l\in I} \alpha_l.P_l$: we have $R = [\![\sum_{l\in I} \alpha_l.P_l, \langle\rangle]\!]$. By applying the encoding we have:

$$[\![\sum_{l\in I} \alpha_l.P_l, \langle\rangle]\!] = \langle\rangle \triangleright \sum_{l\in I} \alpha_l.P_l$$

Since no structural congruence (beyond α-conversion) can be applied $R'' = R$. The only applicable rule is R-ACT and we have

$$\langle\rangle \triangleright \sum_{l\in I} \alpha_l.P_l \xrightarrow{k,\alpha_j} \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} \alpha_l.P_l\rangle \cdot \langle\rangle \triangleright P_j$$

By using CCSK rule K-ACT1 (since P is a CCS process) followed by K-SUM we can derive

$$\sum_{l\in I} \alpha_l.P_l \xrightarrow{\alpha_j[k]} \alpha_j[k].P_j + \sum_{l\in I\setminus\{j\}} \alpha_l.P_l$$

We can conclude by noticing that

$$[\![\alpha_j[k].P_j + \sum_{l\in I\setminus\{j\}} \alpha_l.P_l, \langle\rangle]\!] = [\![P_j, \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} \alpha_l.P_l\rangle \cdot \langle\rangle]\!] = \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} \alpha_l.P_l\rangle \cdot \langle\rangle \triangleright P_j$$

$P = P_1 \parallel P_2$: we have that $R = [\![P_1 \parallel P_2, \langle\rangle]\!]$ and by applying the encoding

$$[\![P_1 \parallel P_2, \langle\rangle]\!] = \langle\rangle \triangleright P_1 \parallel P_2$$

We have now two possibilities: either $R'' = R$ or $R'' = \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_1 \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_2$. In the first case no rule can be applied and we are done. Let us consider the second one. Now we distinguish three cases depending on whether the last applied rule is R-PAR-L, R-PAR-R or R-SYN.
If R-PAR-L is applied, then we have that

$$\langle\uparrow\rangle \cdot \langle\rangle \triangleright P_1 \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_2 \xrightarrow{k,\alpha} R_1 \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_2$$

with premise $\langle\uparrow\rangle \cdot \langle\rangle \triangleright P_1 \xrightarrow{k,\alpha} R_1$. Thanks to Lemma 6 we can derive $\langle\rangle \triangleright P_1 \xrightarrow{k,\alpha} R_1'$ with $R_1 = R_1' @ \langle\uparrow\rangle \cdot \langle\rangle$. Since $\langle\rangle \triangleright P_1 = [\![P_1, \langle\rangle]\!]$ we can apply the inductive hypothesis and get $P_1 \xrightarrow{\alpha[k]} X_1'$ with $[\![X_1', \langle\rangle]\!] = R_1'$. We can now apply CCSK rule K-PAR-L and derive

$$P_1 \parallel P_2 \xrightarrow{\alpha[k]} X_1' \parallel P_2$$

We can notice that $[\![X_1' \parallel P_2, \langle\rangle]\!] = [\![X_1', \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![P_2, \langle\uparrow\rangle \cdot \langle\rangle]\!]$. By applying Lemma 3 on $[\![X_1', \langle\rangle]\!] = R_1'$ we can derive $[\![X_1', \langle\uparrow\rangle \cdot \langle\rangle]\!] = R_1$. We can now conclude since

$$[\![X', \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![P_2, \langle\uparrow\rangle \cdot \langle\rangle]\!] = R_1 \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright P_2$$

as desired.
If R-PAR-R is applied we can reason as in the previous case. If R-SYN is applied, we use twice the inductive hypothesis and we reason as in the previous cases.

$P = P_1\backslash a$: we have that $R = [\![P_1\backslash a, \langle\rangle]\!]$ and by applying the encoding

$$[\![P_1\backslash a, \langle\rangle]\!] = \langle\rangle \triangleright (P_1\backslash a)$$

We have two possibilities: either $R'' = R$ or $R'' = (\langle\rangle \triangleright P_1)\backslash a$. In the first case no rule can be applied and we are done. Let us consider the second one. The only applicable rule is R-RES. We get

$$(\langle\rangle \triangleright P_1)\backslash a \xrightarrow{k,\alpha}\!\!\triangleright\ R'\backslash a$$

with premise $\langle\rangle \triangleright P_1 \xrightarrow{k,\alpha}\!\!\triangleright\ R'$. Since $\langle\rangle \triangleright P_1 = [\![P_1, \langle\rangle]\!]$ we can apply the inductive hypothesis and get that $P_1 \xrightarrow{\alpha[k]} X'$ with $[\![X', \langle\rangle]\!] = R'$. The thesis follows by applying the CCSK rule K-RES.

In the second case process X is not standard. We proceed by structural induction on X.

$X = \alpha[k].Y$: we have that $R = [\![\alpha[k].Y, \langle\rangle]\!]$, and by applying the encoding

$$[\![\alpha[k].Y, \langle\rangle]\!] = [\![Y, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!]$$

Take any RCCS transition

$$[\![Y, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!] \equiv R'' \xrightarrow{h,\beta}\!\!\triangleright\ R'$$

Let a be the name in α. By Assumption 1 $a \notin \mathtt{bn}([\![Y, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!])$. Also, working up to α-conversion we can assume that $a \notin \mathtt{bn}(R'') \cup \mathtt{bn}(R')$. Since memory has no impact on the structural congruence (Lemma 7) there exists $S'' \equiv [\![Y, \langle\rangle]\!]$ such that $R'' = S''@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$. By Lemma 6 we have that $S'' \xrightarrow{h,\beta}\!\!\triangleright\ S'$ with $R' = S'@\langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle$. By applying the inductive hypothesis on $[\![Y, \langle\rangle]\!] \equiv S'' \xrightarrow{h,\beta} S'$ we have that $Y \xrightarrow{\beta[h]} Y'$ with $[\![Y', \langle\rangle]\!] \equiv S'$. By applying CCSK rule K-ACT2 we can also derive $\alpha[k].Y \xrightarrow{\beta[h]} \alpha[k].Y'$. The thesis follows since $[\![\alpha[k].Y', \langle\rangle]\!] = [\![Y', \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!] \equiv R'$ thanks to Lemma 3.

$X = \alpha_j[k].X_j + \sum_{l\in I\setminus\{j\}} X_l$: we have that $R = [\![\alpha_j[k].X_j + \sum_{l\in I\setminus\{j\}} X_l, \langle\rangle]\!]$, and by applying the encoding

$$[\![\alpha_j[k].X_j + \sum_{l\in I\setminus\{j\}} X_l, \langle\rangle]\!] = [\![X_j, \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} X_l\rangle \cdot \langle\rangle]\!]$$

Take any RCCS transition

$$[\![X_j, \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} X_l\rangle \cdot \langle\rangle]\!] \equiv R'' \xrightarrow{h,\beta}\!\!\triangleright\ R'$$

Let $R''' = [\![X_j, \langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} X_l\rangle \cdot \langle\rangle]\!]$. From Assumption 1 we have that $\mathtt{fn}(\langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} X_l\rangle \cdot \langle\rangle) \cap \mathtt{bn}(R''') = \emptyset$. Also, working up to α-conversion

we can assume that $\mathtt{fn}(\langle k, \alpha_j, \sum_{l\in I\setminus\{j\}} X_l\rangle\cdot\langle\rangle)\cap(\mathtt{bn}(R'')\cup\mathtt{bn}(R'))=\emptyset$. Since memory has no impact on structural congruence (Lemma 7) there exists $S''\equiv[\![X_j,\langle\rangle]\!]$ such that $R''=S''@\langle k,\alpha_j,\sum_{l\in I\setminus\{j\}}X_l\rangle\cdot\langle\rangle$. By Lemma 6 we have that $S''\xrightarrow{h,\beta}S'$ with $R'=S'@\langle k,\alpha_j,\sum_{l\in I\setminus\{j\}}X_l\rangle\cdot\langle\rangle$. By applying the inductive hypothesis on $[\![X_j,\langle\rangle]\!]\equiv S''\xrightarrow{h,\beta}S'$ we have that $X_j\xrightarrow{\beta[h]}X_j'$ with $[\![X_j',\langle\rangle]\!]\equiv S'$.
By applying CCSK rule K-ACT2 and K-SUM we can also derive

$$\alpha_j[k].X_j+\sum_{l\in L\setminus\{j\}}X_l\xrightarrow{\beta[h]}\alpha_j[k].X_j'+\sum_{l\in L\setminus\{j\}}X_l$$

The thesis follows since

$$[\![\alpha_j[k].X_j'+\sum_{l\in L\setminus\{j\}}X_l,\langle\rangle]\!]=[\![X_j',\langle k,\alpha_j,\sum_{l\in L\setminus\{j\}}X_l\rangle\cdot\langle\rangle]\!]\equiv R'$$

thanks to Lemma 3.

$X=Y_1\parallel Y_2$: we have that $R=[\![Y_1\parallel Y_2,\langle\rangle]\!]$, and by applying the encoding we obtain

$$[\![Y_1\parallel Y_2,\langle\rangle]\!]=[\![Y_1,\langle\uparrow\rangle\cdot\langle\rangle]\!]\parallel[\![Y_2,\langle\uparrow\rangle\cdot\langle\rangle]\!]$$

Take any term $R''\equiv[\![Y_1,\langle\uparrow\rangle\cdot\langle\rangle]\!]\parallel[\![Y_2,\langle\uparrow\rangle\cdot\langle\rangle]\!]$. There are two cases: either $R''=R_1''\parallel R_2''$ with $R_1''\equiv[\![Y_1,\langle\uparrow\rangle\cdot\langle\rangle]\!]$ and $R_2''\equiv[\![Y_2,\langle\uparrow\rangle\cdot\langle\rangle]\!]$, or the two parallel sub-processes have been merged by applying the (Split) rule from right to left. In this last case no transition can be performed. Let us consider the first case. We have a case analysis depending on whether the last applied rule is R-PAR-L, R-PAR-R or R-SYN. If rule R-PAR-L is applied we have that $R_1''\parallel R_2''\xrightarrow{k,\alpha}S_1''\parallel R_2''$ with hypothesis $R_1''\xrightarrow{k,\alpha}S_1''$. Moreover, from Lemma 6 (condition verified since $\mathtt{fn}(\langle\uparrow\rangle\cdot\langle\rangle)=\emptyset$) there exist R_1''' such that $R_1''=R_1'''@\langle\uparrow\rangle\cdot\langle\rangle$ and $R_1'''\xrightarrow{k,\alpha}S_1'''$, where $S_1''=S_1'''@\langle\uparrow\rangle\cdot\langle\rangle$.
Since $R_1'''\equiv[\![Y_1,\langle\rangle]\!]$ we can apply the inductive hypothesis and obtain that

$$Y_1\xrightarrow{\alpha[k]}Y_1'\text{ with }[\![Y_1',\langle\rangle]\!]\equiv S_1'''$$

We can now apply CCSK rule K-PAR-L and derive the transition

$$Y_1\parallel Y_2\xrightarrow{\alpha[k]}Y_1'\parallel Y_2$$

and we conclude by noticing that

$$[\![Y_1'\parallel Y_2,\langle\rangle]\!]=[\![Y_1',\langle\uparrow\rangle\cdot\langle\rangle]\!]\parallel[\![Y_2,\langle\uparrow\rangle\cdot\langle\rangle]\!]\equiv S_1''\parallel R_2''$$

The other two cases are similar.

$X=Y\backslash a$: we have that $R=[\![Y\backslash a,\langle\rangle]\!]$ and by applying the encoding $[\![Y\backslash a,\langle\rangle]\!]=[\![Y\{^b/_a\},\langle\rangle]\!]\backslash b$ with $b=a\vee b\notin\mathtt{fn}(Y)$. Take any term $R''\equiv[\![Y\{^b/_a\},\langle\rangle]\!]\backslash b$. There are two cases: either $R''=R_1''\backslash b$ with $R_1''\equiv[\![Y\{^b/_a\},\langle\rangle]\!]$, or the restriction has been put back inside the term using structural rule Res. In this last

case no transition can be performed. Let us consider the first case. The only applicable rule is R-RES. We have $R_1''\backslash b \xrightarrow{k,\alpha} R'''\backslash b$ with hypothesis $R_1'' \xrightarrow{k,\alpha} R'''$. By applying the inductive hypothesis we obtain that

$$Y\{^b/_a\} \xrightarrow{\alpha[k]} Y'\{^b/_a\}$$

with $[\![Y'\{^b/_a\}, \langle\rangle]\!] \equiv R'''$. By applying CCSK rule K-RES we also have

$$Y\{^b/_a\}\backslash b \xrightarrow{\alpha[k]} Y'\{^b/_a\}\backslash b$$

The thesis follows since $[\![Y'\backslash a, \langle\rangle]\!] = [\![Y'\{^b/_a\}, \langle\rangle]\!]\backslash b \equiv R'''\backslash b$. □

Proposition 4 (Backward Completeness) *Let X be a reachable CCSK process and $R = [\![X, \langle\rangle]\!]$. For each RCCS transition $R \overset{k,\alpha}{\rightsquigarrow} R'$ there exists a corresponding CCSK transition $X \overset{\alpha[k]}{\rightsquigarrow} X'$ with $R' \equiv [\![X', \langle\rangle]\!]$.*

Proof From RCCS Loop Lemma (Lemma 1) we have that $R' \xrightarrow{k,\alpha} R$ in RCCS. From Proposition 3 we have that $X' \xrightarrow{\alpha[k]} X$ with $[\![X, \langle\rangle]\!] \equiv R$ and $[\![X', \langle\rangle]\!] = R'$. By applying RCCS rule R-EQUIV to $R' \xrightarrow{k,\alpha} R$ and $R \equiv [\![X, \langle\rangle]\!]$ we have that $R' \xrightarrow{k,\alpha} [\![X, \langle\rangle]\!]$. By applying CCSK Loop Lemma (Lemma 2) we have that $X \overset{\alpha[k]}{\rightsquigarrow} X'$. □

4 Encoding RCCS into CCSK

In this section we present an encoding of RCCS into CCSK and prove the operational correspondence result.

We need to encode together processes which share a prefix of the memory, while we can consider in isolation processes that do not share any prefix of the memory. We encode memories of RCCS processes starting from the oldest action (right in the textual representation).

First we define the trimming function δ. The purpose of function δ is to get rid of some split elements, starting from the right side of the memory. For example the result of applying δ to $\langle\uparrow\rangle \cdot \langle k, \alpha, Q\rangle \cdot \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle$ is $\langle\uparrow\rangle \cdot \langle k, \alpha, Q\rangle \cdot \langle\rangle$.

Definition 12 The function $\delta : \texttt{Mem} \to \texttt{Mem}$, is inductively defined as follows:

$$\delta(m@\langle\uparrow\rangle \cdot \langle\rangle) = \delta(m) \qquad \delta(\langle\rangle) = \langle\rangle$$
$$\delta(m@\langle k, \alpha, Q\rangle \cdot \langle\rangle) = m@\langle k, \alpha, Q\rangle \cdot \langle\rangle$$

We now define the notion of history context, which is the CCSK context corresponding to an RCCS memory without $\langle\uparrow\rangle$ elements.

Definition 13 (History Context) Given a memory m such that $\langle\uparrow\rangle \notin m$, the corresponding *history context* $\texttt{H}_m$ is defined as follows:

$$\begin{aligned} \texttt{H}_{\langle\rangle} &= \bullet \\ \texttt{H}_{\langle k,\alpha,\texttt{0}\rangle \cdot m} &= \texttt{H}_m[\alpha[k].\bullet] \\ \texttt{H}_{\langle k,\alpha,Q\rangle \cdot m} &= \texttt{H}_m[\alpha[k].\ \bullet + Q] \text{ if } Q \neq \mathbf{0} \end{aligned}$$

Let $\mathcal{P}_K$ and $\mathcal{P}_R$ be the sets of, respectively, CCSK and RCCS processes. The encoding function $(\!|\cdot|\!) : \mathcal{P}_R \to \mathcal{P}_K$ is inductively defined as follows:

$$(\!|R\backslash a|\!) = (\!|R|\!)\backslash a$$

$$(\!|C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle \cdot m) \triangleright P_l]|\!) = \mathtt{H}_m[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]] \quad \text{if } \langle\uparrow\rangle \notin m \text{ and the top operator in } C \text{ is } \|$$

$$(\!|m \triangleright P|\!) = \mathtt{H}_m[P]$$

We note that in the second rule C is used both as an RCCS context and as a CCSK context. This little abuse of notation is justified by the fact that context C only features parallel composition and restriction operators, which are available in both calculi. Thus, in the following, we will use the same notation for both kinds of contexts.

Let us comment on the rules. The function $(\!|\cdot|\!)$ is in charge of collecting common parts of the memories. Memories are analyzed starting from their oldest element, and the common part m is taken. The rest of the memory elements, m_l, are kept for further encoding after having being trimmed by function δ. This removes $\langle\uparrow\rangle$ elements which correspond to the parallel compositions which have been consumed by the encoding. When a single monitored process $m \triangleright P$ remains, the encoding translates the memory m into the history context $\mathtt{H}_m$, which is the non-standard part of the resulting CCSK process.

Observe that the clauses of the encoding cover all possibilities for reachable processes: remember that each RCCS process can be written as $C_{i\in\{1,\ldots,n\}}[i \mapsto m_i \triangleright P_i]$. If the context contains no parallel composition than clauses for restriction and monitored processes are enough, otherwise the memory m_i of each monitored process contains at least a $\langle\uparrow\rangle$ element and the second clause can be applied.

The following example shows how the encoding works.

Example 5 Let R be the following RCCS process:

$$R = \langle k, d, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle \triangleright a \parallel \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle \triangleright b \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright c$$

Its encoding in CCSK is:

$$\begin{aligned}
(\!|R|\!) &= (\!|\langle k, d, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle \triangleright a \parallel \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle \triangleright b \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright c|\!) \\
&= \mathtt{H}_{\langle\rangle}[(\!|\delta(\langle k, d, \mathbf{0}\rangle \cdot \langle\uparrow\rangle \cdot \langle\rangle) \triangleright a|\!) \parallel (\!|\delta(\langle\uparrow\rangle \cdot \langle\rangle) \triangleright b|\!) \parallel (\!|\delta(\langle\rangle) \triangleright c|\!)] \\
&= (\!|\langle k, d, \mathbf{0}\rangle \cdot \langle\rangle \triangleright a|\!) \parallel (\!|\langle\rangle \triangleright b|\!) \parallel (\!|\langle\rangle \triangleright c|\!) \\
&= \mathtt{H}_{\langle k,d,\mathbf{0}\rangle\cdot\langle\rangle}[a] \parallel \mathtt{H}_{\langle\rangle}[b] \parallel \mathtt{H}_{\langle\rangle}[c] \\
&= \mathtt{H}_{\langle\rangle}[d[k].a] \parallel b \parallel c \\
&= d[k].a \parallel b \parallel c
\end{aligned}$$

We start by presenting some auxiliary results and definitions.

Definition 14 Given a context $C_{l\in L}[l \mapsto \bullet_l]$ we denote with h_l the number of parallel composition operators in the path connecting $\bullet_l$ to the root in the syntax tree of $C_{l\in L}[l \mapsto \bullet_l]$.

Lemma 9 *For any memory $m_1@m_2$ and reachable CCSK process X we have that $\mathtt{H}_{m_1@m_2}[X] = \mathtt{H}_{m_2}[\mathtt{H}_{m_1}[X]]$*

Proof By induction on the length of m_1.

– The base case is when $m_1 = \langle\rangle$. In this case:

$$\mathtt{H}_{m_1@m_2}[X] = \mathtt{H}_{m_2}[X]$$

since $m_1@m_2 = m_2$ and:

$$\mathtt{H}_{m_2}[\mathtt{H}_{m_1}[X]] = \mathtt{H}_{m_2}[X]$$

since $\mathtt{H}_{m_1} = \bullet$. The thesis follows.

– The inductive case is when $m_1 = \langle k, \alpha, \mathbf{0}\rangle \cdot m_1'$. We have that:

$$\mathtt{H}_{m_1@m_2}[X] = \mathtt{H}_{(\langle k,\alpha,\mathbf{0}\rangle\cdot m_1')@m_2}[X] = \mathtt{H}_{m_1'@m_2}[\alpha[k].X]$$

By applying inductive hypothesis on $m_1'@m_2$ we have:

$$\mathtt{H}_{m_1'@m_2}[\alpha[k].X] = \mathtt{H}_{m_2}[\mathtt{H}_{m_1'}[\alpha[k].X]] = \mathtt{H}_{m_2}[\mathtt{H}_{\langle k,\alpha,\mathbf{0}\rangle\cdot m_1'}[X]]$$

as desired.

The case where we have Q instead of $\mathbf{0}$ is analogous. □

Lemma 10 *For each reachable RCCS process of the form $C_{l\in L}[l \mapsto m_l' \triangleright P_l]$, if $m_l' = m_l@\langle\uparrow\rangle \cdot m_l''$ where $\langle\uparrow\rangle \notin m_l''$ then we have*

$$m_l = \delta(m_l)@(\langle\uparrow\rangle^{h_l-1} \cdot \langle\rangle)$$

Proof By induction on the number of steps leading from an initial RCCS process to $C_{l\in L}[l \mapsto m_l' \triangleright P_l]$. The base case is trivial, since the memory of an initial process never contains $\langle\uparrow\rangle$ elements. Let us consider the inductive case. If the reduction is derived without using structural congruence, the thesis follows by inductive hypothesis using Lemma 5. If structural congruence is used, let us consider the application of a single axiom. If the axiom is (Res) or (α), then the thesis follows by inductive hypothesis. If the axiom is (Split), then a memory with depth h_l is split into two memories with depth $h_l + 1$, and a $\langle\uparrow\rangle$ element is added to both of them. For all the other memories the depth is unchanged and no $\langle\uparrow\rangle$ element is added. The thesis follows by inductive hypothesis. □

Lemma 11 *For each reachable CCSK process X and history context $\mathtt{H}_m$ we have*

$$[\![\mathtt{H}_m[X], \langle\rangle]\!] = [\![X, \langle\rangle]\!]@m$$

Proof The proof is by structural induction on m. The case $m = \langle\rangle$ is trivial. Let us consider the case $m = \langle k, \alpha, \mathbf{0}\rangle \cdot m'$ (the case with Q instead of $\mathbf{0}$ is analogous). By definition of history context we have

$$[\![\mathtt{H}_{\langle k,\alpha,\mathbf{0}\rangle\cdot m'}[X], \langle\rangle]\!] = [\![\mathtt{H}_{m'}[\alpha[k].X], \langle\rangle]\!]$$

By inductive hypothesis we have

$$[\![\mathtt{H}_{m'}[\alpha[k].X], \langle\rangle]\!] = [\![\alpha[k].X, \langle\rangle]\!]@m'$$

From the definition of the $[\![\cdot,\cdot]\!]$ encoding we get

$$[\![\alpha[k].X, \langle\rangle]\!]@m' = [\![X, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!]@m'$$

From Lemma 3 we have

$$[\![X, \langle k, \alpha, \mathbf{0}\rangle \cdot \langle\rangle]\!]@m' = [\![X, \langle\rangle]\!]@(\langle k, \alpha, \mathbf{0}\rangle \cdot m')$$

as desired. □

We can now prove the main result of this section, namely that RCCS is more abstract than CCSK. Indeed, by taking an RCCS process R, encoding it in CCSK and encoding the result back in RCCS we get the starting process, as shown by the theorem below. If instead we start from a CCSK process X, we encode it in RCCS and then back to CCSK we get a normalization of X, as shown by Theorem 2.

Theorem 1 *Let R be a reachable RCCS process. Then $[\![(\!|R|\!), \langle\rangle]\!] = R$.*

Proof We perform an induction on the derivation of $(\!|R|\!)$.

$(\!|R\backslash a|\!)$: by using the definitions of the encodings we get

$$[\![(\!|R\backslash a|\!), \langle\rangle]\!] = [\![(\!|R|\!)\backslash a, \langle\rangle]\!] = [\![(\!|R|\!), \langle\rangle]\!]\backslash a$$

since $a \notin \mathtt{fn}(\langle\rangle)$, and by applying the inductive hypothesis we have that $[\![(\!|R|\!), \langle\rangle]\!]\backslash a = R\backslash a$, as desired.

$(\!|C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle \cdot m) \triangleright P_l]|\!)$ with $\langle\uparrow\rangle \notin m$: by using the definition of the encoding $(\!|\cdot|\!)$

$$[\![(\!|C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle \cdot m) \triangleright P_l]|\!), \langle\rangle]\!] = [\![\mathtt{H}_m[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]], \langle\rangle]\!]$$

By using Lemma 11 we have:

$$[\![\mathtt{H}_m[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]], \langle\rangle]\!] = [\![C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)], \langle\rangle]\!]@m$$

From the definition of $[\![\cdot,\cdot]\!]$ we have:

$$[\![C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)], \langle\rangle]\!]@m = (C_{l\in L}[l \mapsto [\![(\!|\delta(m_l) \triangleright P_l|\!), \langle\uparrow\rangle^{h_l} \cdot \langle\rangle]\!]])@m$$

By applying Lemma 3 we have:

$$(C_{l\in L}[l \mapsto [\![(\!|\delta(m_l) \triangleright P_l|\!), \langle\uparrow\rangle^{h_l} \cdot \langle\rangle]\!]])@m = (C_{l\in L}[l \mapsto [\![(\!|\delta(m_l) \triangleright P_l|\!), \langle\rangle]\!]@\langle\uparrow\rangle^{h_l} \cdot \langle\rangle])@m$$

By inductive hypothesis we have:

$$(C_{l\in L}[l \mapsto [\![(\!|\delta(m_l) \triangleright P_l|\!), \langle\rangle]\!]@\langle\uparrow\rangle^{h_l} \cdot \langle\rangle])@m = (C_{l\in L}[l \mapsto (\delta(m_l) \triangleright P_l)@\langle\uparrow\rangle^{h_l} \cdot \langle\rangle])@m$$

By definition of append we have:

$$(C_{l\in L}[l \mapsto (\delta(m_l) \triangleright P_l)@\langle\uparrow\rangle^{h_l} \cdot \langle\rangle])@m = C_{l\in L}[l \mapsto \delta(m_l)@(\langle\uparrow\rangle^{h_l} \cdot \langle\rangle)@m \triangleright P_l]$$

The thesis follows thanks to Lemma 10.

$(\!|m \triangleright P|\!)$: by definition of $(\!|\cdot|\!)$ we have $(\!|m \triangleright P|\!) = \mathtt{H}_m[P]$. By applying $[\![\cdot,\langle\rangle]\!]$ we get:

$$[\![(\!|m \triangleright P|\!), \langle\rangle]\!] = [\![\mathtt{H}_m[P], \langle\rangle]\!]$$

By applying Lemma 11 and the definition of the $[\![\cdot,\cdot]\!]$ encoding we have:

$$[\![\mathtt{H}_m[P], \langle\rangle]\!] = [\![P, \langle\rangle]\!]@m = (\langle\rangle \triangleright P)@m = m \triangleright P$$

as desired. □

Before stating the converse of Theorem 1 we need to define a *normal form* for CCSK processes to match the abstraction made by encoding a CCSK process to RCCS and coming back. Essentially the normal form pushes all the restrictions in the non-standard part of a sequential process outside such sequential process.

Definition 15 (CCSK normal form) The normal form $\mathtt{nf}(X)$ of a CCSK process X is a CCSK process Y obtained from X by applying as many times as possible the following rewriting rules (in any context):

$$\alpha[k].(X_1\backslash a) \to (\alpha[k].X_1\{^b/_a\})\backslash b \qquad \text{if } b \notin \mathtt{fn}(\alpha.\mathbf{0}) \wedge (b = a \vee b \notin \mathtt{fn}(X_1))$$
$$\alpha[k].(X_1\backslash a) + Q \to (\alpha[k].X_1\{^b/_a\} + Q)\backslash b \quad \text{if } b \notin \mathtt{fn}(\alpha.Q) \wedge (b = a \vee b \notin \mathtt{fn}(X_1))$$

with $\neg\mathtt{std}(X_1)$.

To highlight the need for $\mathtt{nf}(\cdot)$ we show some examples:

Example 6 Let $X = a[k].(b[w].P)\backslash a$. We have that:

$$\begin{aligned}&[\![X, \langle\rangle]\!] = [\![(b[w].P)\backslash a, \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle]\!] = [\![(b[w].P\{^c/_a\}), \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle]\!]\backslash c = \\ &[\![P\{^c/_a\}, \langle w, b, \mathbf{0}\rangle \cdot \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle]\!]\backslash c = \\ &(\langle w, b, \mathbf{0}\rangle \cdot \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P\{^c/_a\})\backslash c\end{aligned}$$

Let $m = \langle w, b, \mathbf{0}\rangle \cdot \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle$. By applying the $(\!|\cdot|\!)$ encoding we get:

$$(\!|(m \triangleright P\{^c/_a\})\backslash c|\!) = (\!|(m \triangleright P\{^c/_a\})|\!)\backslash c = \mathtt{H}_m[P\{^c/_a\}]\backslash c = (a[k].b[w].P\{^c/_a\})\backslash c$$

We have that $X \neq (a[k].b[w].P\{^c/_a\})\backslash c$, but $\mathtt{nf}(X) =_\alpha (a[k].b[w].P\{^c/_a\})\backslash c$.

Note that reduction to normal form is the identity if all the restrictions are in the standard part of the process. For instance, consider $X = a[k].(P\backslash a)$. We have:

$$[\![X, \langle\rangle]\!] = [\![P\backslash a, \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle]\!] = \langle k, a, \mathbf{0}\rangle \cdot \langle\rangle \triangleright (P\backslash a)$$

If we apply the $(\!|\cdot|\!)$ encoding to $\langle k, a, \mathbf{0}\rangle \cdot \langle\rangle \triangleright (P\backslash a)$ we obtain:

$$(\!|\langle k, a, \mathbf{0}\rangle \cdot \langle\rangle \triangleright (P\backslash a)|\!) = \mathtt{H}_{\langle k,a,\mathbf{0}\rangle\cdot\langle\rangle}[P\backslash a] = a[k].(P\backslash a) = X = \mathtt{nf}(X)$$

Imagine to use a different definition of normal form $\mathtt{nf}^\bullet(\cdot)$, allowing to pop out restrictions also from the standard part of processes, obtainable by dropping the side condition $\neg\mathtt{std}(X_1)$ from Definition 15. We then would have that:

$$\mathtt{nf}^\bullet(X) = (a[k].P\{^b/_a\})\backslash b$$

but $(\!|[\![X, \langle\rangle]\!]|\!) \neq \mathtt{nf}^\bullet(X)$

We remark here that processes with the same normal form have the same behavior, as shown by the following lemma.

Lemma 12 *Let X and Y be CCSK processes such that $\mathtt{nf}(X) = \mathtt{nf}(Y)$. Then $X \xrightarrow{\alpha[k]} X'$ iff $Y \xrightarrow{\alpha[k]} Y'$ with $\mathtt{nf}(X') = \mathtt{nf}(Y')$, and similarly for backward transitions.*

Proof It is easy to see that the lemma holds for a single application of the rewriting rules used to compute the normal form, both in the direction used for normalization and in the opposite direction. The thesis follows by induction on the number of applications. □

Next lemma proves some properties of the $(\!|\cdot|\!)$ encoding.

Lemma 13 *For each memory m and reachable RCCS processes R and S we have:*

$$(\!|R@m|\!) = \mathtt{nf}(\mathtt{H}_m[(\!|R|\!)]) \quad \textit{where } \langle\uparrow\rangle \notin m \tag{1}$$

$$(\!|R@\langle\uparrow\rangle\cdot\langle\rangle \parallel S@\langle\uparrow\rangle\cdot\langle\rangle|\!) = (\!|R|\!) \parallel (\!|S|\!) \tag{2}$$

Proof The proof of item 1 is by structural induction on R, with a case analysis according to its shape.

$R = R'\backslash a$: we have $(\!|(R'\backslash a)@m|\!) = (\!|R'@m|\!)\backslash a$. By inductive hypothesis $(\!|R'@m|\!) = \mathtt{nf}(\mathtt{H}_m[(\!|R'|\!)])$, hence $(\!|R'@m|\!)\backslash a = \mathtt{nf}(\mathtt{H}_m[(\!|R'|\!)])\backslash a$. Thanks to the definition of the $(\!|\cdot|\!)$ encoding and of normal form: $\mathtt{nf}(\mathtt{H}_m[(\!|R'|\!)])\backslash a = \mathtt{nf}(\mathtt{H}_m[(\!|R'|\!)\backslash a]) = \mathtt{nf}(\mathtt{H}_m[(\!|R'\backslash a|\!)]) = \mathtt{nf}(\mathtt{H}_m[(\!|R|\!)])$ as desired.

$R = C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle\cdot m) \triangleright P_l]$: we have:

$$(\!|R|\!) = (\!|C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle\cdot m') \triangleright P_l]|\!) = \mathtt{H}_{m'}[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]]$$

hence:

$$\begin{aligned}(\!|R@m|\!) =& (\!|C_{l\in L}[l \mapsto m_l@(\langle\uparrow\rangle\cdot m') \triangleright P_l]@m|\!) = \\ & \mathtt{H}_{m'@m}[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]] = \\ & \mathtt{H}_m[\mathtt{H}_{m'}[C_{l\in L}[l \mapsto (\!|\delta(m_l) \triangleright P_l|\!)]]] = \mathtt{H}_m[(\!|R|\!)]\end{aligned}$$

where we used Lemma 9. Note that $\mathtt{H}_m[(\!|R|\!)] = \mathtt{nf}(\mathtt{H}_m[(\!|R|\!)])$ since $\mathtt{H}_m$ and $\mathtt{H}_{m'}$ do not contain restrictions, the top operator of context C is a parallel composition, and the encoding of monitored processes produces CCSK processes already in normal form. Hence, the thesis follows.

$R = m' \triangleright P$: we have $(\!|m' \triangleright P|\!) = \mathtt{H}_{m'}[P]$, hence $(\!|(m' \triangleright P)@m|\!) = \mathtt{H}_{m'@m}[P] = \mathtt{H}_m[\mathtt{H}_{m'}[P]] = \mathtt{H}_m[(\!|R|\!)]$. Note that $\mathtt{H}_m[(\!|R|\!)] = \mathtt{nf}(\mathtt{H}_m[(\!|R|\!)])$ since $\mathtt{H}_m$ and $\mathtt{H}_{m'}$ do not contain restrictions, and P is standard. Hence, the thesis follows.

Item 2 follows immediately from the definition of the encoding, noting that $\mathtt{H}_{\langle\rangle} = \bullet$. □

We can now prove the converse of Theorem 1. As discussed before we need to rely on both normal form and α-conversion.

Theorem 2 *Let X be a reachable CCSK process. Then $(\!|[\![X, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(X)$.*

Proof The proof is by structural induction on X, with a case analysis on the form of X. We first consider the case of standard processes, that is $X = P$.

$X = P$: by using the definitions of encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$ we have:

$$(\!|[\![P, \langle\rangle]\!]|\!) = (\!|\langle\rangle \triangleright P|\!) = \mathtt{H}_{\langle\rangle}[P] = P = \mathtt{nf}(P)$$

as desired.

We now consider non-standard processes.

$X = \alpha[k].Y + \sum_{j\in J} \alpha_j.P_j$: by using the definition of encoding $[\![\cdot]\!]$, we have:

$$(\!|[\![\alpha[k].Y + \sum_{j\in J} \alpha_j.P_j, \langle\rangle]\!]|\!) = (\!|[\![Y, \langle k, \alpha, \sum_{j\in J} \alpha_j.P_j\rangle \cdot \langle\rangle]\!]|\!)$$

Thanks to Lemma 3 we have:

$$(\!|[\![Y, \langle k, \alpha, \sum_{j\in J} \alpha_j.P_j\rangle \cdot \langle\rangle]\!]|\!) = (\!|[\![Y, \langle\rangle]\!]@\langle k, \alpha, \sum_{j\in J} \alpha_j.P_j\rangle \cdot \langle\rangle|\!)$$

By Lemma 13 we have:

$$(\!|[\![Y, \langle\rangle]\!]@\langle k, \alpha, \sum_{j\in J} \alpha_j.P_j\rangle \cdot \langle\rangle|\!) = \mathtt{nf}(\mathtt{H}_{\langle k,\alpha,\sum_{j\in J} \alpha_j.P_j\rangle}[(\!|[\![Y, \langle\rangle]\!]|\!)])$$

Thanks to the inductive hypothesis we have $(\!|[\![Y, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(Y)$, and hence:

$$\begin{aligned}\mathtt{nf}(\mathtt{H}_{\langle k,\alpha,\sum_{j\in J} \alpha_j.P_j\rangle}[(\!|[\![Y, \langle\rangle]\!]|\!)]) =_\alpha & \mathtt{nf}(\mathtt{H}_{\langle k,\alpha,\sum_{j\in J} \alpha_j.P_j\rangle}[\mathtt{nf}(Y)]) = \\ & \mathtt{nf}(\alpha[k].\mathtt{nf}(Y) + \sum_{j\in J} \alpha_j.P_j) = \\ & \mathtt{nf}(\alpha[k].Y + \sum_{j\in J} \alpha_j.P_j)\end{aligned}$$

as desired.

$X = X_1 \parallel X_2$: by definition of the $[\![\cdot]\!]$ encoding, we have:

$$(\!|[\![X_1 \parallel X_2, \langle\rangle]\!]|\!) = (\!|[\![X_1, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![X_2, \langle\uparrow\rangle \cdot \langle\rangle]\!]|\!)$$

Thanks to Lemma 3 we can rewrite $[\![X_1, \langle\uparrow\rangle \cdot \langle\rangle]\!]$ and $[\![X_2, \langle\uparrow\rangle \cdot \langle\rangle]\!]$ as:

$$\begin{aligned}[\![X_1, \langle\uparrow\rangle \cdot \langle\rangle]\!] = C_{i\in I}^{[\![X_1,\langle\rangle]\!]}[i \mapsto m_i@(\langle\uparrow\rangle \cdot \langle\rangle) \triangleright P_i] = \\ C_{i\in I}^{[\![X_1,\langle\rangle]\!]}[i \mapsto m_i \triangleright P_i]@(\langle\uparrow\rangle \cdot \langle\rangle) = \\ [\![X_1, \langle\rangle]\!]@(\langle\uparrow\rangle \cdot \langle\rangle) \\ [\![X_2, \langle\uparrow\rangle \cdot \langle\rangle]\!] = C_{j\in J}^{[\![X_2,\langle\rangle]\!]}[j \mapsto m_j@(\langle\uparrow\rangle \cdot \langle\rangle) \triangleright P_j] = \\ C_{j\in J}^{[\![X_2,\langle\rangle]\!]}[j \mapsto m_j \triangleright P_j]@(\langle\uparrow\rangle \cdot \langle\rangle) = \\ [\![X_2, \langle\rangle]\!]@(\langle\uparrow\rangle \cdot \langle\rangle)\end{aligned}$$

and obtain:

$$(\!|[\![X_1 \parallel X_2, \langle\rangle]\!]|\!) = (\!|[\![X_1, \langle\rangle]\!]@(\langle\uparrow\rangle \cdot \langle\rangle) \parallel [\![X_2, \langle\rangle]\!]@(\langle\uparrow\rangle \cdot \langle\rangle)|\!)$$

Thanks to Lemma 13 we have:

$$(\!|[\![X_1 \parallel X_2, \langle\rangle]\!]|\!) = (\!|[\![X_1, \langle\rangle]\!]|\!) \parallel (\!|[\![X_2, \langle\rangle]\!]|\!))$$

By inductive hypothesis we have that $(\!|[\![X_1, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(X_1)$ and $(\!|[\![X_2, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(X_2)$. Moreover, by definition of the normal form we have that:

$$\mathtt{nf}(X_1 \parallel X_2) = \mathtt{nf}(X_1) \parallel \mathtt{nf}(X_2)$$

Then we can conclude by noticing that:

$$(\!|[\![X_1, \langle\rangle]\!]|\!) \parallel (\!|[\![X_2, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(X_1) \parallel \mathtt{nf}(X_2) = \mathtt{nf}(X_1 \parallel X_2) = \mathtt{nf}(X)$$

as desired.

$X = X_1\backslash a$: by using the definition of encodings $[\![\cdot]\!]$ and $(\!|\cdot|\!)$, we have:

$$(\!|[\![X_1\backslash a, \langle\rangle]\!]|\!) = (\!|[\![X_1, \langle\rangle]\!]\backslash a|\!) = (\!|[\![X_1, \langle\rangle]\!]|\!)\backslash a$$

By inductive hypothesis we have $(\!|[\![X_1, \langle\rangle]\!]|\!) =_\alpha \mathtt{nf}(X_1)$ and we can conclude by noticing that:

$$(\!|[\![X_1, \langle\rangle]\!]|\!)\backslash a =_\alpha \mathtt{nf}(X_1)\backslash a = \mathtt{nf}(X_1\backslash a) = \mathtt{nf}(X)$$

□

Using the theorem above, we can now prove operational correspondence results for the encoding.

Proposition 5 (Forward Correctness and Completeness) *Let R and S be two reachable RCCS processes. There exists an RCCS transition $R \xrightarrow{k,\alpha} S$ iff there exists a CCSK transition $(\!|R|\!) \xrightarrow{\alpha[k]} (\!|S'|\!)$ with $S \equiv S'$.*

Proof $\Rightarrow$) Since from Theorem 1 $R = [\![(\!|R|\!), \langle\rangle]\!]$ and $S = [\![(\!|S|\!), \langle\rangle]\!]$, we have that $[\![(\!|R|\!), \langle\rangle]\!] \xrightarrow{k,\alpha} [\![(\!|S|\!), \langle\rangle]\!]$, then by Proposition 3 we have that $(\!|R|\!) \xrightarrow{\alpha[k]} (\!|S'|\!)$ with $S' \equiv S$.

$\Leftarrow$) If $(\!|R|\!) \xrightarrow{\alpha[k]} (\!|S'|\!)$ with $S' \equiv S$, then by Proposition 1 we have that $[\![(\!|R|\!), \langle\rangle]\!] \xrightarrow{k,\alpha} [\![(\!|S'|\!), \langle\rangle]\!]$. Then from Theorem 1 we have $R \xrightarrow{k,\alpha} S'$. The thesis follows by applying RCCS rule R-EQUIV. □

Proposition 6 (Backward Correctness and Completeness) *Let R and S be two reachable RCCS processes. There exists an RCCS transition $R \overset{k,\alpha}{\rightsquigarrow} S$ iff there exists a CCSK transition $(\!|R|\!) \overset{\alpha[k]}{\rightsquigarrow} (\!|S'|\!)$ with $S \equiv S'$.*

Proof The proof is similar to the one of Proposition 5. □

5 Isomorphism and Compositionality

In this section we first combine the operational correspondence results for the two encodings to show that the LTS of RCCS up to structural congruence and the LTS of CCSK up to normal form and α-conversion are isomorphic. We then discuss compositionality issues related to the encodings.

We first introduce the notion of isomorphism.

Definition 16 (LTS Isomorphism) Two LTSs $LTS_i : \langle \mathcal{P}_i, \mathcal{L}_i, \rightarrow_i \rangle$, with $i \in \{1, 2\}$ are *isomorphic* iff there exist two bijective functions $\gamma_L : \mathcal{L}_1 \rightarrow \mathcal{L}_2$ and $\gamma_P : \mathcal{P}_1 \rightarrow \mathcal{P}_2$ such that $P_1 \xrightarrow{\alpha} P_2$ iff $\gamma_P(P_1) \xrightarrow{\gamma_L(\alpha)} \gamma_P(P_2)$.

We will prove isomorphisms between the two forward LTSs and between the two backward LTSs.

We define below the quotient of an LTS w.r.t. an equivalence relation on states.

Definition 17 (LTSs up to equivalence) Given an $LTS : \langle \mathcal{P}, \mathcal{L}, \rightarrow \rangle$ and an equivalence relation $\sim$ on $\mathcal{P}$, we denote by $[P]$ the equivalence class of $P \in \mathcal{P}$. We define the LTS up to $\sim$ as $LTS_{\sim} : \langle \mathcal{P}_{\sim}, \mathcal{L}, \rightarrow_{\sim} \rangle$ where $\mathcal{P}_{\sim}$ is the set of equivalence classes of $\mathcal{P}$ modulo $\sim$, and $[P_1] \xrightarrow{l}_{\sim} [P_2]$ iff there exist $P_1 \in [P_1]$ and $P_2 \in [P_2]$ such that $P_1 \xrightarrow{l} P_2$.

We define the equivalence relation $\overset{ccsk}{\sim}$ on CCSK processes as follows: $X \overset{ccsk}{\sim} Y$ iff $\mathtt{nf}(X) =_{\alpha} \mathtt{nf}(Y)$. On RCCS we just consider structural congruence as equivalence relation.

We need a few auxiliary lemmas.

Lemma 14 *For each CCSK processes X and Y, $[\![X, \langle\rangle]\!] \equiv [\![Y, \langle\rangle]\!]$ implies $[\![X, \langle\rangle]\!] = [\![Y, \langle\rangle]\!]$.*

Proof RCCS structural congruence just allows one to split threads with a toplevel parallel composition, and to put restrictions which are at toplevel inside threads outside the same threads. One can easily notice that in the encoding threads correspond to standard processes, hence parallel threads are split and restrictions moved outside iff the corresponding processes are not standard. The thesis follows. □

Lemma 15 *Let X and Y be CCSK processes. If $X \overset{ccsk}{\sim} Y$ then $[\![X, \langle\rangle]\!] =_{\alpha} [\![Y, \langle\rangle]\!]$.*

Proof By definition of $\overset{ccsk}{\sim}$ we have $\mathtt{nf}(X) =_{\alpha} \mathtt{nf}(Y)$. It is trivial to see that encoding CCSK processes equal up to α-conversion gives RCCS processes equivalent up to α-conversion. One can also notice that the encoding moves all the restrictions in the non-standard part outside the corresponding process, hence the thesis follows. □

We can now prove our main result.

Theorem 3 (RCCS and CCSK are isomorphic) *The forward LTS of RCCS up to structural congruence is isomorphic to the forward LTS of CCSK up to $\overset{ccsk}{\sim}$. Both LTSs are restricted to reachable processes. The same result holds for backward LTSs.*

Proof We will show the thesis for the forward LTSs. The thesis for backward LTSs follows by using the Loop Lemma for CCSK and RCCS.

The function on labels maps $\alpha[k]$ to k, α and is trivially bijective. The function on states maps an equivalence class of CCSK processes $[X]$ into $[[\![X, \langle\rangle]\!]]$. We remark that the function is well defined thanks to Lemma 15. In order to show that it is bijective we show that the function $(\!|\cdot|\!)$ is its inverse. It is easy to see that also function $(\!|\cdot|\!)$ is well defined on equivalence classes since structural congruence just allows one to move restrictions and parallel composition between monitored processes and processes, but this difference is immaterial in CCSK. Also, both calculi have α-conversion. Given an RCCS process R we have $[\![(\!|R|\!), \langle\rangle]\!] = R$ from Theorem 1. Given a CCSK process X we have $(\!|[\![X, \langle\rangle]\!]|\!) =_{\alpha} \mathtt{nf}(X)$ from Theorem 2. This proves bijectivity.

We need to show that $[X_1] \xrightarrow{\alpha[k]}_{\overset{ccsk}{\sim}} [X_2]$ iff $[[\![X_1, \langle\rangle]\!]] \xrightarrow{k,\alpha}_{\equiv} [[\![X_2, \langle\rangle]\!]]$. By expanding the definitions of LTS up to equivalence, we need to show that there

exist X_1' and X_2' such that $X_1' \xrightarrow{\alpha[k]} X_2'$ with $\mathtt{nf}(X_1') =_\alpha \mathtt{nf}(X_1)$ and $\mathtt{nf}(X_2') =_\alpha \mathtt{nf}(X_2)$ iff there exist S_1 and S_2 such that $[\![X_1, \langle\rangle]\!] \equiv S_1 \xrightarrow{k,\alpha} S_2 \equiv [\![X_2, \langle\rangle]\!]$. By applying RCCS rule R-EQUIV the second part can be equivalently written as $[\![X_1, \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X_2, \langle\rangle]\!]$.

Let us show the implication from left to right. Using Lemma 12 we obtain $\mathtt{nf}(X_1') \xrightarrow{\alpha[k]} X_2''$ with $\mathtt{nf}(X_2'') = \mathtt{nf}(X_2')$. Using again Lemma 12 and α-conversion we get $X_1 \xrightarrow{\alpha[k]} X_2'''$ with $\mathtt{nf}(X_2''') =_\alpha \mathtt{nf}(X_2'')$. As a result we also have $\mathtt{nf}(X_2''') =_\alpha \mathtt{nf}(X_2'')$. From Proposition 1 we have $[\![X_1, \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X_2''', \langle\rangle]\!]$. From Lemma 15 we have $[\![X_2''', \langle\rangle]\!] =_\alpha [\![X_2', \langle\rangle]\!]$ as desired.

Let us show the implication from right to left. From $[\![X_1, \langle\rangle]\!] \xrightarrow{k,\alpha} [\![X_2, \langle\rangle]\!]$ using Proposition 3 we have that $X_1 \xrightarrow{\alpha[k]} X_2'$ in CCSK, with $[\![X_2, \langle\rangle]\!] \equiv [\![X_2', \langle\rangle]\!]$. From Lemma 14 we have that $[\![X_2, \langle\rangle]\!] = [\![X_2', \langle\rangle]\!]$. The thesis follows. □

We remark that the result above is very strong, since isomorphism implies all reasonable behavioral equivalences, including back and forth bisimilarity used in [35]. Nevertheless, such a result can only be obtained via encodings which are *not uniform* [38]. We recall below the notion of uniform encoding.

Definition 18 (Uniform Encoding) An encoding $(\!|\cdot|\!)$ is *uniform* iff it is:

homomorphic w.r.t. the parallel composition operator: given two processes, $\mathtt{R}_1$ and $\mathtt{R}_2$, $(\!|\mathtt{R}_1 \parallel \mathtt{R}_2|\!) = (\!|\mathtt{R}_1|\!) \parallel (\!|\mathtt{R}_2|\!)$,

renaming preserving: given a process $\mathtt{R}$ and an injective renaming function σ mapping names to names and keys to keys, $(\!|\mathtt{R}\sigma|\!) = (\!|\mathtt{R}|\!)\sigma$.

The only difference between the definition above and the one in [38] is that we require renamings to map names to names and keys to keys. This is justified by the fact that names and keys are distinct entities and should not be mixed.

It is easy to see that our encoding of CCSK into RCCS is not uniform.

Proposition 7 *The encoding $[\![\bullet]\!]$ of CCSK into RCCS is not uniform.*

Proof It is easy to see that the encoding is not homomorphic w.r.t. parallel composition, since:

$$[\![a \parallel b]\!] = [\![a \parallel b, \langle\rangle]\!] = [\![a, \langle\uparrow\rangle \cdot \langle\rangle]\!] \parallel [\![b, \langle\uparrow\rangle \cdot \langle\rangle]\!] = \langle\uparrow\rangle \cdot \langle\rangle \triangleright a \parallel \langle\uparrow\rangle \cdot \langle\rangle \triangleright b$$

is not equal to:

$$[\![a]\!] \parallel [\![b]\!] = [\![a, \langle\rangle]\!] \parallel [\![b, \langle\rangle]\!] = \langle\rangle \triangleright a \parallel \langle\rangle \triangleright b$$

Notably, this last process is not reachable in RCCS. □

The fact that the encoding is not uniform is not a coincidence. Indeed, RCCS reachable processes are not closed under parallel composition, in a very strong sense.

Proposition 8 *Given two reachable RCCS processes R and S, there is no context $C[1 \mapsto \cdot, 2 \mapsto \cdot]$ such that $C[1 \mapsto R, 2 \mapsto S]$ is reachable.*

Proof Consider a reachable process R. It is easy to show by induction on the derivation from the initial process to R that the following invariant holds: each memory m inside R contains a number of $\langle\uparrow\rangle$ elements equal to the number of parallel composition operators in the path from the root to m. Given that C adds at least one parallel composition operator in the path to R and to S, if the invariant holds for R and S it cannot hold for their parallel composition. Note in fact that additional $\langle\uparrow\rangle$ elements need to be added inside memories, hence cannot be part of the C context. □

As a consequence of the proposition above, there exist no uniform encoding (Definition 18) from CCSK to RCCS. Even more, there is no encoding mapping the CCSK parallel operator to any RCCS context. This is formalized by the following corollary.

Corollary 1 *Let $(\!|\bullet|\!)^{any}$ be any encoding of CCSK reachable processes into RCCS reachable processes. Then there exists no RCCS context $C[1 \mapsto \cdot, 2 \mapsto \cdot]$ such that $(\!|R \parallel S|\!)^{any} = C[1 \mapsto (\!|R|\!)^{any}, 2 \mapsto (\!|S|\!)^{any}]$.*

Proof Assume towards a contradiction that such an encoding exists. Take any reachable CCSK process of the form $R \parallel S$. In CCSK, R and S are also reachable. Then, $(\!|R|\!)^{any}$, $(\!|S|\!)^{any}$ and $(\!|R \parallel S|\!)^{any} = C[1 \mapsto (\!|R|\!)^{any}, 2 \mapsto (\!|S|\!)^{any}]$ need to be all reachable, but this is impossible because of Proposition 8. This concludes the proof. □

Notably, this rules out any encoding homomorphic w.r.t. the parallel composition operator, and in particular uniform encodings.

For the encoding from RCCS to CCSK we cannot even state a uniformity result, since the terms in the statement would not be all reachable.

6 Cross-fertilization Results

In this section we exploit the theory developed in the previous sections to prove some cross-fertilization results. In particular, we import the CCSK property called *Reverse Diamond Property* into RCCS, and, in the other direction, we import the RCCS property called *Parabolic Lemma* into CCSK.

The Reverse Diamond Property [42, Proposition 5.10], reported below, states that backward transitions are confluent.

Proposition 9 (CCSK Reverse Diamond Property) *Let X, Y and Z be reachable CCSK processes.*

1. *if $X \overset{\alpha[k]}{\rightsquigarrow} Y$ and $X \overset{\beta[k]}{\rightsquigarrow} Z$ then $\alpha = \beta$ and $Y = Z$.*
2. *if $X \overset{\alpha[k]}{\rightsquigarrow} Y$ and $X \overset{\beta[h]}{\rightsquigarrow} Z$ with $k \neq h$ then there exists W such that $Y \overset{\beta[h]}{\rightsquigarrow} W$ and $Z \overset{\alpha[k]}{\rightsquigarrow} W$.*

We can now import this result in RCCS.

Proposition 10 (RCCS Reverse Diamond Property) *Let R, R_1 and R_2 be reachable RCCS processes.*

1. *if* $R \overset{k,\alpha}{\rightsquigarrow} R_1$ *and* $R \overset{k,\beta}{\rightsquigarrow} R_2$ *then* $\alpha = \beta$ *and* $R_1 \equiv R_2$.
2. *if* $R \overset{k,\alpha}{\rightsquigarrow} R_1$ *and* $R \overset{h,\beta}{\rightsquigarrow} R_2$ *with* $k \neq h$ *then there exists* S *such that* $R_1 \overset{h,\beta}{\rightsquigarrow} S$ *and* $R_2 \overset{k,\alpha}{\rightsquigarrow} S$.

Proof We have two cases, one for each item in the statement.

1. By hypothesis we have that $R \overset{k,\alpha}{\rightsquigarrow} R_1$ and $R \overset{k,\beta}{\rightsquigarrow} R_2$. By applying Proposition 6 we have that $(\!|R|\!) \overset{\alpha[k]}{\rightsquigarrow} (\!|R'_1|\!)$ with $R'_1 \equiv R_1$ and $(\!|R|\!) \overset{\beta[k]}{\rightsquigarrow} (\!|R'_2|\!)$ with $R'_2 \equiv R_2$. By Reverse Diamond Property on CCSK transitions (Proposition 9) we have that $(\!|R'_1|\!) = (\!|R'_2|\!)$ and $\alpha = \beta$. By Theorem 1 we have that $[\![(\!|R'_1|\!), \langle\rangle]\!] = R'_1$ and $[\![(\!|R'_2|\!), \langle\rangle]\!] = R'_2$, with $R'_1 = R'_2$. Since $R_1 \equiv R'_1$ and $R'_2 \equiv R'_2$ we also have $R_1 \equiv R_2$ as desired.
2. By hypothesis we have that $R \overset{k,\alpha}{\rightsquigarrow} R_1$ and $R \overset{h,\beta}{\rightsquigarrow} R_2$ with $k \neq h$. By applying Proposition 6 we have that $(\!|R|\!) \overset{\alpha[k]}{\rightsquigarrow} (\!|R'_1|\!)$ with $R'_1 \equiv R_1$ and $(\!|R|\!) \overset{\beta[h]}{\rightsquigarrow} (\!|R'_2|\!)$ with $R'_2 \equiv R_2$. Since, $k \neq h$ by Reverse Diamond Property on CCSK transitions (Proposition 9) we have that there exists W such that $(\!|R'_1|\!) \overset{\beta[h]}{\rightsquigarrow} W$ and $(\!|R'_2|\!) \overset{\alpha[k]}{\rightsquigarrow} W$. Let S be a RCCS process such that $(\!|S|\!) = W$. By Proposition 6 we have that $(\!|R'_1|\!) \overset{\beta[h]}{\rightsquigarrow} (\!|S|\!)$ implies $R'_1 \overset{h,\beta}{\rightsquigarrow} S'$ with $S' \equiv S$, and $(\!|R'_2|\!) \overset{\alpha[k]}{\rightsquigarrow} (\!|S|\!)$ implies $R'_2 \overset{k,\alpha}{\rightsquigarrow} S''$ with $S'' \equiv S$. Since $R_1 \equiv R'_1$, $S' \equiv S$, $R_2 \equiv R'_2$ and $S'' \equiv S$ by applying RCCS rule R-EQUIV we have that $R_1 \overset{h,\beta}{\rightsquigarrow} S$ and $R_2 \overset{k,\alpha}{\rightsquigarrow} S$, as desired. □

The Parabolic Lemma [15, Lemma 10], reported below, states that each reversible computation can be decomposed into a backward computation followed by a forward one. Intuitively, this means that the process can first go backward so to enable as many choices as possible, and then go only forward.

Proposition 11 (RCCS Parabolic Lemma) *For any reachable RCCS process* R, *if* $R \rightleftharpoons \cdots \rightleftharpoons S$, *then there exists* R' *such that* $R \rightsquigarrow^* R' \rightarrow^* S$.

We can now import the Parabolic Lemma in CCSK. We remark that CCSK Parabolic Lemma has been proved in the literature [42, Lemma 5.12], yet the direct proof is more complex than importing the result from RCCS.

Proposition 12 (CCSK Parabolic Lemma) *For any reachable CCSK process* X, *if* $X \rightleftharpoons \cdots \rightleftharpoons Y$, *then there exists* X' *such that* $X \rightsquigarrow^* X' \rightarrow^* Y$.

Proof By hypothesis we have that $X \rightleftharpoons \cdots \rightleftharpoons Y$. Let $R = [\![X, \langle\rangle]\!]$ and $S = [\![Y, \langle\rangle]\!]$. We can apply Proposition 1 to each forward transition and Proposition 2 to each backward transition to obtain a sequence of transitions $R = [\![X, \langle\rangle]\!] \rightleftharpoons \cdots \rightleftharpoons [\![Y, \langle\rangle]\!] = S$. We can then apply Proposition 11 to the sequence of transitions above and obtain that there exists R' such that $R \rightsquigarrow^* R' \rightarrow^* S$. Since $R \rightsquigarrow^* R'$ with $R = [\![X, \langle\rangle]\!]$ by applying Proposition 4 (and rule R-EQUIV) we have that $X \rightsquigarrow^* X'$ with $[\![X', \langle\rangle]\!] \equiv R'$. From the reduction $R' \rightarrow^* S$, by applying Proposition 3 (and rule R-EQUIV) we obtain that $X' \rightarrow^* Y$ with $[\![Y, \langle\rangle]\!] \equiv S$, as desired. □

7 Conclusions, Related and Future Work

In this paper we have shown that the two main forms of reversible CCS, namely RCCS [15] and CCSK [42], give rise to isomorphic LTSs, hence they are different syntactic representations for the same behaviors. Nevertheless, the syntactic differences have an impact on their possible uses and extensions. CCSK is more compositional, as discussed in Section 5, hence more easy to work with. On the other side, the approach of RCCS is more flexible, as proved by the fact that it has been successfully applied also to π-calculus [13]. We note however that the mentioned extension is far from trivial.

Related Work In the literature another approach to define reversible calculi exists, but it has never been applied to CCS. Indeed, it has been first defined for higher-order π-calculus [31,32]. However, applying this approach to CCS is trivial, as shown below, since CCS is a subcalculus of higher-order π.

In [31,32], reversibility in higher-order π calculus is obtained by using thread tags which act as unique identifiers (a role close to the one of keys here), and process terms, called memories, which are dedicated to undo a single forward step. For instance, a computational step of the process $a\langle P\rangle|a(X)\triangleright Q$, where process P is sent on channel a and replaced for variable X, is:

$$(\kappa_1 : a\langle P\rangle)|(\kappa_2 : a(X)\triangleright Q) \rightarrow \nu k.k : Q\{P/X\}|[M;k]$$

where $M = (\kappa_1 : a\langle P\rangle)|(\kappa_2 : a(X)\triangleright Q)$. The two processes participating in the communication are uniquely identified by tags κ_1 and κ_2. The resulting process $Q\{P/X\}$ is tagged with the fresh name k where ν is the restriction operator in π-calculus. Additionally, memory process $[M;k]$ is created to record the configuration on the left side of the reduction. The corresponding backward step discards the process tagged with k and reinstates the configuration M. We can apply the same approach to CCS process $a \parallel \overline{a}.Q$, obtaining:

$$(\kappa_1 : \overline{a}) \parallel (\kappa_2 : a.Q) \rightarrow \nu k.k : Q \parallel [M;k]$$

with $M = (\kappa_1 : \overline{a}) \parallel (\kappa_2 : a.Q)$. Notably, the approach is for asynchronous calculi (where the output has no continuation, such as in the processes above), but one can extend it to synchronous calculi, as shown by the following example:

$$(\kappa_1 : \overline{a}.P) \parallel (\kappa_2 : a.Q) \rightarrow \nu k_1 \nu k_2.k_1 : P \parallel k_2 : Q \parallel [M;k_1;k_2]$$

with $M = (\kappa_1 : \overline{a}.P) \parallel (\kappa_2 : a.Q)$.

The main difference between the approach above and the RCCS and CCSK approaches is that the one above deals only with reduction semantics and not with LTS semantics. Hence, the downside of this approach is that it can only be applied to closed systems. The upside is that its application to more complex calculi such as higher-order π [32] or Klaim [22] is simpler. However, no formal relation between this approach and the RCCS and CCSK approaches exists in the literature. More in general, while the number of concurrent reversible calculi and languages is increasing, there is very little literature on the relations between them. We are only aware of [33], where a classification of the different approaches is presented. However, the classification is just at a descriptive level. Some similarities between

different reversible π-calculi are highlighted in [36], where a parametric framework for reversible π-calculi is presented. By varying the parameters, three notions of causality can be considered [5,12,13], and for two of them the corresponding reversible calculus can be defined [5,12]. Notably, different notions of causality imply different restrictions on the allowed order of backward steps.

Future Work The problem of understanding the relations between the different approaches stays open and we plan to further investigate it in future work. Another open point for future work is looking for other cross-fertilization results along the lines of the ones described in Section 6. Finally, we note that the approach of CCSK is very close to event transition systems [9], which are related also to flow event structures [7,11] and flow nets [8,6]. Further analysis of this connection may shed light on the classes of calculi to which the CCSK approach can be applied and the ones to which it cannot be applied.

Acknowledgment

We would like to thank Irek Ulidowski for insightful discussions on CCSK and opinions on an earlier version of the encoding from RCCS to CCSK. We would also like to thank the anonymous reviewers of the present paper for their useful remarks and suggestions, which led to substantial improvements.

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. J. C. M. Baeten and C. Verhoef. A congruence theorem for structured operational semantics with predicates. In E. Best, editor, *CONCUR '93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*, volume 715 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 1993.
3. A. Berut, A. Arakelyan, A. Petrosyan, S. Ciliberto, R. Dillenschneider, and E. Lutz. Experimental verification of Landauer' s principle linking information and thermodynamics. *Nature*, 483(7388):187–189, 03 2012.
4. B. Boothe. Efficient algorithms for bidirectional debugging. In *Proceedings of the ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation*, PLDI '00, pages 299–310, New York, NY, USA, 2000. ACM.
5. M. Boreale and D. Sangiorgi. A fully abstract semantics for causality in the pi-calculus. In *STACS 95, 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 2-4, 1995, Proceedings*, pages 243–254, 1995.
6. G. Boudol. Flow event structures and flow nets. In *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings*, pages 62–95, 1990.
7. G. Boudol and I. Castellani. Permutation of transitions: An event structure semantics for CCS and SCCS. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings*, pages 411–427, 1988.
8. G. Boudol and I. Castellani. Flow models of distributed computations : event structures and nets. Research Report RR-1482, INRIA, 1991.
9. G. Boudol and I. Castellani. Flow models of distributed computations: Three equivalent semantics for CCS. *Inf. Comput.*, 114(2):247–314, 1994.
10. L. Cardelli and C. Laneve. Reversible structures. In F. Fages, editor, *Computational Methods in Systems Biology, 9th International Conference, CMSB 2011, Paris, France, September 21-23, 2011. Proceedings*, pages 131–140. ACM, 2011.

11. I. Castellani and G. Zhang. Parallel product of event structures. *Theor. Comput. Sci.*, 179(1-2):203–215, 1997.
12. S. Crafa, D. Varacca, and N. Yoshida. Event structure semantics of parallel extrusion in the pi-calculus. In *Foundations of Software Science and Computational Structures - 15th International Conference, FOSSACS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 225–239, 2012.
13. I. Cristescu, J. Krivine, and D. Varacca. A compositional semantics for the reversible pi-calculus. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 388–397. IEEE Computer Society, 2013.
14. I. Cristescu, J. Krivine, and D. Varacca. Rigid families for the reversible π-calculus. In *Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings*, volume 9720 of *Lecture Notes in Computer Science*, pages 3–19. Springer, 2016.
15. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR 2004 - Concurrency Theory, 15th International Conference, London, UK, August 31 - September 3, 2004, Proceedings*, pages 292–307, 2004.
16. V. Danos and J. Krivine. Transactions in RCCS. In *CONCUR 2005 - Concurrency Theory, 16th International Conference, CONCUR 2005, San Francisco, CA, USA, August 23-26, 2005, Proceedings*, pages 398–412, 2005.
17. V. Danos and J. Krivine. Formal molecular biology done in CCS-R. In *Proceedings of the First Workshop on Concurrent Models in Molecular Biology (BioConcur 2003)*, volume 180(3) of *Electr. Notes Theor. Comput. Sci.*, pages 31–49. Elsevier, 2007.
18. V. Danos, J. Krivine, and F. Tarissan. Self-assembling trees. In *Proceedings of the Third Workshop on Structural Operational Semantics (SOS 2006)*, volume 175(1) of *Electr. Notes Theor. Comput. Sci.*, pages 19–32. Elsevier, 2007.
19. E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Comput. Surv.*, 34(3):375–408, Sept. 2002.
20. J. Engblom. A review of reverse debugging. In *System, Software, SoC and Silicon Debug Conference (S4D), 2012*, pages 1–6, Sept 2012.
21. E. Giachino, I. Lanese, and C. A. Mezzina. Causal-consistent reversible debugging. In *Fundamental Approaches to Software Engineering - 17th International Conference, FASE 2014*, pages 370–384, 2014.
22. E. Giachino, I. Lanese, C. A. Mezzina, and F. Tiezzi. Causal-consistent rollback in a tuple-based language. *J. Log. Algebr. Meth. Program.*, 88:99–120, 2017.
23. E. Graversen, I. Phillips, and N. Yoshida. Event structure semantics of (controlled) reversible CCS. In *Reversible Computation - 10th International Conference, RC 2018, Leicester, UK, September 12-14, 2018, Proceedings*, pages 102–122, 2018.
24. J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1992.
25. J. Krivine. A verification technique for reversible process algebra. In *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3, 2012. Revised Papers*, pages 204–217, 2012.
26. S. Kuhn and I. Ulidowski. Towards modelling of local reversibility. In *Reversible Computation - 7th International Conference, RC 2015, Grenoble, France, July 16-17, 2015, Proceedings*, volume 9138 of *Lecture Notes in Computer Science*, pages 279–284. Springer, 2015.
27. S. Kuhn and I. Ulidowski. A calculus for local reversibility. In *Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings*, volume 9720 of *Lecture Notes in Computer Science*, pages 20–35. Springer, 2016.
28. O. Laadan and J. Nieh. Transparent checkpoint-restart of multiple processes on commodity operating systems. In *2007 USENIX Annual Technical Conference on Proceedings of the USENIX Annual Technical Conference*, ATC'07, pages 25:1–25:14, Berkeley, CA, USA, 2007. USENIX Association.
29. R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183–191, 1961.
30. I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, and J.-B. Stefani. Concurrent flexible reversibility. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013*, pages 370–390, 2013.

31. I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order pi. In P. Gastin and F. Laroussinie, editors, *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010.
32. I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversibility in the higher-order π-calculus. *Theor. Comput. Sci.*, 625:25–84, 2016.
33. I. Lanese, C. A. Mezzina, and F. Tiezzi. Causal-consistent reversibility. *Bulletin of the EATCS*, 114, 2014.
34. I. Lanese, N. Nishida, A. Palacios, and G. Vidal. Cauder: A causal-consistent reversible debugger for Erlang. In J. P. Gallagher and M. Sulzmann, editors, *Functional and Logic Programming - 14th International Symposium, FLOPS 2018, Nagoya, Japan, May 9-11, 2018, Proceedings*, volume 10818 of *Lecture Notes in Computer Science*, pages 247–263. Springer, 2018.
35. D. Medic and C. A. Mezzina. Static VS dynamic reversibility in CCS. In S. J. Devitt and I. Lanese, editors, *Reversible Computation - 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings*, volume 9720 of *Lecture Notes in Computer Science*, pages 36–51. Springer, 2016.
36. D. Medic, C. A. Mezzina, I. Phillips, and N. Yoshida. A parametric framework for reversible pi-calculi. In *Proceedings Combined 25th International Workshop on Expressiveness in Concurrency and 15th Workshop on Structural Operational Semantics and 15th Workshop on Structural Operational Semantics, EXPRESS/SOS 2018, Beijing, China, September 3, 2018.*, pages 87–103, 2018.
37. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
38. C. Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.
39. K. S. Perumalla and A. J. Park. Reverse computation for rollback-based fault tolerance in large parallel systems - evaluating the potential gains and systems effects. *Cluster Computing*, 17(2):303–313, 2014.
40. I. Phillips and I. Ulidowski. Reversibility and models for concurrency. In M. H. R. van Glabbeek, editor, *Proceedings of the Fourth Workshop on Structural Operational Semantics (SOS 2007)*, volume 192(1) of *Electr. Notes Theor. Comput. Sci.*, pages 93–108. Elsevier, 2007.
41. I. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Reversible Computation, 4th International Workshop, RC 2012, Copenhagen, Denmark, July 2-3, 2012. Revised Papers*, pages 218–232, 2012.
42. I. C. C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2):70–96, 2007.
43. J. J. P. Tsai, K. Fang, H. Chen, and Y. Bi. A noninterference monitoring and replay mechanism for real-time software testing and debugging. *IEEE Trans. Software Eng.*, 16(8):897–916, 1990.