Static VS Dynamic Reversibility in CCS*

Doriana Medić and Claudio Antares Mezzina

IMT Institute for Advanced Studies Lucca, Italy
doriana.medic@imtlucca.it, claudio.mezzina@imtlucca.it

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. Reversible CCS (RCCS), proposed by Danos and Krivine, enacts reversibility by means of memory stacks. Ulidowski and Phillips proposed a general method to reverse a process calculus given in a particular SOS format, by exploiting the idea of making all the operators of a calculus static. CCSK is then derived from CCS with this method. In this paper we show that RCCS is at least as expressive as CCSK.

1 Introduction

The interest in reversibility dates back to the 60's, with Landauer [6] observing that only irreversible computations need to consume energy, fostering application of reversible computing in scenarios of low-energy computing. Landauer's principle has only been shown empirically in 2012 [1]. Nowadays reversible computing is attracting interests because of its applications in diverse fields: biological modelling [10], since many biochemical reactions are by nature reversible; program debugging and testing [4], allowing during debugging time to bring the program state back to a certain execution point in which certain conditions are met [8]; and parallel discrete event simulations [9]. 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, system-recovery schemes and checkpoint-rollback protocols, 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. Good examples on how reversibility in CCS and Higher-Order π can be used to model transactional models are respectively [3] and [7].

The first reversible variant of CCS, called RCCS, was introduced by Danos and Krivine [2]. In RCCS each process is monitored by a *memory*, that serves as stack of past actions. Memories are considered as unique process identifiers, and in order to preserve this uniqueness along a parallel composition, a structural

* Research partly supported by the EU COST Action IC1405.

$$P, Q ::= \mathbf{0} \mid a.P \mid \overline{a}.P \mid \tau.P \mid (P \parallel Q) \mid \sum \alpha_i.P \mid P\backslash A$$

Fig. 1. CCS Syntax

law permits to obtain unique memories though a parallel composition. A general method for reversing process calculi, given in a particular SOS format, has been proposed by Phillips and Ulidowski in [11]. The main idea of this approach is the use of communication keys to uniquely identify communications, and to make *static* each operator of the calculus. By applying this method to CCS, CCSK is obtained. Since in CCSK the history is directly annotated in the process itself, there is no need of splitting history through a parallel composition. We call this kind of recording histories as *static* reversibility; while we call the one used by RCCS as *dynamic*, since each thread is endowed with its own history. Hence a natural question arises: are these two reversible calculi equivalent? In this paper we start answering to this question by showing that RCCS is at least as expressive as CCSK. We do it by means of an encoding and show its correctness by means of strong back and forth bisimulaiton.

The rest of the paper is organized as follows: Section 2 starts with a brief recall to the syntax of CCS. In Section 2.1 will present RCCS with its syntax and semantics. Section 2.2 will be about CCSK and its semantics. In Section 3 we will present our encoding function from CCSK to RCCS and prove our main result. In Section 4 we will sketch an encoding of RCCS into CCKS and discuss about the difficulties it takes to prove its correctness. Section 5 concludes the paper with a discussion of the future work.

2 CCS and its reversible variants

In this section we briefly present the syntax of CCS [8], and then we show the two reversible extensions of it, namely RCCS [2] and CCSK [11].

Let $\mathcal{A}$ the set of actions such that $a \in \mathcal{A}$, and $\overline{\mathcal{A}}$ the set of co-actions such that $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$. 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 τ is the *silent* action.

The syntax of CCS is given in Figure 1. $\mathbf{0}$ represents the idle process. A prefix (or action) can be an input a, an output $\overline{a}$ and the silent action τ. $P \parallel Q$ represents the parallel composition of processes P and Q, while $\sum \alpha_i.P$ represents the guarded choice. Some actions in a process P can be restricted, and this is represented by the process $P\backslash A$, where A is the set of restricted actions. The set $\mathcal{P}$ denotes the set of all possible CCS processes.

2.1 Reversible CCS

One of approaches to make CCS reversible is to add a memory to each process. A memory then will be recording every action and communication that the

process will undergo. Syntax of RCCS is given in Figure 2. As we can see,

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

Fig. 2. RCCS syntax

RCCS processes are built on top of CCS processes. A term of the form $m \triangleright P$, is called *monitored* process, where m represents a memory carrying the information that this process will need in case it wants to backtrack, and P is a standard CCS process. Two monitored process R and S can be composed in parallel $S \parallel R$, and some actions of a monitored process R can be restricted via $R\backslash A$. *Memories* are organised as stacks of events, with the top of the memory representing the very last action of the monitored process. $\langle\rangle$ represent the empty memory; $\langle i, \alpha, Q\rangle$ represent an action event meaning that the monitored process did the action α identified by i and its "context" was Q; while $\langle\uparrow\rangle$ represents a splitting event. When there is no ambiguity, we will omit the trailing event $\langle\rangle$ in memories.

We assume the existence of a infinite denumerable set of action *identifiers* (sometimes called keys) $\mathcal{K}$ such that $\mathcal{K} \cap \mathtt{Act} = \emptyset$. Let $\mathtt{ActK} = \mathtt{Act} \times \mathcal{K}$ the set of pairs formed by an action μ and an identifier i. In the same way we define $\mathtt{ActK}_\tau = \mathtt{Act}_\tau \times \mathcal{K}$. The operational semantics of RCCS is defined as a labelled transition system (LTS), $(\mathcal{P}_R, \rightarrow \cup \rightsquigarrow, \mathtt{ActK}_\tau)$ where $\mathcal{P}_R$ is the set of RCCS (monitored) processes, $\rightarrow \subseteq \mathcal{P}_R \times \mathtt{ActK}_\tau \times \mathcal{P}_R$ and $\rightsquigarrow \subseteq \mathcal{P}_R \times \mathtt{ActK}_\tau \times \mathcal{P}_R$. Relations $\rightarrow$ and $\rightsquigarrow$ are the smallest reduction relations induced by respectively rules in Figure 4 and Figure 5. Both reduction relations exploit the structural congruence $\equiv$ relation, which is the smallest congruence, on processes and monitored processes, containing the abelian monoid laws for choice (that is *commutativity*, *associativity* and $\mathbf{0}$ as the identity element) and the rules of Figure 3.

Remark 1. In its first incarnation ([2]) RCCS used events of this form $\langle n_*, \alpha, Q\rangle$, $\langle 1\rangle$ and $\langle 2\rangle$, where: n_* is n in case the process synchonized with a process monitored by memory n or $*$ is case of partial synchronization. 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 version we are using, appeared in [5], simplifies the handling of memories and makes the splitting through the parallel composition commutative. However they are conceptually the same, and we have chosen this version since it simplifies some technicalities when dealing with the proof of our main Theorem.

Identifiers of RCCS are similar to communication keys of CCSK. They are defined as follows:

$$\text{(Split)} \quad m \triangleright (P \parallel Q) \equiv (\langle\uparrow\rangle \cdot m \triangleright P \parallel \langle\uparrow\rangle \cdot m \triangleright Q)$$
$$\text{(Res)} \quad m \triangleright P\backslash A \equiv (m \triangleright P)\backslash A$$

Fig. 3. RCCS Structural laws

Definition 1 (Memory identifiers). *The set of identifiers of a memory m, written $\mathtt{id}(m)$, is inductively defined as follows:*

$$\mathtt{id}(\langle\uparrow\rangle \cdot m) = \mathtt{id}(m) \qquad \mathtt{id}(\langle i, \alpha, Q\rangle \cdot m) = \{i\} \cup \mathtt{id}(m)$$

Definition 2. *A identifier i belongs to a memory m, written $i \in m$, if $i \in \mathtt{id}(m)$.*

Definition 3 (Process identifiers). *The set of identifiers of a process R, written $\mathtt{id}(R)$, is inductively defined as follows:*

$$\mathtt{id}(\alpha.P) = \mathtt{id}(\mathbf{0}) = \emptyset \qquad \mathtt{id}(m \triangleright P) = \mathtt{id}(m)$$
$$\mathtt{id}(R\backslash A) = \mathtt{id}(R) \qquad \mathtt{id}(R \parallel S) = \mathtt{id}(R) \cup \mathtt{id}(S)$$

Definition 4. *A identifier i belongs to a process R, written $i \in R$, if $i \in \mathtt{id}(R)$.*

$$\text{(R-ACT)}\ \frac{i \notin m}{m \triangleright \alpha.P + Q \rightarrow^i_\alpha \langle i, \alpha, Q\rangle \cdot m \triangleright P} \qquad \text{(R-PAR)}\ \frac{R \rightarrow^i_\alpha R' \qquad i \notin S}{R \parallel S \rightarrow^i_\alpha R' \parallel S}$$

$$\text{(R-SYN)}\ \frac{R \rightarrow^i_\alpha R' \qquad S \rightarrow^i_{\bar\alpha} S'}{R \parallel S \rightarrow^i_\tau R' \parallel S'} \qquad \text{(R-RES)}\ \frac{R \rightarrow^i_\alpha R' \qquad \alpha \notin A \cup \bar A}{R\backslash A \rightarrow^i_\alpha R'\backslash A}$$

$$\text{(R-EQUIV)}\ \frac{R \equiv R \qquad R' \rightarrow^i_\alpha S' \qquad S' \equiv S}{R \rightarrow^i_\alpha S}$$

Fig. 4. RCCS forward semantics

Let us now comment on the forward rules of Figure 4. Rule R-ACT allows a monitored process to perform a forward action. As we can see, this action is bound with a particular fresh identifier i. Moreover, the part of the process which has not contributed to the action, that is Q, is stored on top of the memory along with the action and the identifier. Rule R-PAR propagates an action along a parallel composition, with the condition that the identifier of the action is not used by other processes. This check guarantees that all the identifiers are unique. Rule R-SYN allows two processes in parallel to syncrhonize. To do so, they have to match both the action and the identifier. Rule R-RES deals with restriction

$$\text{(R-ACT}^\bullet\text{)}\ \frac{i \notin m}{\langle i, a, Q\rangle \cdot m \triangleright P \rightsquigarrow^i_\alpha m \triangleright \alpha.P + Q} \qquad \text{(R-PAR}^\bullet\text{)}\ \frac{R \rightsquigarrow^i_\alpha R' \qquad i \notin S}{R \parallel S \rightsquigarrow^i_\alpha R' \parallel S}$$

$$\text{(R-SYN}^\bullet\text{)}\ \frac{R \rightsquigarrow^i_\alpha R' \qquad S \rightsquigarrow^i_{\bar{\alpha}} S'}{R \parallel S \rightsquigarrow^i_\tau R' \parallel S'} \qquad \text{(R-RES}^\bullet\text{)}\ \frac{R \rightsquigarrow^i_\alpha R' \qquad \alpha \notin A \cup \bar{A}}{R\backslash A \rightsquigarrow^i_\alpha R'\backslash A}$$

$$\text{R-EQUIV}^\bullet\ \frac{R \equiv R' \qquad R' \rightsquigarrow^i_\alpha S' \qquad S' \equiv S}{R \rightsquigarrow^i_\alpha S}$$

Fig. 5. RCCS backward semantics

in the normal way, while rule R-EQUIV brings structural equivalence into the reduction relation.

Backward rules are reported in Figure 5. For each of forward rule there exists an opposite 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 build back the previous form of the process, that is the prefix and the process that was composed with the + operator. Rule R-PAR$^\bullet$ allows a reversible action to be propagated through a parallel composition, only when the identifier of the action does not belong to monitored processes in parallel. This check is crucial to avoid partial undo of some synchronizations. The remaining rules are similar to the forward ones.

Definition 5 (Reachable Process). *A RCCS process R is reachable if it can be derived from an initial process $\langle\rangle \triangleright P$, by using rules of Figure 4 and Figure 5.*

Lemma 1. *For any transition $m \triangleright \alpha.P + Q \rightarrow^i_\alpha \langle i, \alpha, Q\rangle \cdot m \triangleright P$ we can derive the following transitions:*

- $\langle j, \beta, Q_1\rangle \cdot m \triangleright \alpha.P + Q \rightarrow^i_\alpha \langle i, \alpha, Q\rangle \cdot \langle j, \beta, Q_1\rangle \cdot m \triangleright P$, *for* $i \neq j$
- $\langle\uparrow\rangle \cdot m \triangleright \alpha.P + Q \rightarrow^i_\alpha \langle i, \alpha, Q\rangle \cdot \langle\uparrow\rangle \cdot m \triangleright P$

and its opposite:

Lemma 2. *For any transition $\langle i, \alpha, Q\rangle \cdot m \triangleright P \rightsquigarrow^i_\alpha \langle\rangle \triangleright \alpha.P + Q$, we can derive the following transitions:*

- $\langle i, \alpha, Q\rangle \cdot \langle j, \beta, Q_1\rangle \cdot m \triangleright P \rightsquigarrow^i_\alpha \langle j, \beta, Q_1\rangle \cdot m \triangleright \alpha.P + Q$, *for* $i \neq j$
- $\langle i, \alpha, Q\rangle \cdot \langle\uparrow\rangle \cdot m \triangleright P \rightsquigarrow^i_\alpha \langle\uparrow\rangle \cdot m \triangleright \alpha.P + Q$

An easy induction on the structure of terms provides us with a kind of normal form for RCCS processes (by convention $\prod_{i\in I} R_i = \mathbf{0}$ if $I = \emptyset$):

Lemma 3 (Normal form). *For any RCCS reachable process R we have that*

$$R \equiv (\prod_{i\in I}(m_i \triangleright \alpha_i.P_i + Q_i)\backslash A_i)\backslash B$$

2.2 CCS with communication keys

The main idea behind this approach is to directly record the actions inside a process and to make all the operator of CCS static. In this way there is no need of using an external memory, since all the information are syntactically presents inside a term. Syntax of CCSK is given in Figure 6. The only difference with respect to CCS processes is that prefixes now can be annotated with an identifiers.

$$\begin{array}{rl} \text{(CCS Processes)} & P, Q ::= \mathbf{0} \mid a.P \mid \overline{a}.P \mid \tau.P \mid (P \parallel Q) \mid \sum \alpha_i.P_i \mid P\backslash A \\ \text{(CCSK Processes)} & X, Y ::= P \mid \alpha[i].X \mid X + Y \mid (X \parallel Y) \mid X\backslash A \end{array}$$

Fig. 6. CCSK syntax

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

$$\begin{array}{ll} \mathtt{key}(\alpha.P) = \mathtt{key}(\mathbf{0}) = \emptyset & \mathtt{key}(\alpha[i].X) = \{i\} \cup \mathtt{key}(X) \\ \mathtt{key}(X \parallel Y) = \mathtt{key}(X) \cup \mathtt{key}(Y) & \mathtt{key}(X + Y) = \mathtt{key}(X) \cup \mathtt{key}(Y) \\ \mathtt{key}(X\backslash A) = \mathtt{key}(X) & \end{array}$$

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

Definition of keys in CCSK correspond to the definition of identifiers in RCCS. The operational semantics of CCSK is defined as a labelled transition system (LTS), $(\mathcal{P}_K, \rightarrow \cup \rightsquigarrow, \mathtt{ActK}_\tau)$ where $\mathcal{P}_K$ is the set of CCSK processes, $\rightarrow \subseteq \mathcal{P}_R \times \mathtt{ActK}_\tau \times \mathcal{P}_R$ and $\rightsquigarrow \subseteq \mathcal{P}_R \times \mathtt{ActK}_\tau \times \mathcal{P}_R$. Relations $\rightarrow$ and $\rightsquigarrow$ are the smallest reduction relations induced by respectively rules in Figure 7 and Figure 8. Differently from RCCS, CCSK does not exploit any structural congruence.

Remark 2. In the following when in proofs, rules and so on we use P instead of X we just indicate that the process P has no labelled actions, as P being a CCS process. An alternative is to use predicate $\mathtt{std}(X)$ as in [11].

Rules for forward transitions are given in Figure 7. Rule K-ACT1 deals with prefixed processes $\alpha.P$. It just transforms a prefix into a label but differently from the normal CCS rule for prefix, it generates a fresh new key i which is bound to the action α becoming $\alpha[i]$. As we can note the prefix is not discarded after the reduction. Rule K-ACT-2 inductively allows a prefixed process $\alpha[i].X$ to execute if X can execute. The actions that X can do are forced to use keys different from i. Rules K-PLUS-L and K-PLUS-R deal with the $+$ operator. Let us note that these rule do not discard the context, that is part of the process which has not contributed to the action. In more detail, if the process $P+Q$ does an action, say $\alpha[i]$, and becomes X then the process becomes $X+Q$. In this way the information about $+Q$ is preserved. Moreover since Q is a standard process

$$\text{(K-ACT1)}\ \frac{}{\alpha.P \xrightarrow{\alpha[i]} \alpha[i].P} \qquad \text{(K-ACT2)}\ \frac{X \xrightarrow{\beta[j]} X' \qquad i \neq j}{\alpha[i].X \xrightarrow{\beta[j]} \alpha[i].X'}$$

$$\text{(K-PLUS-L)}\ \frac{X \xrightarrow{\alpha[i]} X'}{X + P \xrightarrow{\alpha[i]} X' + P} \qquad \text{(K-PLUS-R)}\ \frac{Y \xrightarrow{\alpha[i]} Y'}{P + Y \xrightarrow{\alpha[i]} P + Y'}$$

$$\text{(K-PAR-L)}\ \frac{X \xrightarrow{\alpha[i]} X' \qquad \texttt{fresh}(i, Y)}{X \parallel Y \xrightarrow{\alpha[i]} X' \parallel Y} \qquad \text{(K-PAR-R)}\ \frac{Y \xrightarrow{\alpha[i]} Y' \qquad \texttt{fresh}(i, X)}{X \parallel Y \xrightarrow{\alpha[i]} X \parallel Y'}$$

$$\text{(K-SYN)}\ \frac{X \xrightarrow{\alpha[i]} X' \quad Y \xrightarrow{\bar{\alpha}[i]} Y' \quad a \neq \tau}{X \parallel Y \xrightarrow{\tau[i]} X' \parallel Y'} \qquad \text{(K-RES)}\ \frac{X \xrightarrow{\alpha[i]} X' \qquad \alpha \notin A \cup \bar{A}}{X \backslash A \xrightarrow{\alpha[i]} X' \backslash A}$$

Fig. 7. CCSK forward semantics

then it will never executes even if it is present in the process $X+Q$. So we can say that $+Q$ is just a decoration of X. Let us note that in order to apply one of the plus rule one of the two processes has to be a CCS process P (e.g. not containing labelled prefixes), meaning that it is impossible for two non standard process to both execute if composed by the choice operator. Rules K-PAR-L and K-PAR-R propagate an action $\alpha[i]$ through a parallel composition, provided that the key i 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 syncrhonize. To do so, they have to match both the action and the identifier. Rule K-RES deals with restriction in the canonical (CCS) way. Backward rules are the exact opposite of the forward ones.

Definition 8 (Reachable Process). *A CCSK process X is reachable if it can be derived from an CCS process P, by using rules of Figure 7 and Figure 8.*

Property 1 (Plus form). If X is a reachable process, and $X = Y + Q$, then

$$Y = P_1 + \ldots + (Y_1 \parallel \ldots \parallel Y_m) + P_j + \ldots + P_n$$

for some α, m, n and with P_i not having top level $+$.

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.

3 Encoding CCSK in RCCS

We now adapt the concept of *bisimulation* ([12]) to work in a reversible setting and with two different semantics. To this aim, we indicate with $\rightarrow_{s_i}$ the forward

$$\text{(K-ACT1}^\bullet\text{)}\ \frac{}{\alpha[i].P \overset{\alpha[i]}{\rightsquigarrow} \alpha.P} \qquad \text{(K-ACT2}^\bullet\text{)}\ \frac{X \overset{\beta[j]}{\rightsquigarrow} X' \qquad i \neq j}{\alpha[i].X \overset{\beta[j]}{\rightsquigarrow} \alpha[i].X'}$$

$$\text{(K-PLUS-L}^\bullet\text{)}\ \frac{X \overset{\alpha[i]}{\rightsquigarrow} X'}{X + P \overset{\alpha[i]}{\rightsquigarrow} X' + P} \qquad \text{(K-PLUS-R}^\bullet\text{)}\ \frac{Y \overset{\alpha[i]}{\rightsquigarrow} Y'}{P + Y \overset{\alpha[i]}{\rightsquigarrow} P + Y'}$$

$$\text{(K-PAR-L}^\bullet\text{)}\ \frac{X \overset{\alpha[i]}{\rightsquigarrow} X' \qquad \texttt{fresh}(i, Y)}{X \parallel Y \overset{\alpha[i]}{\rightsquigarrow} X' \parallel Y} \qquad \text{(K-PAR-R}^\bullet\text{)}\ \frac{Y \overset{\alpha[i]}{\rightsquigarrow} Y' \qquad \texttt{fresh}(i, X)}{X \parallel Y \overset{\alpha[i]}{\rightsquigarrow} X \parallel Y'}$$

$$\text{(K-SYN}^\bullet\text{)}\ \frac{X \overset{\alpha[i]}{\rightsquigarrow} X' \ \ Y \xrightarrow{\bar{\alpha}[i]} Y' \ \ \alpha \neq \tau}{X \parallel Y \overset{\tau[i]}{\rightsquigarrow} X' \parallel Y'} \qquad \text{(K-RES}^\bullet\text{)}\ \frac{X \overset{\alpha[i]}{\rightsquigarrow} X' \qquad \alpha \notin A \cup \bar{A}}{X \backslash A \overset{\alpha[i]}{\rightsquigarrow} X' \backslash A}$$

Fig. 8. CCSK backward semantics

relation of the s_i semantics, and with $\rightsquigarrow_{s_i}$ the backward one. Moreover, we indicate with $\mathcal{P}_{s_i}$ the set of processes of semantics s_i and with $\mathcal{L}_{s_i}$ the set of labels produced by semantics s_i.

Definition 9 (Back and Forth Bisimulation). *Given a bijective function $\gamma : \mathcal{L}_{s_1} \to \mathcal{L}_{s_2}$, a relation ${}_{s_1}\mathcal{R}_{s_2} \subseteq \mathcal{P}_{s_1} \times \mathcal{P}_{s_2}$ is a strong back and forth simulation if whenever $P\, {}_{s_1}\mathcal{R}_{s_2} R$:*

- *$P \xrightarrow{\alpha[i]}_{s_1} Q$ implies $R \xrightarrow{\gamma(\alpha[i])}_{s_2} S$ with $Q\, {}_{s_1}\mathcal{R}_{s_2} S$*
- *$P \overset{\alpha[i]}{\rightsquigarrow}_{s_1} Q$ implies $R \overset{\gamma(\alpha[i])}{\rightsquigarrow}_{s_2} S$ with $Q\, {}_{s_1}\mathcal{R}_{s_2} S$*

A relation ${}_{s_1}\mathcal{R}_{s_2} \subseteq \mathcal{P}_{s_1} \times \mathcal{P}_{s_2}$ is called strong back and forth bisimulation if ${}_{s_1}\mathcal{R}_{s_2}$ and $({}_{s_1}\mathcal{R}_{s_2})^{-1}$ are strong back and forth simulations. We call strong bisimilarity and note ${}_{s_1} \sim_{s_2}$ the largest bisimulation with respect to semantics s_1 and s_2.

This definition when instantiated with a single semantics, that is $s_1 = s_2$ and γ being the *identity*, is similar to the definition of *forward-reverse* bisimulation used in [11], with the only difference is that our definition does not take into account predicates. Moreover, when instantiated with CCSK semantics, the two notions coincide.

In this section we will show how CCSK can be encoded in RCCS. We will use the same notation like before. P stands for processes from CCS and X for CCSK processes. Let $\mathcal{P}_K$ and $\mathcal{P}_R$ the set of processes from CCSK and RCCS, respectively, and $\mathcal{M}$ is the set of all the memories derivable by productions in Figure 2. The encoding function $[\![\cdot]\!] : \mathcal{P}_K \times \mathcal{M} \times \mathcal{P} \to \mathcal{P}_R$, is inductively defined

as follows:

$$\begin{aligned}
[\![P, m, \mathbf{0}]\!] &= m \triangleright P \\
[\![X + P, m, Q]\!] &= [\![X, m, P + Q]\!] \\
[\![P + X, m, Q]\!] &= [\![X, m, Q + P]\!] \\
[\![\alpha[i].X, m, P]\!] &= [\![X, \langle i, \alpha, P\rangle \cdot m, \mathbf{0}]\!] \\
[\![X\backslash A, m, P]\!] &= [\![X, m, P]\!]\backslash A \\
[\![X \parallel Y, m, P]\!] &= [\![X, \langle\uparrow\rangle \cdot m, P]\!] \parallel [\![Y, \langle\uparrow\rangle \cdot m, P]\!]
\end{aligned}$$

Let us comment it. The main difference between RCCS and CCSK is on the way they keep track of the history. In RCCS all the information is local to each monitored process, while in CCSK the information is spread along the structure of a process. Moreover, a CCSK process may correspond to several monitored processes, since in CCSK there is no need of splitting memories through a parallel composition. So the encoding has to inductively drill the structure of a CCSK process X, in order to build the final memory of the process and to find the plus context of each labelled action $\alpha[i]$ present inside X. To this aim, the encoding takes two additional parameters: a memory m and a CCS process P. The parallel and the restriction of CCSK operator are mapped to the corresponding operators of RCCS. Let us note that in the parallel case, the memory m is split into two $\langle\uparrow\rangle \cdot m$. The encoding of a process $\alpha[i].X$ with memory m and context Q is the encoding of process X where the memory stack is augmented of the event $\langle i, \alpha, Q\rangle$. In this case the action $\alpha[i]$ disappears from the process as it goes inside the memory m. The encoding of a process $P + X$ is the encoding of X where its context is the sum composition of its previous context and P. Finally, the encoding of a normal CCS process P is just its monitored version, with memory m representing its history. Since the context parameter is used for past actions, in the case of normal process P, we impose this parameter to be $\mathbf{0}$. In order to understand how the encoding works let us consider the following example. Let $X = (a + b) + c[i].(d[h] \parallel P)$ then

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

Before stating our main Theorem, we need some lemmata about operational correspondence.

Lemma 4 (Forward Correspondence). *For all transitions $X \xrightarrow{\alpha[i]} X'$ in CCSK, with $R = [\![X, \langle\rangle, \mathbf{0}]\!]$, there exists a corresponding RCCS transition such that $R \rightarrow^{i}_{\alpha} R'$ with $[\![X', \langle\rangle, \mathbf{0}]\!] = R'$.*

Proof. By induction on the derivation $X \xrightarrow{\alpha[i]} X'$ and by case analysis on the last applied rule. We show the relevant cases:

K-ACT2: We have $\alpha[i].X \xrightarrow{\beta[j]} \alpha[i].X'$ with $X \xrightarrow{\beta[j]} X'$. Be $R = [\![X, \langle\rangle, \mathbf{0}]\!]$, by Lemma 3 we know that: $R \equiv (\prod_{i\in I}(m_i \triangleright \alpha_i.P_i + Q_i)\backslash A_i)\backslash B$

By applying inductive hypothesis we have that $[\![X, \langle\rangle, \mathbf{0}]\!] \rightarrow^j_\beta [\![X', \langle\rangle, \mathbf{0}]\!]$, that is $R \rightarrow^j_\beta R'$ with $R' = [\![X', \langle\rangle, \mathbf{0}]\!]$. Now we to distinguish two cases: either β is a single action or it has been produced by a synchonization. In the first case we have then that there exists an index $h \in I$ such that $\alpha_h = \beta$, and then

$$[\![X, \langle\rangle, \mathbf{0}]\!] \equiv (\prod_{i\in I}(m_i \triangleright \alpha_i.P_i + Q_i)\backslash A_i)\backslash B \rightarrow^j_\beta$$

$$(\prod_{i\in I\backslash h}(m_i \triangleright \alpha_i.P_i + Q_i)\backslash A_i \parallel (\langle j, \beta, Q_h\rangle \cdot m_h \triangleright P_h)\backslash A_h)\backslash B \equiv [\![X', \langle\rangle, \mathbf{0}]\!]$$

Moreover, by definition of encoding we have that

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

and by using Lemma 1 we can mimic the same transition with an augmented memory: $[\![X, \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle, \mathbf{0}]\!] \rightarrow^j_\beta [\![X', \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle, \mathbf{0}]\!]$ as desired. The synchronisation case is similar.

K-PLUS-L: We have $X = Y + P \xrightarrow{\alpha[i]} Y' + P$. By Property 1, we have that:

$$Y = P_1 + \ldots + (Y_1 \parallel \ldots \parallel Y_m) + P_j + \ldots + P_n$$

Let $T = \sum_{i\in n\backslash l} P_i$, by applying the encoding we have that

$$[\![Y + P, \langle\rangle, \mathbf{0}]\!] = [\![(Y_1 \parallel \ldots \parallel Y_m), \langle\rangle, P + T]\!]$$

$$[\![Y, \langle\rangle, \mathbf{0}]\!] = [\![(Y_1 \parallel \ldots \parallel Y_m), \langle\rangle, T]\!]$$

By Lemma 3 we know that: $[\![Y + P, \langle\rangle]\!] \equiv (\prod_{l\in I}(m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l)\backslash B$

This implies that there exists a subset $J \subseteq I$ on indexes such that memories in J share the action $\langle k, \beta, T\rangle$, with $T = \sum_{i\in n\backslash l} P_i$, such that:

$$[\![Y, \langle\rangle, \mathbf{0}]\!] \equiv (\prod_{l\in I\backslash J}(m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h\in J}(m_h \cdot \langle k, \beta, T\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h)\backslash B$$

$$[\![Y + P, \langle\rangle, \mathbf{0}]\!] \equiv (\prod_{l\in I\backslash J}(m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h\in J}(m_h \cdot \langle k, \beta, T + P\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h)\backslash B$$

By hypothesis we have that $Y \xrightarrow{\alpha[i]} Y$ and by inductive hypothesis we have that $[\![Y, \langle\rangle, \mathbf{0}]\!] \rightarrow^i_\alpha [\![Y', \langle\rangle, \mathbf{0}]\!]$, but then also $[\![Y + P, \langle\rangle, \mathbf{0}]\!] \rightarrow^i_\alpha [\![Y', \langle\rangle, P]\!] = [\![Y' + P, \langle\rangle, \mathbf{0}]\!]$, as desired. ■

Lemma 5 (Backward correspondance). *For all transitions $X \overset{\alpha[i]}{\rightsquigarrow} X'$ in CCSK, with $R = [\![X, \langle\rangle, \mathbf{0}]\!]$, there exists a corresponding transition $R \rightsquigarrow^{i}_{\alpha} R'$ in RCCS with $[\![X', \langle\rangle, \mathbf{0}]\!] = R'$.*

Proof. By induction on the derivation $X \overset{\alpha[i]}{\rightsquigarrow} X'$ and by case analysis on the last applied rule. The proof follows the lines of the one of Lemma 4.

With the previous two lemmata we have proved that if we have a couple of processes $(X, R) = (X, [\![X, \langle\rangle, \mathbf{0}]\!])$ where X is reachable, and if process X does an action α in CCSK, then process R does the same action in RCCS. Obtained process $R' = [\![X', \langle\rangle, \mathbf{0}]\!]$ is still encoding of process X'. Now we have to show the opposite direction.

Lemma 6 (Forward Completeness). *For any CCSK process X and RCCS process R, such that $R = [\![X, \langle\rangle, \mathbf{0}]\!]$, if $R \rightarrow^{i}_{\alpha} R'$ in RCCS, then there exists a corresponding transition $X \xrightarrow{\alpha[i]} X'$ in CCSK, with $R' = [\![X', \langle\rangle, \mathbf{0}]\!]$.*

Proof. By structural induction on X. We have two main cases, whether $X = P$ or not. We will show just the most significant cases.

In first case we observe form of the processes $X = P$, where P is (standard) CCS process. We then do an induction of the form of P. If $P = \alpha.P_1$: we have that $R = [\![\alpha.P_1, \langle\rangle, \mathbf{0}]\!]$ and by applying encoding

$$[\![\alpha.P_1, \langle\rangle, \mathbf{0}]\!] = \langle\rangle \triangleright \alpha.P_1$$

Then, by using R-ACT we get $\langle\rangle \triangleright \alpha.P_1 \rightarrow^{i}_{\alpha} \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P_1$, where $\langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle \triangleright P_1 = [\![\alpha[i].P_1, \langle\rangle, \mathbf{0}]\!] = R'$.
In CCSK process $\alpha.P_1$, can do the same action α by applying the rule K-ACT1 and we get

$$\alpha.P_1 \xrightarrow{\alpha[i]} \alpha[i].P_1 \text{ where } X' = \alpha[i].P_1 \text{ as we desired.}$$

In the second case we observe form of the processes X, when he have a structure of CCSK process and it is not standard process. We consider the significant cases:

$X = \alpha[i].Y$ **:** we have that $R = [\![\alpha[i].Y, \langle\rangle, \mathbf{0}]\!]$. By Lemma 3 we know that:

$$[\![Y, \langle\rangle, \mathbf{0}]\!] \equiv (\prod_{l \in I} (m_l \triangleright \alpha_l.P_l + Q_l) \backslash A_l) \backslash B$$

Now we know that there exists some subset $H \subseteq I$ such that all processes from that subset share the very first action α . For some $t \in H$ such that

$\alpha_t = \beta$ we have:

$$\begin{aligned}[\![Y, \langle\rangle, \mathbf{0}]\!] \equiv (&\prod_{l\in I\setminus H} (m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h\in H\setminus t} (m'_h \cdot \langle i, \alpha, \mathbf{0}\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h \parallel \\ &(m'_t \cdot \langle i, \alpha, \mathbf{0}\rangle \triangleright \beta.P_t + Q_t)\backslash A_t\ )\backslash B \rightarrow^j_\beta \\ &(\prod_{l\in I\setminus H} (m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h\in H\setminus t} (m'_h \cdot \langle i, \alpha, \mathbf{0}\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h \parallel \\ &\langle j, \beta, Q_t\rangle \cdot m'_t \cdot \langle i, \alpha, \mathbf{0}\rangle \triangleright P_t\backslash A_t)\backslash B \equiv [\![Y', \langle\rangle, \mathbf{0}]\!]\end{aligned}$$

By definition of encoding we have that $[\![\alpha[i].Y, \langle\rangle, \mathbf{0}]\!] = [\![Y, \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle, \mathbf{0}]\!]$ and using Lemma 1 we can mimic the same transition:

$$[\![Y, \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle, \mathbf{0}]\!] \rightarrow^j_\beta [\![Y', \langle i, \alpha, \mathbf{0}\rangle \cdot \langle\rangle, \mathbf{0}]\!]$$

By inductive hypothesis we have that $Y \xrightarrow{\beta[j]} Y'$ and using rule K-ACT2 we get:

$$\alpha[i].Y \xrightarrow{\beta[j]} \alpha[i].Y' \quad \text{as desired.}$$

$X = Y_1 \parallel Y_2$ We have that $R = [\![Y_1 \parallel Y_2, \langle\rangle, \mathbf{0}]\!]$ and by applying encoding

$$[\![Y_1 \parallel Y_2, \langle\rangle, \mathbf{0}]\!] = [\![Y_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \parallel [\![Y_2, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!]$$

Now, we distinguish three cases: if first branch of parallel composition do action α, or the second one, or α is sunchronization action. If $R_1 = [\![Y_1, \langle\rangle, \mathbf{0}]\!]$ in first case we have that in R_1 exists an index $h \in I$ such that $\alpha_h = \alpha$ and then by lemma 3, we get

$$\begin{aligned}R_1 \equiv &(\prod_{l\in I}(m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l)\backslash B \rightarrow^i_\alpha \\ &(\prod_{l\in I\setminus h} (m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel (\langle i, \alpha, Q_h\rangle \cdot m_h \triangleright P_h)\backslash A_h)\backslash B \equiv R'_1\end{aligned}$$

By Lemma 1 we have: $[\![Y_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \rightarrow^i_\alpha [\![Y'_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!]$
Using rule R-PAR we get

$$[\![Y_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \parallel [\![Y_2, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \rightarrow^i_\alpha [\![Y'_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \parallel [\![Y_2, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!]$$

where $[\![Y'_1, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] \parallel [\![Y_2, \langle\uparrow\rangle \cdot \langle\rangle, \mathbf{0}]\!] = [\![Y'_1 \parallel Y_2, \langle\rangle, \mathbf{0}]\!] = R'$. By inductive hypothesis we have that also $Y_1 \xrightarrow{\alpha[i]} Y'_1$ and by using rule K-PAR-L, we get: $Y_1 \parallel Y_2 \xrightarrow{\alpha[i]} Y'_1 \parallel Y_2$ as desired. The remaining cases are similar.

$X = Y + P$ **:** By Property 1 we have that:

$$Y = P_1 + \ldots + (Y_1 \parallel \ldots \parallel Y_m) + P_j + \ldots + P_n$$

In the same way, like in Lemma 4 we define processes congruent to processes $[\![Y + P, \langle\rangle, \mathbf{0}]\!]$ and $[\![Y, \langle\rangle, \mathbf{0}]\!]$.

Now we know that there exists some $t \in J$ such that $\alpha_t = \alpha$ and we have

$$(\prod_{l \in I \setminus J} (m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h \in J} (m_h \cdot \langle k, \beta, T\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h \parallel$$
$$(m_t \cdot \langle h, \beta, T\rangle \triangleright \alpha.P_t + Q_t)\backslash A_t)\backslash B \rightarrow^i_\alpha$$
$$(\prod_{l \in I \setminus J} (m_l \triangleright \alpha_l.P_l + Q_l)\backslash A_l \parallel \prod_{h \in J} (m_h \cdot \langle k, \beta, T\rangle \triangleright \alpha_h.P_h + Q_h)\backslash A_h \parallel$$
$$(\langle i, \alpha, Q_t\rangle \cdot m_t \cdot \langle k, \beta, T\rangle \triangleright P_t)\backslash A_t)\backslash B$$

Then we also have $[\![Y + P, \langle\rangle, \mathbf{0}]\!] \rightarrow^i_\alpha [\![Y', \langle\rangle, P]\!] = [\![Y' + P, \langle\rangle, \mathbf{0}]\!]$. By applying the inductive hypothesis we also have $X \xrightarrow{\alpha[i]} X'$ and by rule K-PLUS-L, we get $X + P \xrightarrow{\alpha[i]} X' + P$ as desired. ■

Lemma 7 (Backward Completeness). *For any CCSK process X and RCCS process R, such that $R = [\![X, \langle\rangle, \mathbf{0}]\!]$, if $R \rightsquigarrow^i_\alpha R'$ in RCCS, then there exists a corresponding transition $X \overset{\alpha[i]}{\rightsquigarrow} X'$ in CCSK, with $R' = [\![X', \langle\rangle, \mathbf{0}]\!]$.*

Proof. By structural induction on X. Full proof can be found in Appendix **??**.

We now can state our main result:

Theorem 1 (Operational Correspondance). *For any CCS process P, $P \sim [\![P, \langle\rangle, \mathbf{0}]\!]$.*

Proof. We just need to show that the relation

$$\mathcal{R} = \{(X, [\![X, \langle\rangle, \mathbf{0}]\!]) \text{ with } X \text{ CCSK reachable } \}$$

is a strong back and forth bisimulation.

If X does a forward transition, $X \xrightarrow{\alpha[i]} Y$ by Lemma 4 we have also that $[\![X, \langle\rangle, \mathbf{0}]\!] \rightarrow^\alpha_i [\![Y, \langle\rangle, \mathbf{0}]\!]$, with $(Y, [\![Y, \langle\rangle, \mathbf{0}]\!]) \in \mathcal{R}$. If the transition is a backward one we apply the Lemma 5.

If $R = [\![X, \langle\rangle, \mathbf{0}]\!]$ does a forward transition, $R \rightarrow^i_\alpha S$ then by Lemma 6 we also have that $X \xrightarrow{\alpha[i]} Y$ with $R = [\![Y, \langle\rangle, \mathbf{0}]\!]$, and we have that $(Y, [\![Y, \langle\rangle, \mathbf{0}]\!]) \in \mathcal{R}$. If the transition is a backward one we apply the Lemma 7. ■

4 Encoding RCCS in CCSK

In this section we just give the encoding of RCCS into CCSK and discuss how it works, without showing its correctness.

The main difference between RCCS and CCSK is that in RCCS via structural congruence is possible to split a parallel composition of processes sharing the same memory into a parallel composition of different monitored processes. This allows the single monitored processes to continue independently their computation. In CCSK there is no need of splitting rule, as its reversibility is *static*.

Then it is the case that different RCCS processes may correspond to a single CCSK process. In order to better understand this main issues, let us consider the following RCCS process:

$$R = \langle j, \beta, P_1 \rangle \cdot \langle \uparrow \rangle \cdot \langle i, \alpha, Q \rangle \cdot \langle \rangle \triangleright P_1 \parallel \langle \uparrow \rangle \cdot \langle i, \alpha, Q \rangle \cdot \langle \rangle \triangleright \gamma.P_2$$

derived from the initial process $\langle\rangle \triangleright \alpha.(\beta[j].P_1 \parallel \gamma.P_2) + Q$. Now the corresponding CCSK process is the following one:

$$\alpha[i].(\beta[j].P_1 \parallel \gamma.P_2) + Q$$

So the encoding has to be able, while encoding monitored processes, to collect partially encoded processes sharing the same memory. In the example before, the encoding has to join together processes $\beta[j].P_1$ and $\gamma.P_2$ and put them in the context $\alpha[i].[\bullet] + Q$. It is clear that such encoding cannot be compositional as it has to reason on the whole process while reconstructing back the history of monitored processes up to a split $\langle \uparrow \rangle$, then somehow apply the structural law Split in order to marge partially encoded processes and then to continue the encoding of the obtained parallel composition under the common memory. This is why the encoding of a RCCS reachable process R is defined as $\delta(\llparenthesis R \rrparenthesis)$, where function $\llparenthesis \cdot \rrparenthesis$ is inductively defined as follows

$$\begin{array}{ll} \llparenthesis R \parallel S \rrparenthesis = \llparenthesis R \rrparenthesis \parallel \llparenthesis S \rrparenthesis & \llparenthesis \langle\rangle, X \rrparenthesis = X \\ \llparenthesis R \backslash A \rrparenthesis = \llparenthesis R \rrparenthesis \backslash A & \llparenthesis \langle i, \alpha, Q \rangle \cdot m, X \rrparenthesis = \llparenthesis m, \alpha[i].X + Q \rrparenthesis \\ \llparenthesis m \triangleright P \rrparenthesis = \llparenthesis m, P \rrparenthesis & \llparenthesis \langle \uparrow \rangle \cdot m, X \rrparenthesis = \wr \langle \uparrow \rangle \cdot m, X \wr \end{array}$$

As we can see, the encoding of a monitored process $\llparenthesis m, P \rrparenthesis$ proceed as long as in m there are events of the form $\langle i, \alpha, Q \rangle$ and *freezes* when it encounters a memory m on top of which there is a split event $\langle \uparrow \rangle \cdot m$, and the act of this freezing produces a partially encoded process of the form $\wr \langle \uparrow \rangle \cdot m, X \wr$.

Function $\delta(\cdot)$, which is in charge of fusing two partially encoded CCSK processes sharing the same memory, is defined as follows:

$$\begin{array}{ll} \delta\left(\prod \wr \langle \uparrow \rangle \cdot m_l, X_l \wr \parallel \prod \wr \langle \uparrow \rangle \cdot m_t, X_t \wr \parallel \prod \wr \langle \uparrow \rangle \cdot m_z, X_z \wr\right) & = \\ \quad \delta\left(\prod \llparenthesis m_l, X_l \parallel X_t \rrparenthesis \parallel \prod \wr \langle \uparrow \rangle \cdot m_z, X_z \wr\right) & \text{if } \forall l \in L\, \exists t \in T \text{ s.t } m_l = m_t \\ \delta(X) = X & \end{array}$$

Let us note that when the δ only stops when an entire CCSK process has been derived, otherwise it applies again the encoding $\llparenthesis \cdot \rrparenthesis$ on the fused processes.

The following example shows how the entire mechanism work:

$$
\begin{aligned}
R =& \langle i, \alpha, T\rangle \cdot \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle \triangleright P_1 \parallel \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle \triangleright P_2 \parallel \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle \triangleright P_3 \\
(\!|R|\!) =& \delta\,((\!|\langle i, \alpha, T\rangle \cdot \langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_1|\!) \parallel (\!|\langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_2|\!) \parallel (\!|\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_3|\!)) \\
=& \delta\,((\!|\langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, \alpha[i].P_1 + T|\!) \parallel (\!|\langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_2|\!) \parallel (\!|\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_3|\!)) \\
=& \delta\,(\wr\langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, \alpha[i].P_1 + T\wr \parallel \wr\langle\uparrow\rangle \cdot \langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_2\wr \parallel (\!|\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_3|\!)) \\
=& \delta\,((\!|\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, (\alpha[i].P_1 + T \parallel P_2)|\!) \parallel (\!|\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_3|\!)) \\
=& \delta\,(\wr\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, (\alpha[i].P_1 + T \parallel P_2)\wr \parallel \wr\langle\uparrow\rangle \cdot \langle j, \beta, \mathbf{0}\rangle, P_3\wr) \\
=& \delta\,(\wr\langle j, \beta, \mathbf{0}\rangle, (\alpha[i].P_1 + T \parallel P_2) \parallel P_3\wr) \\
=& \delta\,(\wr\langle\rangle, \beta[j].((\alpha[i].P_1 + T \parallel P_2) \parallel P_3)\wr) \\
=& \beta[j].((\alpha[i].P_1 + T \parallel P_2) \parallel P_3)
\end{aligned}
$$

5 Conclusions and future work

The first reversible variant of CCS, called RCCS, was introduced by Danos and Krivine [2]. In RCCS each process is monitored by a *memory*, that serves as stack of past actions. Memories are considered as unique process identifiers, and in order to preserve this uniqueness along a parallel composition, a structural law permits to obtain unique memories though a parallel composition. A general method for reversing process calculi, given in a particular SOS format, has been proposed by Phillips and Ulidowski in [11]. The main idea of this approach is the use of communication keys to uniquely identify communications, and to make *static* each operator of the calculus. By applying this method to CCS, CCSK is obtained. Since in CCKS the history is directly annotated in the process itself, there is no need of splitting history through a parallel composition. We call this kind of recording histories as *static* reversibility; while we call the one used by RCCS as *dynamic*, since each thread is endowed with its own history. In order to show that these two methods are similar, e.g. two reversible CCS derived by them are strongly bisimilar, we have provided and encoding from a CCSK process to possibly several RCCS monitored processes. Then we have showed that a CCSK term and its encoding in RCCS are strongly back and forth bisimilar. We then sketched a possible encoding from RCCS to CCSK and discussed the difficulties behind it, mostly due to the fact that multiple split monitored process may correspond to a single CCSK process. We leave as future work showing the correctness of this encoding. Once this will be proven, then we can state that the two calculi (and their underling semantics) are equivalent, and will allow us to bring to CCSK some results about causally consistency already proven for RCCS.

We leave as future work showing that back and forth bisimulation is a congruence. Moreover, another interesting result would be to show that the two calculi are fully abstract, e.g. that two bisimilar CCSK terms are translated into two bisimimilar RCCS terms.

References

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. R. Landauer. Irreversibility and heat generation in the computing process. *IBM J. Res. Dev.*, 5:183–191, 1961.
7. I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, and J. Stefani. Concurrent flexible reversibility. In *Programming Languages and Systems - 22nd European Symposium on Programming, ESOP 2013*, pages 370–390, 2013.
8. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980.
9. 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.
10. 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.
11. I. C. C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2):70–96, 2007.
12. D. Sangiorgi and D. Walker. *The Pi-Calculus - a theory of mobile processes*. Cambridge University Press, 2001.