Towards a truly concurrent semantics for reversible CCS

Hernán Melgratti, Claudio Antares Mezzina, G Michele Pinna

Towards a truly concurrent semantics for reversible CCS[⋆]

Hernán Melgratti[1], Claudio Antares Mezzina[2], and G. Michele Pinna[3]

[1] ICC - Universidad de Buenos Aires - Conicet, Argentina
[2] Dipartimento di Scienze Pure e Applicate, Università di Urbino, Italy
[3] Università di Cagliari, Italy

Abstract. Reversible CCS (RCCS) is a well-established, formal model for reversible communicating systems, which has been built on top of the classical Calculus of Communicating Systems (CCS). In its original formulation, each CCS process is equipped with a memory that records its performed actions, which is then used to reverse computations. More recently, abstract models for RCCS have been proposed in the literature, basically, by directly associating RCCS processes with (reversible versions of) event structures. In this paper we propose a detour: starting from one of the well-known encoding of CCS into Petri nets we apply a recently proposed approach to incorporate causally-consistent reversibility to Petri nets, obtaining as result the (reversible) net counterpart of every RCCS term.

Keywords: Petri Nets · Reversible CCS · Concurrency.

1 Introduction

CCS [17] is a foundational calculus for concurrent systems. Typically, systems are described as the parallel composition of *processes* (i.e., components) that communicate by sending and receiving messages over named channels. Processes are then defined in terms of communication actions performed over specific channels: we write a and $\overline{a}$ to respectively represent a receive and a send over the channel a. Basic actions can be composed through prefixing (i.e., sequencing) ($\_.\_$), choice ($\_ + \_$) and parallel ($\_ \parallel \_$) operators. The original formulation of the semantics of CCS adheres to the so called *interleaved* approach, which only accounts for the executions that arise from a single processor; hence, parallelism can be reduced to non-deterministic choices and prefixing. For instance, the CCS processes $a \parallel b$ and $a.b + b.a$ are deemed *equivalent*, i.e., it does not distinguish a process that can perform a and b concurrently from one that sequentialises their execution in

[⋆] Partially supported by the EU H2020 RISE programme under the Marie Skłodowska-Curie grant agreement 778233, by the French ANR project DCore ANR-18-CE25-0007 and by the Italian INdAM – GNCS 2020 project Sistemi Reversibili Concorrenti: dai Modelli ai Linguaggi, and by the UBACyT projects 20020170100544BA and 20020170100086BA.

any possible order (interleaving/schedule). Successive works have addressed the problem of equipping CCS with *true concurrent* semantics in the style of Petri nets and Event Structures. It has been shown that every CCS process can be associated with a Petri net that can mimic its computations. Different flavours of Petri nets have been used in the literature; for instance, *occurrence* nets [6], a variant of *Conditions/Events* nets [5], and *flow* nets [3]. The works in [20] and [3] have additionally shown that the computation of a CCS process can be represented by using event structures.

Many efforts were made in the last decades to endow computation models with reversible semantics [1,16] and, in particular, two different models have been proposed for CCS: RCCS [4,9] and CCSK [19]. Both of them incorporate a logging mechanism in the operational semantics of CCS that enables the undoing of computation steps. Moreover, it has been shown that they are isomorphic [10] since they only differ on the information that they log: while RCCS relies on some form of *memory/monitor*, CCSK uses *keys*. Previous approaches have also developed true concurrent semantics for reversible versions of CCS. For instance, it has been shown that CCSK can be associated with reversible bundle event structures [7,8]. However, we still lack a Petri net model for RCCS processes. We may exploit some recent results that connect reversible occurrence nets with reversible event structures [15,13,14] to indirectly recover a Petri net model from the reversible bundle event structures defined in [7]. However, we follow a different approach, which is somehow more *direct*:

1. we encode (finite) CCS processes into a generalization of occurrence nets, namely *unravel* nets, in the vein of Boudol and Castellani [3];
2. we show that *unravel* nets can be made *causally-consistent* reversible by applying the approach in [13];
3. we finally shows that the reversible unravel nets derived by our encoding are an interpratation of RCCS terms.

An interesting aspect of the proposed encoding is that it highlights that all the information needed for reversing an RCCS process is already *encoded* in the structure of the net corresponding to the original CCS process, i.e., RCCS memories are represented by the structure of the net. Concretely, if an RCCS process R is a derivative of some CCS process P, then the encoding of R is retrieved from the encoding of P, what changes is the position of the markings. Consider the CCS process $P = a.\mathsf{0}$ that executes a and then terminates. It can be encoded as the Petri net on the left in Figure 1 (the usage of the apparently redundant places in the postset of a will be made clearer in Section 3).

The reversible version of P is $R = \langle\rangle \triangleright a.\mathsf{0}$, where $\langle\rangle$ denotes an initially empty memory. According to RCCS semantics, R evolves to $R' = \langle *, a, \mathsf{0}\rangle \cdot \langle\rangle \triangleright \mathsf{0}$ by executing a. The memory $\langle *, a, \mathsf{0}\rangle \cdot \langle\rangle$ in R' indicates that it can go back to the initial process R by undoing a. Note that the net corresponding to P (on the left) has already all the information needed to reverse the action a; morally, a can be undone by firing it in the opposite direction (i.e., by consuming tokens from the postset and producing them in its preset), or equivalently, by performing a reversing transition $\underline{a}$ that does the job, as shown in the net drawn in the middle

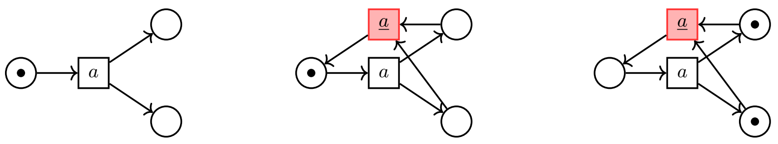


Fig. 1: Encoding of $R = \langle\rangle \triangleright a.0$

of Figure 1. It should be noted that the net on the right of Figure 1 corresponds to the derivative R'. Consequently, the encoding of a CCS term as a net already bears all the information needed for reversing it; which contrasts with the required memories of RCCS. This observation gives an almost straightforward true concurrent representation of RCCS processes.

Organization of the paper. After setting up some notation, we recall CCS and RCCS (Section 2). In Section 3 we summarise the basics of Petri nets, *unravel* nets and present their reversible counterpart. In Section 4, describe the encoding of CCS into unravel nets and introduce the mapping from RCCS terms into reversible unravel nets. In the final section we draw some conclusions and discuss future developments.

Preliminaries. We denote with $\mathbb{N}$ the set of natural numbers. Let A be a set, a *multiset* of A is a function $m : A \to \mathbb{N}$. The set of multisets of A is denoted by ∂A. We assume the usual operations on multisets such as union $+$ and difference $-$. We write $m \subseteq m'$ if $m(a) \leq m'(a)$ for all $a \in A$. For $m \in \partial A$, we denote with $[\![m]\!]$ the multiset defined as $[\![m]\!](a) = 1$ if $m(a) > 0$ and $[\![m]\!](a) = 0$ otherwise. When a multiset m of A is a set, *i.e.* $m = [\![m]\!]$, we write $a \in m$ to denote that $m(a) \neq 0$, and often confuse the multiset m with the set $\{a \in A \mid m(a) \neq 0\}$ or a subset $X \subseteq A$ with the multiset $X(a) = 1$ if $a \in A$ and $X(a) = 0$ otherwise. Furthermore we use the standard set operations like $\cap$, $\cup$ or $\setminus$. The multiset m such that $[\![m]\!] = \emptyset$ is denoted with abuse of notation as $\emptyset$.

2 CCS and reversible CCS

Let $\mathcal{A}$ be a set of actions $a, b, c, \ldots$, and $\overline{\mathcal{A}} = \{\overline{a} \mid a \in \mathcal{A}\}$ the set of their co-actions. We denote the set of all possible actions with $\texttt{Act} = \mathcal{A} \cup \overline{\mathcal{A}}$. We write α, β for the elements of $\texttt{Act}_\tau = \texttt{Act} \cup \{\tau\}$, where $\tau \notin \texttt{Act}$ stands for a *silent* action. The syntax of (finite) CCS is reported in Figure 2. A prefix (or action) is either an input a, an output $\overline{a}$ or the silent action τ. A term of the form $\sum_{i\in I} \alpha_i.P_i$ represents a process that (non-deterministically) starts by selecting and performing some action α_i and then continues as P_i. We write $\mathbf{0}$, the idle process, in lieu of $\sum_{i\in I} \alpha_i.P_i$ when $I = \emptyset$; similarly, $\alpha_i.P$ for a unitary sum in which I is the singleton $\{i\}$. The term $P \parallel Q$ represents the parallel composition of the processes P and Q. An action a can be restricted so to be visible only

$$\begin{array}{lll} \text{(Actions)} & \alpha & ::= a \mid \bar{a} \mid \tau \\ \text{(CCS Processes)} & P, Q & ::= \sum_{i\in I} \alpha_i.P_i \mid (P \parallel Q) \mid P\backslash a \end{array}$$

Fig. 2: CCS Syntax

$$\sum_{i\in I} \alpha_i.P \xrightarrow{\alpha_z} P_z \ \text{(ACT)} \qquad \frac{P \xrightarrow{\alpha} P'}{P \parallel Q \xrightarrow{\alpha} P' \parallel Q}\ \text{(PAR-L)} \qquad \frac{Q \xrightarrow{\alpha} Q'}{P \parallel Q \xrightarrow{\alpha} P \parallel Q'}\ \text{(PAR-R)}$$

$$\frac{P \xrightarrow{\alpha} P' \qquad Q \xrightarrow{\bar{\alpha}} Q'}{P \parallel Q \xrightarrow{\tau} P' \parallel Q'}\ \text{(SYN)} \qquad \frac{P \xrightarrow{\alpha} P' \qquad \alpha \notin \{a, \bar{a}\}}{P\backslash a \xrightarrow{\alpha} P'\backslash a}\ \text{(R-RES)}$$

Fig. 3: CCS semantics

inside process P, written $P\backslash a$. Restriction is the only binder in CCS: a occurs 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.

Definition 1 (CCS Semantics). *The operational semantics of CCS is defined as the LTS* $(\mathcal{P}, \mathtt{Act}_\tau, \to)$ *where the transition relation* $\to$ *is the smallest relation induced by the rules in Figure 3.*

Let us comment on rules of Figure 3. Rule ACT transforms an action α into a label. Rules PAR-L and PAR-R permit respectively the left and the right process of a parallel composition to propagate an action. Rule SYN regulates the syncrhonization allowing two process in parallel to handshake. Rule HIDE forbids a restricted action to be further propagated.

Reversible CCS. Reversible CCS (RCCS) [4,9] is a reversible variant of CCS. Processes in RCCS are equipped with a *memory*, in which a process keeps information about its past actions. The syntax of RCCS is in Figure 4, where CCS processes are defined as in the original formulation. A reversible process is either a *monitored* process $m \triangleright P$ with m a memory and P a CCS process, the parallel composition $R \parallel S$ of the reversible processes R and S, and the restriction $R\backslash a$ in which a is restricted to R. A *memory* is a (possible empty) stack of events that encodes the history of actions previously performed by a process; whose top-most element corresponds to the very last action performed by the monitored process. Memories can contain three different kinds of events[4]: *partial* synchronisations $\langle *, \alpha, Q\rangle$, *full* synchronisations $\langle m, \alpha, Q\rangle$, and memory

[4] In this paper we use the original *RCCS* semantics with partial synchronisation. Later versions, e.g. [9], use communication keys to univocally identify actions.

$$
\begin{array}{lll}
\text{(CCS Processes)} & P, Q ::= \sum_{i \in I} \alpha_i.P_i \mid (P \parallel Q) \mid P\backslash a \\
\text{(RCCS Processes)} & R, S ::= m \triangleright P \mid (R \parallel S) \mid R\backslash a \\
\text{(Memories)} & m \quad ::= \langle *, \alpha, Q\rangle \cdot m \mid \langle m, \alpha, Q\rangle \cdot m' \mid \langle 1\rangle \cdot m \mid \langle 2\rangle \cdot m \mid \langle\rangle
\end{array}
$$

Fig. 4: RCCS syntax

splits $\langle 1\rangle$ and $\langle 2\rangle$. The action α and the process Q in a synchronisation (either partial or full) record the selected action α of a choice and the discarded branches Q. The (technical) distinction between partial and full synchronisation will be made clear when describing the semantics or RCCS. Events $\langle 1\rangle$ and $\langle 2\rangle$ represent the split of a process in two parallel ones. The empty memory is represented by $\langle\rangle$. Let us note that in RCCS memories serves also as unique process identifiers.

As for CCS, the only binder in RCCS is restriction (at the level of both CCS and RCCS processes). We extend functions `n`, `fn` and `bn` to RCCS processes and memories accordingly. We define $\mathcal{M}$ the set of all the possible memories and $\mathcal{P}_R$ the set of all RCCS processes.

Definition 2 (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* $(\mathcal{P}_R, \mathcal{M} \times \mathtt{Act}_\tau, \rightarrow)$ *and a backward LTS* $(\mathcal{P}_R, \mathcal{M} \times \mathtt{Act}_\tau, \rightsquigarrow)$*. Transition relations* $\rightarrow$ *and* $\rightsquigarrow$ *are the smallest relations induced by the rules in Figure 5 (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 6. We define* $\rightleftharpoons\ = \rightarrow \cup \rightsquigarrow$*.*

Let us comment on the forward rules (Figure 5, left column). Rule R-ACT allows a monitored process to perform a forward action. The action goes along with the memory m of the process. Since at this point we do not know whether the process will synchronise or not with the context a partial synchronisation event of the form $\langle *, \alpha_z^z, \sum_{i \in I \backslash z} \alpha_i.P_i\rangle$ is put on top of the memory. The '$*$' will be replaced by a memory, say, m_1 if the process will eventually synchronise with a process monitored by m_1. Let us note that the discarded process Q is recorded in the memory. Moreover, along with the prefix we also store its position 'z' within the sum. This piece of information is redundant for RCCS, and indeed was not present in the original semantics. However, it will be of help when encoding a RCCS process into a net, and when proving the operational correspondence. We remark that this simple modification does not intact the original semantics of RCCS. Rules R-PAR-L and R-PAR-R propagate an action through a parallel composition. Rule R-SYN allows two parallel processes to synchronize. To do so, they have to match both the action α and then the two partial synchronisation of the two processes are updated to two full synchronisation through the operator '@'. Let R be a monitored process and m_1 and m_2 two memories, then $R_{m_2 @ m_1}$ stands for the process obtained from R by substituting all occurrences

$$m \triangleright \sum_{i\in I} \alpha_i.P_i \xrightarrow{m:\alpha_z} \langle *, \alpha_z^z, \sum_{i\in I\setminus\{z\}} \alpha_i.P_i\rangle \cdot m \triangleright P_z \ \ (\textsc{r-act})$$

$$\langle *, \alpha_z^z, \sum_{i\in I\setminus\{z\}} \alpha_i.P_i\rangle \cdot m \triangleright P_z \overset{m:\alpha_z}{\rightsquigarrow} m \triangleright \sum_{i\in I} \alpha_i.P_i \ \ (\textsc{r-act}^\bullet)$$

$$(\textsc{r-par-l})\ \frac{R \xrightarrow{m:\alpha} R'}{R \parallel S \xrightarrow{m:\alpha} R' \parallel S} \qquad \frac{R \overset{m:\alpha}{\rightsquigarrow} R'}{R \parallel S \overset{m:\alpha}{\rightsquigarrow} R' \parallel S}\ (\textsc{r-par-l}^\bullet)$$

$$(\textsc{r-par-r})\ \frac{S \xrightarrow{m:\alpha} S'}{R \parallel S \xrightarrow{m:\alpha} R \parallel S'} \qquad \frac{S \overset{m:\alpha}{\rightsquigarrow} S'}{R \parallel S \overset{m:\alpha}{\rightsquigarrow} R \parallel S'}\ (\textsc{r-par-r}^\bullet)$$

$$(\textsc{r-syn})\ \frac{R \xrightarrow{m_1:\alpha} R' \qquad S \xrightarrow{m_2:\bar{\alpha}} S'}{R \parallel S \xrightarrow{m1,m2:\tau} R'_{m_2@m_1} \parallel S'_{m_1@m_2}} \qquad \frac{R \overset{m_1:\alpha}{\rightsquigarrow} R' \qquad S \overset{m_2:\bar{\alpha}}{\rightsquigarrow} S'}{R \parallel S \overset{m1,m2:\tau}{\rightsquigarrow} R' \parallel S'}\ (\textsc{r-syn}^\bullet)$$

$$(\textsc{r-res})\ \frac{R \xrightarrow{m:\alpha} R' \qquad \alpha \notin \{a, \bar{a}\}}{R\setminus a \xrightarrow{m:\alpha} R'\setminus a} \qquad \frac{R \overset{m:\alpha}{\rightsquigarrow} R' \qquad \alpha \notin \{a, \bar{a}\}}{R\setminus a \overset{m:\alpha}{\rightsquigarrow} R'\setminus a}\ (\textsc{r-res}^\bullet)$$

$$(\textsc{r-equiv})\ \frac{R \equiv R' \quad R' \xrightarrow{m:\alpha} S' \quad S' \equiv S}{R \xrightarrow{m:\alpha} S} \qquad \frac{R \equiv R' \quad R' \overset{m:\alpha}{\rightsquigarrow} S' \quad S' \equiv S}{R \overset{m:\alpha}{\rightsquigarrow} S}\ (\textsc{r-equiv}^\bullet)$$

Fig. 5: RCCS semantics

of $\langle *, \alpha, Q\rangle \cdot m_1$ by $\langle m_2, \alpha, Q\rangle \cdot m_1$. 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 of Figure 6. Structural rule SPLIT allows a monitored process with a top-level parallel composition to split into a left and right branch, 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 $=_\alpha$.

Definition 3 (Initial Process and Coherent process). *A RCCS process of the form $\langle\rangle \triangleright P$ is called initial. Every process R derived from an initial process is called coherent process.*

3 Petri nets, Unravel Nets and Reversible Unravel Nets

A *Petri net* is a tuple $N = \langle S, T, F, \mathsf{m}\rangle$, where S is a set of *places* and T is a set of *transitions* (with $S \cap T = \emptyset$), $F \subseteq (S \times T) \cup (T \times S)$ is the *flow* relation, and $\mathsf{m} \in \partial S$ is called the *initial marking*.

Petri nets are depicted as usual: transitions are boxes, places are circles and the flow relation is depicted using directed arcs. The presence of tokens in places

$$
\begin{array}{lll}
(\textsc{split}) & m \triangleright (P \parallel Q) \equiv \langle 2\rangle \cdot m \triangleright P \parallel \langle 1\rangle \cdot m \triangleright Q & \\
(\textsc{res}) & m \triangleright P\backslash a \equiv (m \triangleright P)\backslash a \qquad \text{if } a \notin \mathtt{fn}(m) & \\
(\alpha) & R \equiv S \qquad \text{if } R =_\alpha S &
\end{array}
$$

Fig. 6: RCCS Structural laws

is signaled by a number of '$\bullet$' in it. The marking represents the *state* of the distributed and concurrent system modeled by the net, and it is distributed.

Given a net $N = \langle S, T, F, \mathsf{m}\rangle$ and $x \in S \cup T$, we define the following multisets: ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$. If x is a place then ${}^\bullet x$ and $x^\bullet$ are (multisets) of transitions; analogously, if $x \in T$ then ${}^\bullet x \in \partial S$ and $x^\bullet \in \partial S$. A transition $t \in T$ is enabled at a marking $m \in \partial S$, denoted by $m\,[t\rangle$, whenever ${}^\bullet t \subseteq m$. A transition t enabled at a marking m can *fire* and its firing produces the marking $m' = m - {}^\bullet t + t^\bullet$. The firing of t at a marking m is denoted by $m\,[t\rangle\, m'$. We assume that each transition t of a net N is such that ${}^\bullet t \neq \emptyset$, meaning that no transition may fire *spontaneously*. Given a generic marking m (not necessarily the initial one), the *firing sequence* (shortened as fs) of $N = \langle S, T, F, \mathsf{m}, \ell\rangle$ starting at m is defined as: (i) m is a firing sequence (of length 0), and (ii) if $m\,[t_1\rangle\, m_1 \cdots m_{n-1}\,[t_n\rangle\, m_n$ is a firing sequence and $m_n\,[t\rangle\, m'$, then also $m\,[t_1\rangle\, m_1 \cdots m_{n-1}\,[t_n\rangle\, m_n\,[t\rangle\, m'$ is a firing sequence. The set of firing sequences of a net N starting at a marking m is denoted by $\mathcal{R}^N_m$ and it is ranged over by σ. Given a fs $\sigma = m\,[t_1\rangle\, \sigma'\,[t_n\rangle\, m_n$, *start*$(\sigma)$ is the marking m, *lead*(σ) is the marking m_n and *tail*(σ) is the fs $\sigma'\,[t_n\rangle\, m_n$. Given a net $N = \langle S, T, F, \mathsf{m}, \ell\rangle$, a marking m is *reachable* iff there exists a fs $\sigma \in \mathcal{R}^N_{\mathsf{m}}$ such that *lead*(σ) is m. The set of reachable markings of N is $\mathcal{M}_N = \{\mathit{lead}(\sigma) \mid \sigma \in \mathcal{R}^N_{\mathsf{m}}\}$. Given a fs $\sigma = m\,[t_1\rangle\, m_1 \cdots m_{n-1}\,[t_n\rangle\, m'$, we write $X_\sigma = \sum_{i=1}^n t_i$ for the multiset of transitions associated to fs, which we call an *execution* of the net and we write $\mathbb{E}(N) = \{X_\sigma \in \partial T \mid \sigma \in \mathcal{R}^N_{\mathsf{m}}\}$ for the set of the executions of N. Observe that an execution simply says which transitions (and the relative number of occurrences of them) has been executed, not their (partial) ordering. Given a fs $\sigma = m\,[t_1\rangle\, m_1 \cdots m_{n-1}\,[t_n\rangle\, m_n \cdots$, with ρ_σ we denote the sequence $t_1 t_2 \cdots t_n \cdots$.

Definition 4. *A net $N = \langle S, T, F, \mathsf{m}\rangle$ is said to be* safe *if each marking $m \in \mathcal{M}_N$ is such that $m = [\![m]\!]$.*

The notion of subnet will be handy in the following. A subnet is obtained by restricting places and transitions, and correspondingly the flow relation and the initial marking.

Definition 5. *Let $N = \langle S, T, F, \mathsf{m}\rangle$ be a Petri net and let $T' \subseteq T$ be a subset of transitions and $S' = {}^\bullet T' \cup T'^\bullet$. Then, the subnet generated by T' $N|_{T'} = \langle S', T', F', \mathsf{m}'\rangle$, where F' is the restriction of F to S' and T', and m' is the multiset on S' obtained by* m *restricting to the places in S'.*

Unravel nets. To define *unravel nets* we need the notion of *causal net.* A safe Petri net $N = \langle S, T, F, \mathsf{m}\rangle$ is a *causal net* (CA for short) when $\forall s \in S.\ |{}^{\bullet}s| \leq 1$ and $|s^{\bullet}| \leq 1$, F^* is acyclic, $T \in \mathbb{E}(N)$, and $\forall s \in S\ \ {}^{\bullet}s = \emptyset \ \Rightarrow\ m(s) = 1$. Requiring that $T \in \mathbb{E}(N)$ implies that each transition can be executed and F^* acyclic means that dependencies among transitions are settled. A causal net has no isolated and unmarked places as $\forall s \in S\ \ {}^{\bullet}s = \emptyset \ \Rightarrow\ m(s) = 1$.

Definition 6. *An* unravel net *(UN for short) $N = \langle S, T, F, \mathsf{m}\rangle$ is a safe net such that for each execution $X \in \mathbb{E}(N)$ the subnet $N|_X$ is a causal net.*

Unravel nets describe the dependencies among the transitions in the executions of a concurrent and distributed device and are similar to *flow nets* [2,3]. Flow nets are safe nets where, for each possible firing sequence, each place can be marked just once. This requirement implies that the subnet obtained by the transitions executed in the firing sequence is a causal net and also that all the transitions t are such that ${}^{\bullet}t \neq \emptyset$.

In an UN conflicting pair of transitions t and t' are those that are never together in an execution, *i.e.* $\forall X \in \mathbb{E}(N).\ \{t, t'\} \not\subseteq X$. Given a place s in an unravel net, if ${}^{\bullet}s$ contains more than one transition, then the transitions in ${}^{\bullet}s$ are in conflict.

It is worthwhile to observe that the classical notion of *occurrence net* [18,20] is indeed a particular kind of UN namely one where the conflict relation is *inherited* along the transitive closure of the flow relation and it can can be inferred from the structure of the net itself. A further evidence that unravel nets generalize occurrence nets is implied also by the fact that flow nets generalize occurrence nets as well [2].

An unravel net $N = \langle S, T, F, \mathsf{m}\rangle$ is *complete* whenever $\forall t \in T\ \exists s_t \in S.\ {}^{\bullet}s_t = \{t\}\ \wedge\ s_t{}^{\bullet} = \emptyset$. Thus in a complete UN the execution of a transition t is signaled by the marked place s_t. Given an UN N, it can be turned easily into a complete one by adding for each transition the suitable place, without changing the executions of the net, thus we consider complete UNs only. Completeness comes handy when defining the reversible counterpart of an UN.

Reversible unravel nets. The definition of *reversible unravel net* follows the one of *reversible occurrence net* [13] and generalize reversible occurrence nets as unravel nets generalize occurrence nets.

Definition 7. *A* reversible unravel net *(rUN for short) is a quintuple $N = \langle S, T, U, F, \mathsf{m}\rangle$ such that*

1. *$U \subseteq T$ and $\forall u \in U.\ \exists!\ t \in T \setminus U$ such that ${}^{\bullet}u = t^{\bullet}$ and $u^{\bullet} = {}^{\bullet}t$,*
2. *$\forall t, t' \in T.\ {}^{\bullet}t = {}^{\bullet}t'\ \wedge\ t^{\bullet} = t'^{\bullet}\ \Rightarrow\ t = t'$,*
3. *$\bigcup_{t\in T}({}^{\bullet}t \cup t^{\bullet}) = S$, and*
4. *$N|_{T\setminus U}$ is a complete unravel net and $\langle S, T, F, \mathsf{m}\rangle$ is safe one.*

The transitions in U are the reversing ones; hence, we often say that a reversible unravel net N is *reversible with respect to U*. A reversing transition u is associated with a unique non-reversing transition t (condition 1) and its effects are intended to *undo* t. The second condition ensures that there is an injective mapping $h : U \to T$ which in turn implies that each reversible transition has exactly one reversing transition. The third requirement guarantees that there are no isolated conditions and the final one states that the subnet obtained forgetting all the reversing transitions is indeed an unravel net.

Along the lines of [13], we can prove that the set of reachable markings of a reversible unravel net is not influenced by performing a reversing transition. Let $N = \langle S, T, U, F, \mathsf{m}\rangle$ be an rUN. Then $\mathcal{M}_N = \mathcal{M}_{N|_{T\setminus U}}$. A consequence of this fact is that each marking can be reached by using just *forward events*. Let $N = \langle S, T, U, F, \mathsf{m}\rangle$ be an rUN and σ be an fs. Then, there exists an fs σ' such that $X_{\sigma'} \subseteq T \setminus U$ and $\mathit{lead}(\sigma) = \mathit{lead}(\sigma')$.

Given an unravel net and a subset of transitions to be reversed, it is straightforward to obtain a reversible unravel net.

Proposition 1. *Let $N = \langle S, T, F, \mathsf{m}\rangle$ be a complete unravel net and $U \subseteq T$ the set of transitions to be reversed. Define $\overleftarrow{N}^U = \langle S', T', U', F', \mathsf{m}'\rangle$ where $S = S'$, $U' = U \times \{\mathtt{r}\}$, $T' = (T \times \{\mathtt{f}\}) \cup U'$,*

$$F' = \{(s,(t,\mathtt{f})) \mid (s,t) \in F\} \cup \{((t,\mathtt{f}),s) \mid (t,s) \in F\} \cup \\ \{(s,(t,\mathtt{r})) \mid (t,s) \in F\} \cup \{((t,\mathtt{r}),s) \mid (s,t) \in F\}$$

and $\mathsf{m}' = \mathsf{m}$. Then $\overleftarrow{N}^U$ is a reversible unravel net.

The construction above simply adds as many events (transitions) as transitions to be reversed in U. The preset of each added event is the postset of the corresponding event to be reversed, and its postset is the preset of the event to be reversed. We write $\overleftarrow{N}$ instead of $\overleftarrow{N}^T$ when $N = \langle S, T, F, \mathsf{m}\rangle$, i.e., when every transition is reversible.

In Figure 7a we show a non-complete unravel net, whose complete version is in Figure 7b. The reversible unravel net obtained by reversing every transition is depicted in Figure 7c.

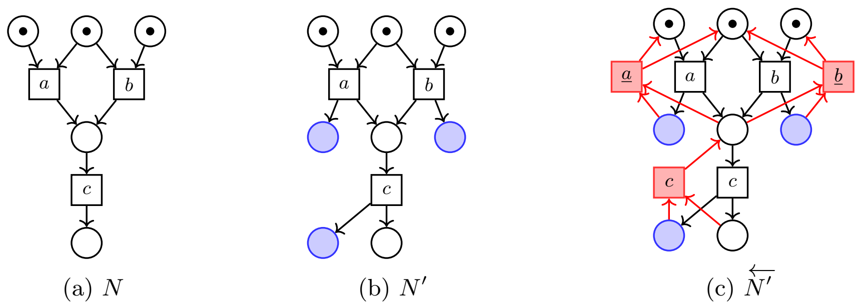
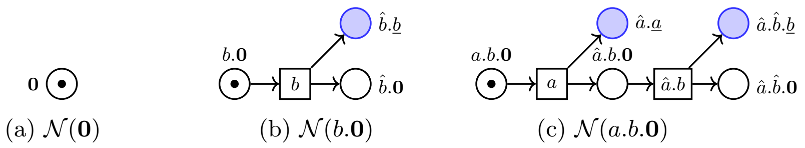


Fig. 7: An UN N, its complete version N' and the associated rUN $\overleftarrow{N'}$



$\mathbf{0}$
$b.\mathbf{0}$
b
$\hat{b}.\underline{b}$
$\hat{b}.\mathbf{0}$
$a.b.\mathbf{0}$
a
$\hat{a}.\underline{a}$
$\hat{a}.b.\mathbf{0}$
$\hat{a}.b$
$\hat{a}.\hat{b}.\underline{b}$
$\hat{a}.\hat{b}.\mathbf{0}$
(a) $\mathcal{N}(\mathbf{0})$
(b) $\mathcal{N}(b.\mathbf{0})$
(c) $\mathcal{N}(a.b.\mathbf{0})$

Fig. 8: Example of nets corresponding to CCS processes

4 CCS processes as unravel nets

We now recall the encoding of CCS terms into Petri nets due to Boudol and Castellani [3]. We just recall that originally the encoding was on proved terms instead of plain CCS. The difference between proved terms and CCS is that somehow in a proved term the labels carry the position of the process who did the action. Hence, we will use *decorated* version of labels. For instance, $\hat{a}.b$ denotes an event b which past was a. That is, if we want to indicate the event b of the term $a.b$ we will write $\hat{a}.b$. Analogously, labels carry also information about the syntactical structure of a term, actions corresponding to subterms of a choice and of a parallel composition are also decorated with an index i that indicates the subterm that performs the actions. An interesting aspect of this encoding is that these information is reflected in the name of the places and the transitions of the nets, which simplifies the formulation of the behavioural correspondence of a term and its associated net. We write $\ell(\_)$ for the function that removes decorations for a name, e.g., $\ell(\hat{a}.\hat{b}.c) = c$.

We now are in place to define and comment the encoding of a CCS term into a net. The encoding is inductively defined on the structure of the CCS process. For a CCS process P, its encoded net is $\mathcal{N}(P) = \langle S_P, T_P, F_P, \mathsf{m}_P\rangle$. The net corresponding to the inactive process $\mathbf{0}$, is just a net with just one marked place and with no transition, that is:

Definition 8. *The net* $\mathcal{N}(\mathbf{0}) = \langle\{\mathbf{0}\}, \emptyset, \emptyset, \{\mathbf{0}\}\rangle$ *is the net associated to* $\mathbf{0}$ *and it is called* zero.

To ease notation in the constructions we are going to present, we adopt following conventions: let $X \subseteq S \cup T$ be a set of places and transitions, we write $\hat{a}.X$ for the set $\{\hat{a}.x \mid x \in X\}$ containing the *decorated* versions of places and transitions in X. Analogously we lift this notation to relations: if R is a binary relation on $(S \cup T)$, then $\hat{\alpha}.R = \{(\hat{\alpha}.x, \hat{\alpha}.y) \mid (x, y) \in R\}$ is a binary relation on $(\alpha.S \cup \alpha.T)$.

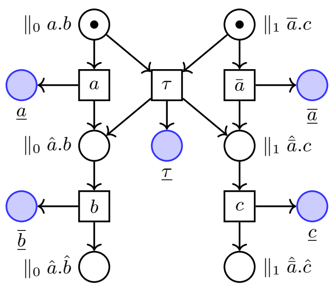
The net $\mathcal{N}(\alpha.P)$ corresponding to a process $\alpha.P$ extends $\mathcal{N}(P)$ with two extra places $\alpha.P$ and $\hat{a}.\underline{\alpha}$ and one transition α. The place $\alpha.P$ stands for the process that executes the prefix α and follows by P. The place $\hat{a}.\underline{\alpha}$ is not in the original encoding of [3]; we have add it to ensure that the obtained net is *complete*, which is essential for the definition of the reversible net. This will become clearer when commenting the encoding of the parallel composition. It should be noted that this addition does not interfere with the behaviour of the net, since all added places are final. Also a new transition, named α is created and added to the net, and the flow relation is updated accordingly. Figures 8a, 8b and 8c report the respectively the encoding of the inactive process, of the process $b.\mathbf{0}$ and $a.b.\mathbf{0}$. Moreover the aforementioned figures systematically show how the prefixing operator is rendered into Petri nets. As a matter of fact, the net $a.b.\mathbf{0}$ is built starting from the net corresponding to $b.\mathbf{0}$ by adding the prefix a. We note that also the label of transitions is affected by appending the label of the new prefix at the beginning. This is rendered in Figure 8c where the transition mimicking the action b is labeled as $\hat{a}.b$ indicating that an a was done before b. In what follows we will often omit such representation from figures.

Definition 9. *Let P a CCS process and $\mathcal{N}(P) = \langle S_P, T_P, F_P, \mathsf{m}_P\rangle$ be the associated net. Then $\mathcal{N}(\alpha.P)$ is the net $\langle S_{\alpha.P}, T_{\alpha.P}, F_{\alpha.P}, \mathsf{m}_{\alpha.P}\rangle$ where*

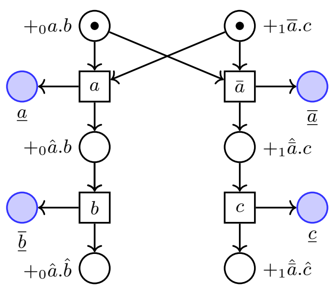
$$\begin{aligned}
S_{\alpha.P} &= \{\alpha.P, \hat{\alpha}.\underline{\alpha}\} \cup \hat{\alpha}.S_P \\
T_{\alpha.P} &= \{\alpha\} \ \cup \hat{\alpha}.T_p \\
F_{\alpha.P} &= \{(\alpha.P, \alpha), (\alpha, \hat{\alpha}.\underline{\alpha})\} \cup \{(\alpha, \hat{\alpha}.b) \mid b \in \mathsf{m}_{0P}\} \ \cup \hat{\alpha}.F_P \\
\mathsf{m}_{\alpha.P} &= \{\alpha.P\}
\end{aligned}$$

For a set X of transitions we write $\|_\mathsf{i} X$ for $\{\|_\mathsf{i} x \mid x \in X\}$, which straightforwardly lifts to relations.

The encoding of the parallel goes along the line of the prefixing one. Also in this case we have to decorate the places (and transitions) with the position of the term in the syntax tree. To this end, each branch of the parallel is decorated with $\|_i$ with i being the i-th position. Regarding the transitions, we have to add all the possible synchronisations among the processes in parallel. This is why, along with the transitions of the branches (properly decorated with $\|_i$) we have to add extra transitions to indicate the possible synchronisation. Naturally a synchronisation is possible when one label is the co-label of the other transition. Figure 9a shows the net corresponding to the process $a.b \parallel \overline{a}.c$. As we can see, the encoding builds upon the encoding of $a.b$ and $\overline{a}.c$, by (i) adding to all the places and transitions whether the branch is the left one or the right one and (ii) adding an extra transition and place for the only possible synchronisation. We add an extra place (in line with the prefixes) to mark the fact that a synchronisation has taken place. Let us note that the extra places $\underline{a}$, $\underline{\overline{a}}$ and $\underline{\tau}$ are used to understand whether the two prefixes have been executed singularly (e.g., no synchronisation) or they contributed to do a synchronisation. Suppose, for example, that the net had not such places, and suppose that we have two tokens in the places $\|_0\ \hat{a}.b$ and $\|_1\ \hat{\overline{a}}.b$. Now, how can we understand whether these two tokens are the result of the firing sequence a,$\overline{a}$ or they are the result of the τ transition? It is impossible, but by using the aforementioned extra-places, which are instrumental to tell if a single prefix has executed, we can distinguish the τ from the firing sequence a,$\overline{a}$ and then reverse accordingly.



(a) $\mathcal{N}(a.b \parallel \bar{a}.c)$



(b) $\mathcal{N}(a.b + \bar{a}.c)$

Fig. 9: Example of nets corresponding to CCS parallel and choice operator. We omit the trailing **0**

Definition 10. *Let* $\mathcal{N}(P_1)$ *and* $\mathcal{N}(P_2)$ *be the net associated to the processes* P_1 *and* P_2*. Then* $\mathcal{N}(P_1 \| P_2)$ *is the net* $\langle S_{P_1\|P_2}, T_{P_1\|P_2}, F_{P_1\|P_2}, \mathsf{m}_{P_1\|P_2}\rangle$ *where*

$$\begin{aligned}
S_{P_1\|P_2} &= \|_{0} S_{P_1} \cup \|_{1} S_{P_2} \cup \{s_{\{t,t'\}} \mid t \in T_{P_1} \wedge t' \in T_{P_2} \wedge \overline{\ell(t)} = \ell(t')\}\\
T_{P_1\|P_2} &= \|_{0} T_{P_1} \cup \|_{1} T_{P_2} \cup \{\{t,t'\} \mid t \in T_{P_1} \wedge t' \in T_{P_2} \wedge \overline{\ell(t)} = \ell(t')\}\\
F_{P_1\|P_2} &= \|_{0} F_{P_1} \cup \|_{1} F_{P_2} \cup \{(\{t,t'\}, s_{\{t,t'\}}) \mid t \in T_{P_1} \wedge t' \in T_{P_2} \wedge \overline{\ell(t)} = \ell(t')\}\\
&\quad \cup \{(\|_{\mathsf{i}} s, (t_1, t_2)) \mid (s, t_i) \in F_{P_i}\} \cup \{(\|_{\mathsf{i}} s, (t_1, t_2)) \mid (s, t_i) \in F_{P_i}\}\\
\mathsf{m}_{P_0\|P2} &= \|_{0} \mathsf{m}_{P_0} \cup \|_{1} \mathsf{m}_{P_1}
\end{aligned}$$

The encoding of choice operator is similar to the parallel one. The only difference is that we do not have to deal with possible synchronisations since the branches of a choice operator are mutually exclusive. Figure 9b reports the net corresponding to the process $a.b + \bar{a}.c$. As in the previous examples, the net is built upon the subnets representing $a.b$ and $\bar{a}.c$.

Definition 11. *Let* $\mathcal{N}(P_i)$ *be the net associated to the processes* P_i *for* $i \in I$*. Then* $+_{i\in I} P_i$ *is the net* $\langle S_{+_{i\in I}P_i}, T_{+_{i\in I}P_i}, F_{+_{i\in I}P_i}, \mathsf{m}_{+_{i\in I}P_i}\rangle$ *where:*

$$\begin{aligned}
S_{+_{i\in I}P_i} &= \cup_{i\in I} +_{\mathsf{i}} S_{P_i}\\
T_{+_{i\in I}P_i} &= \cup_{i\in I} +_{\mathsf{i}} T_{P_i}\\
F_{+_{i\in I}P_i} &= \{(+_{\mathsf{i}} x, +_{\mathsf{i}} y) \mid (x,y) \in F_{P_i}\} \cup \{(+_{\mathsf{j}} s, +_{\mathsf{i}} t) \mid s \in \mathsf{m}_{P_j} \wedge {}^{\bullet}t \in \mathsf{m}_{P_i} \wedge i \neq j\}\\
\mathsf{m}_{+_{i\in I}P_i} &= \cup_{i\in I} +_{\mathsf{i}} \mathsf{m}_{P_i}.
\end{aligned}$$

We write T^a for the set $\{t \in T \mid \mathsf{nm}(\ell(t)) = a\}$. The encoding of the hiding operator simply removes all transitions whose labels corresponds to actions over the restricted name.

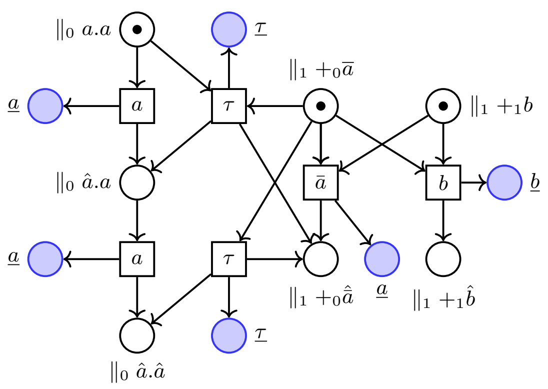


Fig. 10: A complex example: $\mathcal{N}(a.a \parallel \overline{a} + b)$

Definition 12. *Let P a CCS process and $\mathcal{N}(P) = \langle S_P, T_P, F_P, \mathsf{m}_P\rangle$ be the associated net. Then $\mathcal{N}(P \setminus a)$ is the net $\langle S_{P\setminus a}, T_{P\setminus a}, F_{P\setminus a}, \mathsf{m}_{P\setminus a}\rangle$ where*

$$
\begin{aligned}
&S_{P\setminus a} = \setminus_{\mathsf{a}} S_P\\
&T_{P\setminus a} = \setminus_{\mathsf{a}}(T_P \setminus T_a)\\
&F_{P\setminus a} = \{(\setminus_{\mathsf{a}} s, \setminus_{\mathsf{a}} t) \mid (s,t) \in F_P, t \notin T_P^a\} \cup \{(\setminus_{\mathsf{a}} t, \setminus_{\mathsf{a}} s) \mid (t,s) \in F_P, t \notin T_P^a\}\\
&\mathsf{m}_{P\setminus a} = \setminus_{\mathsf{a}} \mathsf{m}_P.
\end{aligned}
$$

Figure 10 shows a more complex example, the net corresponding to the process $a.a \parallel \overline{a} + b$. In this case, the process on the right of the parallel composition can synchronise with the one on the left one in two different occasions. This is why there are two different transitions representing the synchronisation. Also, since the right process of the parallel is a choice operator, it can happen that the right branch of it is executed, thus interdicting the synchronisation. Let us note that since the right branch of the parallel operator is a choice one, composed by two branches, the encoding labels these branches with '$\|_1 +_0$' and '$\|_1 +_1$' to indicate respectively the left and right branch of the choice operator, which is the right branch of a parallel operator. The following proposition is instrumental for our result.

Proposition 2. *The nets defined in Definition 8, Definition 9, Definition 10, Definition 11 and Definition 12 are* complete *unravel nets.*

We are now in place to define what is the net corresponding to a RCCS process. So far we have spoken about encoding CCS processes into nets. We remark that since RCCS is built upon CCS processes, also the encoding of RCCS is built upon the encoding of CCS. To do so, we first need the notion of ancestor, that is the initial process from which an RCCS process is derived. Let us note that since we are considering coherent RCCS processes (see Definition 3), an RCCS process has always an ancestor. The ancestor $\rho(R)$ of an RCCS process R can be calculated syntactically from R, since all the information about the past are

$$
\begin{aligned}
\rho(\langle\rangle \triangleright P) &= P \\
\rho(\langle \_, \alpha_z^{z}, \sum_{i \in I \setminus \{z\}} \alpha_i.P_i\rangle \cdot m \triangleright P) &= \rho(m \triangleright \sum_{i \in I} \alpha_i.P_i) \\
\rho(\langle i\rangle \cdot m \triangleright P) &= \langle i\rangle \cdot m \triangleright P \\
\rho(P_1 \| P_2) &= \rho(m \triangleright P_1' \| P_2') \textit{ where } \rho(P_i) = \langle i\rangle \cdot m \triangleright P_i' \ \wedge i \in \{1,2\} \\
\rho(P \setminus a) &= \rho(P) \setminus a
\end{aligned}
$$

Fig. 11: The ancestor of an RCCS process

stored into memories. The only point in which a process has to wait for its sibling is when a memory fork $\langle 1\rangle$ or $\langle 2\rangle$ is met.

Definition 13. *Given a coherent RCCS process R, its ancestor $\rho(R)$ is inductively defined as in Figure 11.*

The idea behind the ancestor process is that the encoding of an RCCS process and of its ancestor should give the same net, what changes is the position where the markings are placed. And such position is derived by the information stored into memories. We then define the *marking* function $\mu(\cdot)$ defined inductively as follows:

$$
\begin{array}{lll}
\mu(\langle\rangle \triangleright P) = \{P\} & \mu(P_0 \| P_1) = \|_{\mathsf{0}}\mu(P_0) \cup \|_{\mathsf{1}}\mu(P_1) & \mu(P \setminus a) = \setminus_{\mathsf{a}}\mu(P_1)
\end{array}
$$

$$
\mu(\langle m', \alpha^{i}, Q\rangle \cdot m \triangleright P) = \mu(m \triangleright +_{\mathsf{i}}\hat{\alpha}.P) \cup \{s_{t,t'} \mid t = m \triangleright +_{\mathsf{i}}\alpha \wedge t = m' \triangleright +_{\mathsf{i}}{}'\overline{\alpha}\}
$$

$$
\mu(\langle *, \alpha^{i}, Q\rangle \cdot m \triangleright P) = \mu(m \triangleright +_{\mathsf{i}}\hat{\alpha}.P) \cup \mu(m \triangleright +_{\mathsf{i}}\underline{\alpha})
$$

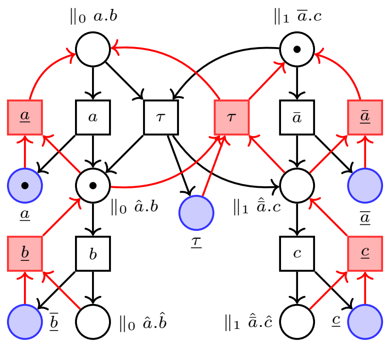
We are now in place to define what is the reversible net corresponding to an RCCS process:

Definition 14. *Let R be an RCCS term with $\rho(R) = P$. Then $\overleftarrow{\mathcal{N}(R)}$ is the net $\langle S, T, F, \mu(R)\rangle$ where $\mathcal{N}(P) = \langle S, T, F, \mathsf{m}\rangle$.*

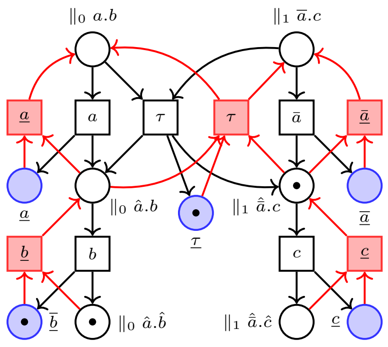
Proposition 3. *Let R be an RCCS term with $\rho(R) = P$. Then $\overleftarrow{\mathcal{N}(R)}$ is a reversible unravel net.*

In a few words Proposition 3 tells us that the reversible net corresponding to a coherent RCCS R is that one of its ancestor. The contribution of R to the construction of its net relies in the markings, that is the computational history contained in R is what determines the markings. This is rendered in Figures 12a and 12b where the two nets are the same since the two processes R_1 and R_2 shares the same origin. What changes is the where markings are placed, since R_1 and R_2 represents different computation from the origin process.

We can now state our main result in terms of bisimulation:



(a) $\overleftarrow{\mathcal{N}}(R_1)$



(b) $\overleftarrow{\mathcal{N}}(R_2)$

Fig. 12: Example of nets corresponding to RCCS process $R_1 = \langle *, a^1, \mathbf{0}\rangle \cdot \langle 1\rangle \cdot \langle\rangle \triangleright b \parallel \langle 2\rangle \cdot \langle\rangle \triangleright \overline{a}.c$ and $R_2 = \langle *, b^1, \mathbf{0}\rangle \cdot \langle m_2, a_1, \mathbf{0}\rangle \cdot \langle 1\rangle \cdot \langle\rangle \triangleright b \parallel \langle m_1, \overline{a}^1, \mathbf{0}\rangle \cdot \langle 2\rangle \cdot \langle\rangle \triangleright c$ with $m_i = \langle i\rangle \cdot \langle\rangle$

Theorem 1. *Let P be a finite CCS process, then $\langle\rangle \triangleright P \sim \overleftarrow{\mathcal{N}(P)}$.*

Proof sketch. It is sufficient to show that

$$\mathcal{R} = \{(R, \langle S, T, F, \mu(R)\rangle) \mid \rho(R) = P,\ \overleftarrow{\mathcal{N}(P)} = \langle S, T, F, m\rangle\}$$

is a bisimulation. Let us note that all the transitions in the generated net have a unique name, which is the path from the root to the term in the abstract syntax tree. There is a one to one correspondence between this path and the memory of a process which can mimic the same action/transition.

5 Conclusions and future works

On the line of previous research we have equipped a reversible process calculus with a non sequential semantics by using the classical encoding of process calculi into nets. What comes out from the encoding is that the machinery to reverse a process was already present in the encoding.

Our result relies on unravel nets, that are able to represent *or*-causality. The consequence is that the same event may have different pasts. Unravel nets are naturally related to *bundle* event structures [11,12], where the dependencies are represented using *bundles*, namely finite subsets of conflicting events, and the bundle relation is usually written as $X \mapsto e$. Starting from an unravel net $\langle S, T, F, \mathsf{m}\rangle$, and considering the transition $t \in T$, the bundles representing the dependencies are ${}^\bullet s \mapsto t$ for each $s \in {}^\bullet t$, and the conflict relation can be easily inferred by the semantic one definable on the unravel net. This result relies on the fact that in any unravel net, for each place s, the transitions in ${}^\bullet s$ are pairwise conflicting. The *reversible* bundle structures add to the bundle relation (defined also on the reversing events) a prevention relation, and the intuition behind this relation is the usual one: some events, possibly depending on the one to be reversed, are still present and they *prevent* that event to be reversed. The problem here is that in an unravel net, differently from occurrence nets, is not so easy to determine which transitions depend on the happening of a specific one, thus potentially preventing it from being reversed. An idea would be to consider all the transitions in $s^\bullet$ for each $s \in t^\bullet$, but it has to be carefully checked if this is enough. Thus, which is the proper "reversible bundle event structure" corresponding to the reversible unravel nets has to be answered, though it is likely that the conditions to be posed on the prevention relations will be similar to the ones considered in [7,8]. Once that also this step is done, we will have the full correspondence between reversible processes calculi and non sequential models.

References

1. B. Aman et al. Foundations of reversible computation. In *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, volume 12070 of *LNCS*, pages 1–40. Springer, 2020.
2. 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*, volume 469 of *LNCS*, pages 62–95. Springer, 1990.
3. G. Boudol and I. Castellani. Flow models of distributed computations: Three equivalent semantics for CCS. *Information and Computation*, 114(2):247–314, 1994.
4. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR 2004 - Concurrency Theory*, volume 3170 of *LNCS*, pages 292–307. Springer, 2004.
5. P. Degano, R. D. Nicola, and U. Montanari. A distributed operational semantics for CCS based on condition/event systems. *Acta Informatica*, 26(1/2):59–91, 1988.
6. U. Goltz. CCS and Petri Nets. In *Semantics of Systems of Concurrent Processes, LITP Spring School on Theoretical Computer Science, La Roche Posay, France, April 23-27, 1990, Proceedings*, volume 469 of *LNCS*, pages 334–357. Springer, 1990.
7. E. Graversen, I. Phillips, and N. Yoshida. Event structure semantics of (controlled) reversible CCS. In *Reversible Computation RC 2018*, volume 11106 of *LNCS*, pages 102–122. Springer, 2018.
8. E. Graversen, I. Phillips, and N. Yoshida. Event structure semantics of (controlled) reversible CCS. *Journal of Logical and Algebraic Methods in Programming*, 2021.
9. J. Krivine. A verification technique for reversible process algebra. In *Reversible Computation RC 2012. Revised Papers*, volume 7581 of *LNCS*. Springer, 2013.
10. I. Lanese, D. Medic, and C. A. Mezzina. Static versus dynamic reversibility in CCS. *Acta Informatica*, 58(1):1–34, 2021.
11. R. Langerak. Bundle event structures: a non-interleaving semantics for LOTOS. In *Formal Description Techniques, V, Proceedings of the IFIP TC6/WG6.1 FORTE 92*, volume C-10 of *IFIP Transactions*, pages 331–346. North-Holland, 1992.
12. R. Langerak, E. Brinksma, and J. Katoen. Causal ambiguity and partial orders in event structures. In *CONCUR '97: Concurrency Theory*, volume 1243 of *LNCS*, pages 317–331. Springer, 1997.

13. H. C. Melgratti, C. A. Mezzina, I. Phillips, G. M. Pinna, and I. Ulidowski. Reversible occurrence nets and causal reversible prime event structures. In *Reversible Computation RC 2020*, volume 12227 of *LNCS*, pages 35–53. Springer, 2020.
14. H. C. Melgratti, C. A. Mezzina, and G. M. Pinna. A distributed operational view of reversible prime event structures. In *Proceedings of the 36rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021*. ACM, 2021. *(to appear)*.
15. H. C. Melgratti, C. A. Mezzina, and I. Ulidowski. Reversing place transition nets. *Log. Methods Comput. Sci.*, 16(4), 2020.
16. C. A. Mezzina et al. Software and reversible systems: A survey of recent activities. In *Reversible Computation: Extending Horizons of Computing - Selected Results of the COST Action IC1405*, volume 12070 of *LNCS*, pages 41–59. Springer, 2020.
17. R. Milner. *A Calculus of Communicating Systems*, volume 92 of *LNCS*. Springer, 1980.
18. M. Nielsen, G. Plotkin, and G. Winskel. Petri Nets, Event Structures and Domains, Part 1. *Theoretical Computer Science*, 13:85–108, 1981.
19. I. C. C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebraic Methods Program.*, 73(1-2):70–96, 2007.
20. G. Winskel. Event Structures. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 325–392. Springer, 1986.