Reversible structures

Luca Cardelli[1] Cosimo Laneve[2]

[1] Microsoft Research, Cambridge
[2] Università di Bologna

Abstract. Reversible structures are computational units that may progress forward and backward and are primarily inspired by DNA circuits. We demonstrate a standardization theorem that bears a quadratic algorithm for reachability when units have unique id. We also discuss the encoding of a reversible concurrent calculus into reversible structures.

1 Introduction

In abstract computation systems, such as automata, lambda calculus, process algebra, etc., we usually model the forward progress of computations through a sequence of irreversible steps. But physical implementations of these steps are usually reversible: in physics and chemistry all operations are reversible, and only an appropriate injection of energy and entropy can move the computational system in a desired direction. Reversible computation has been shown to have very interesting physical properties [1]. Here we discuss the implementation of a simple computational calculus into a chemical system, reflecting the reversibility of the chemical system into the calculus instead of abstracting it.

In general, since a process calculus is not confluent and processes are non-deterministic, reversing a (forward) computation history means undoing the history not in a deterministic way but in a causally consistent fashion, where states that are reached during a backward computation are states that could have been reached during the computation history by just performing independent actions in a different order. In RCCS [8], Danos and Krivine achieve this with CCS without recursion by attaching a memory m to each process P, in the monitored process construct $m : P$. Memories in RCCS are stacks of information needed for processes to backtrack.

Chemical systems, however, are naturally reversible but have no such backtracking memory. Reversibility there means reversibility of configurations, while time of course keeps marching forward. The only way to make such a system exactly reversible is to remember the position and momentum of each molecule, which is precisely contrary to the well-mixing assumption of chemical soups, namely that the probability of collision between two molecules is independent of their position. Moreover, notions of causality and independence of events need to be adapted to reflect the fundamental fact that different molecules of the same chemical species are indistinguishable. Their interactions can cause effects, but not to the point of being able to identify the precise molecule that caused an effect.

We use DNA chemical systems as an example implementation, because DNA systems can be precisely and programmably orchestrated in a 'calculus-like' fashion. These systems can model CCS-style interaction and (massive) concurrency, and they naturally model structural congruence as well-mixed chemical solutions [2]. They can achieve irreversible computation, but they cannot avoid using reversible steps to do it (for example, for binary operators), and hence it is interesting to study their intrinsic reversibility. We provide a scheme for the molecular implementation of significant computational primitives – the *weak coherent reversible structures* (coarser, irreversible primitives and their DNA implementation were introduced in [4]).

We then study the formal interplay between causal dependency and (weak coherent) reversible structures where terms bear multiplicities, which are a way of expressing concentrations of chemical soups. Following Lévy [11], we define an equivalence on computations that abstracts away from the order of causally independent reductions – the *permutation equivalence*. Because of multiplicities this abstraction is more discriminating than usual. In particular, our permutation equivalence does not always exchange independent reductions. For example, two reductions that use a same signal cannot be exchanged because one cannot grasp whether the two reductions are competing on a same signal or are using two different occurrences of a same signal. Notwithstanding this inadequacy, permutation equivalence in (weak coherent) reversible structures yields a standardization theorem that allows one to remove converse reductions from computations. To our knowledge, the study of causality in a language with multiplicities is original (similar studies have been carried out in models such as Petri nets [9]).

We finally study *coherent* reversible structures where terms have unique ids – they have multiplicity one – and we draw a precise comparison with asynchronous RCCS [8]. (Coherence is not realizable in mass action systems, but may become realizable in the future if we learn how to control individual molecules.) The reachability problem in these structures has a computational complexity that is quadratic with respect to the size of the structure, a problem that is EXPSPACE-complete otherwise. As a byproduct, reachability in asynchronous RCCS is quadratic as well.

Due to space limitations, the technical details are omitted as well as the formal definitions of the encoding of asynchronous RCCS into reversible structures and the encoding of the latter ones into DNA circuits. We refer the interested reader to the full paper [**?**].

Related work. The studies about reversibility in calculi date back at least to the seventies when Bennett theorized reversible Turing machines that compute by dissipating less energy than irreversible ones [1]. Already Bennett's machines use histories for backtracking computations that are deterministic in that case.

More recently, areas such as bio-systems and quantum computing have stimulated foundational studies of reversible and distributed computations. For this reason, several reversible process calculi have been developed. In [8], Danos and Krivine define a reversible concurrent calculus – RCCS – and undertake a thorough algebraic study of reversibility. In RCCS the histories are recorded in memories that need a complex ad-hoc management. In particular, the congruence rule of distribution of memories in parallel contexts requires a global synchronization in the backward direction. Using a similar technique, [10] studies reversibility in the context of higher order concurrent languages and demonstrate that reversibility does not augment the expressive power of the language.

A general technique for reversing process calculi without using memories is proposed in [15]. As in our structures, in this technique, the structure of processes is not destroyed and the progress is noted by underlying the actions that have been performed (while we use the symbol ^). Unlike our structures, the technique, in order to tag the communicating processes, generates ids on-the-fly during the communications. As for RCCS, when the computation must be reverted in a distributed setting, this technique requires a global synchronization between parallel processes that have been spawned at the same time.

The authors of the above papers have all noticed that reversing a computation history means undoing the history not in a deterministic way but in a way that is consistent with causal dependency. This is discussed in some detail in [14].

2 The algebra of reversible structures

The syntax of reversible structures uses five disjoint infinite sets: *names* $\mathcal{N}$, ranged over by $a, b, c, \ldots$, *co-names* $\overline{\mathcal{N}}$, ranged over by $\overline{a}, \overline{b}, \overline{c}, \ldots$, and a countable set of *ids*, ranged over u, v, w, $\cdots$. Names and co-names are ranged over by $\alpha, \alpha', \ldots$ and $\overline{\overline{\alpha}} = \alpha$. The following notations for *sequences of actions* will be taken: $\mathtt{A}$, $\mathtt{B}$, $\cdots$ range over sequences of $\mathcal{N}$; $\overline{\mathtt{A}}$, $\overline{\mathtt{B}}$, $\cdots$ range over sequences of elements $u : \overline{a}$; $\mathtt{A}^{\perp}$, $\mathtt{B}^{\perp}$, $\cdots$ range over sequences of elements $u : a$.

Sequences of ids are ranged over by $\widetilde{u}$, $\widetilde{v}$, $\cdots$. The dots in sequences of ids are always omitted, that is $u.v.w$ is shortened into uvw, and the empty sequence is represented by ε. The length of a sequence is given by the function *length*$(\cdot)$.

The syntax of *reversible structures* includes *gates* g and *structures* $\mathtt{S}$, which are defined by the following grammar:

$$\begin{array}{rll} g ::= & \mathtt{A}^{\perp}.\hat{}\mathtt{B}.\overline{\mathtt{C}} & (\mathit{length}(\mathtt{A}^{\perp}.\mathtt{B}) > 0) \\ | & \mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}\overline{\mathtt{C}} & (\mathit{length}(\mathtt{A}^{\perp}) > 0) \end{array}$$

$$\mathtt{S} ::= \quad \mathbf{0} \quad | \quad u : \overline{a} \quad | \quad g \quad | \quad \mathtt{S} \mid \mathtt{S} \quad | \quad (\mathtt{new}\ a)\mathtt{S}$$

$\mathbf{0}$ is the void structure. A signal $u : \overline{a}$ is an elementary message with an id u; a gate is a term that accepts input signals and emits output signals, reversibly. The form $\mathtt{A}^{\perp}.\hat{}\mathtt{B}.\overline{\mathtt{C}}$ represents input-accepting gates, at least when not considering reverse reactions. $\mathtt{A}^{\perp}$ are the inputs that have been processed, $\mathtt{B}$ are the inputs still to be processed, and $\overline{\mathtt{C}}$ are the outputs to be emitted. The other form $\mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}\overline{\mathtt{C}}$ represents an output-producing gate (when not considering reverse reactions). The $\mathtt{A}^{\perp}$ is as before, $\overline{\mathtt{B}}$ are the outputs that have been emitted, and $\overline{\mathtt{C}}$ are the outputs still to be emitted. Since all the inputs in a gate have to be processed before the outputs are produced, we do not need to consider other forms. In both forms, the symbol ^ indicates the next operations (one forward and one backward) that the gate can perform.

For example, a *transducer gate* transforming a signal from a name a to b is defined by $\hat{}a.u : \overline{b}$. This gate may evolve into $v : a.\hat{}u : \overline{b}$ by inputting a signal $v : \overline{a}$. At this stage it may emit the signal $u : \overline{b}$, thus becoming $v : a.u : \overline{b}\hat{}$ or may backtrack to $\hat{}a.u : \overline{b}$ by releasing the signal $v : \overline{a}$ (see the following semantics). Another example is a *sink gate*, such as $\hat{}a.b$, that collects signals (and, in a stochastic model, may freeze them for a while). This gate may evolve into $u : a.\hat{}b$, and then may become $u : a.v : b\hat{}$.

A parallel composition " | " allows gates and signals to interact. We often abbreviate the parallel of $\mathtt{S}_i$ for $i \in I$, where I is a finite set, with $\prod_{i\in I} \mathtt{S}_i$. The new operator $(\mathtt{new}\ a)\mathtt{S}$ limits the scope of a to $\mathtt{S}$; the name a is said to be *bound* in $(\mathtt{new}\ a)\mathtt{S}$. This is the only binding operator in reversible structures. We write $(\mathtt{new}\ a_1, \cdots, a_n)\mathtt{S}$ for $(\mathtt{new}\ a_1)\cdots(\mathtt{new}\ a_n)\mathtt{S}$, $n \geq 0$, and sometimes we shorten $a_1, \cdots, a_n$ into $\widetilde{a}$. The *free names* in $\mathtt{S}$, denoted $\mathsf{fn}(\mathtt{S})$, are the names in $\mathtt{S}$ with a non-bound occurrence.

Structures we will never want to distinguish for any semantic reason are identified by a congruence. Let $\equiv$, called *structural congruence*, be the least congruence between structures containing alpha equivalence and satisfying the abelian monoid laws for parallel (associativity, commutativity and $\mathbf{0}$ as identity), and the scope laws

$$(\mathtt{new}\ a)\mathbf{0} \equiv \mathbf{0} \qquad (\mathtt{new}\ a)(\mathtt{new}\ a')\mathtt{S} \equiv (\mathtt{new}\ a')(\mathtt{new}\ a)\mathtt{S},$$

$$\mathtt{S} \mid (\mathtt{new}\ a)\mathtt{S}' \equiv (\mathtt{new}\ a)(\mathtt{S} \mid \mathtt{S}'), \quad \textit{if } a \notin \mathsf{fn}(\mathtt{S})$$

It is folklore that, for every structure $\mathtt{S}$, there is a structure $\mathtt{S}' = (\mathtt{new}\ \widetilde{a})(\prod_{i\in I} g_i \mid \prod_{j\in J} u_j : \overline{a_j})$. The structure $\mathtt{S}'$, which is unique up-to $\equiv$, is called the *normal form* of $\mathtt{S}$.

Definition 1. *The reduction relation of reversible structures is the least relation $\longrightarrow$ satisfying the axioms*

(*input capture*) $u : \overline{a} \mid \mathtt{A}^{\perp}.\hat{}a.\mathtt{B}.\overline{\mathtt{C}} \longrightarrow \mathtt{A}^{\perp}.u : a.\hat{}\mathtt{B}.\overline{\mathtt{C}}$,
(*input release*) $\mathtt{A}^{\perp}.u : a.\hat{}\mathtt{B}.\overline{\mathtt{C}} \longrightarrow u : \overline{a} \mid \mathtt{A}^{\perp}.\hat{}a.\mathtt{B}.\overline{\mathtt{C}}$,
(*output release*) $\mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}u : \overline{a}.\overline{\mathtt{C}} \longrightarrow u : \overline{a} \mid \mathtt{A}^{\perp}.\overline{\mathtt{B}}.u : \overline{a}.\hat{}\overline{\mathtt{C}}$,
(*output capture*) $u : \overline{a} \mid \mathtt{A}^{\perp}.\overline{\mathtt{B}}.u : \overline{a}.\hat{}\overline{\mathtt{C}} \longrightarrow \mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}u : \overline{a}.\overline{\mathtt{C}}$,

and closed under the rules

$$\frac{\mathtt{S} \longrightarrow \mathtt{S}'}{(\mathtt{new}\ a)\mathtt{S} \longrightarrow (\mathtt{new}\ a)\mathtt{S}'} \qquad \frac{\mathtt{S} \longrightarrow \mathtt{S}'}{\mathtt{S} \mid \mathtt{S}'' \longrightarrow \mathtt{S}' \mid \mathtt{S}''} \qquad \frac{\mathtt{S}_1 \equiv \mathtt{S}_1' \quad \mathtt{S}_1' \longrightarrow \mathtt{S}_2' \quad \mathtt{S}_2' \equiv \mathtt{S}_2}{\mathtt{S}_1 \longrightarrow \mathtt{S}_2}$$

As usual, sequences of reductions, called *computations*, are noted $\longrightarrow^*$. The reductions (*input capture*) and (*output release*) are called *forward reductions*, the reductions (*input release*) and (*output capture*) are called *backward reductions.*

We explain the axioms of reversible structures semantics by discussing the reductions of the transducer $\hat{}a.u : \overline{b}$ when exposed to signals $v : \overline{a}$ and $w : \overline{a}$. The transducer may behave either as $v : \overline{a} \mid w : \overline{a} \mid \texttt{^}a.u : \overline{b} \longrightarrow w : \overline{a} \mid v : a.\texttt{^}u : \overline{b}$ or as $v : \overline{a} \mid w : \overline{a} \mid \texttt{^}a.u : \overline{b} \longrightarrow v : \overline{a} \mid w : a.\texttt{^}u : \overline{b}$ according to the axiom (*input capture*) is instantiated either with the signal $v : \overline{a}$ or with $w : \overline{a}$ – in these cases $\mathtt{A}^{\perp}$ is empty. In turn, $w : \overline{a} \mid v : a.\texttt{^}u : \overline{b}$ may reduce with (*output release*) as $w : \overline{a} \mid v : a.\texttt{^}u : \overline{b} \longrightarrow w : \overline{a} \mid v : a.u : \overline{b}\texttt{^} \mid u : \overline{b}$ or may backtrack with (*input release*) as follows $w : \overline{a} \mid v : a.\texttt{^}u : \overline{b} \longrightarrow v : \overline{a} \mid w : \overline{a} \mid \texttt{^}a.u : \overline{b}$. This backtracking is always possible in our algebra. In fact, it is a direct consequence of the property that, for every axiom $\mathtt{S} \longrightarrow \mathtt{S}'$ of Definition 1, there is a "converse one" $\mathtt{S}' \longrightarrow \mathtt{S}$.

Proposition 1. *For any reduction* $\mathtt{S} \longrightarrow \mathtt{S}'$ *there exists a converse one* $\mathtt{S}' \longrightarrow \mathtt{S}$.

In the following sections we limit our analysis to a subclass of structures.

Definition 2. *A structure* $\mathtt{S}$ *is* weak coherent *whenever in its normal form* $(\mathtt{new}\ \widetilde{a})\mathtt{S}'$*, ids are uniquely associated to names and co-names. That is, if* $u : \alpha$ *and* $u : \alpha'$ *occur in* $\mathtt{S}'$ *then either* $\alpha = \alpha'$ *or* $\alpha = \overline{\alpha'}$*.*

For example, the structure $u : a.v : \overline{b}\texttt{^} \mid v : \overline{c}$ is not weak coherent because v is associated to two different co-names, while $u : a.v : \overline{b}\texttt{^} \mid v : \overline{b}$ is weak coherent. It is worth to remark that weak coherence is easily and compositionally enforced in reversible structures by appropriate use of new operators (see Section 3). Weak coherent structures are intended to include the "real-life" structures. In experimental (initial) solutions one has structures like

$$(\mathtt{new}\ \widetilde{a})(\prod_{i \in I} (\mathtt{new}\ u_i)(n_i \times (u_i : a_i)) \quad \mid \quad \prod_{j \in J} (\mathtt{new}\ \widetilde{v})(n_j \times g_j))$$

where n_h, with $h \in I \cup J$, are (usually huge) naturals and where $(\mathtt{new}\ \widetilde{u})\,\mathtt{S}$ is syntactic sugar for $\mathtt{S}\{\widetilde{u'}/\widetilde{u}\}$, where $\widetilde{u'}$ are fresh ids (we recall that news do not apply to ids; this simplifies the definition of labels in the next section).

Proposition 2. *If* $\mathtt{S}$ *is weak coherent and* $\mathtt{S} \longrightarrow \mathtt{S}'$ *then* $\mathtt{S}'$ *is weak coherent.*

It turns out that weak coherent reversible structures are a subcalculus of a language for DNA circuits – the DSD language [13]. We refer to [**?**] for details about this correspondence.

3 Weak coherence and causality

Because of reversibility, computations in our algebra may have a lot of forward and backward reductions that continuously do and undo stuff. For example, in the transducer of Section 2, the computation

$$v : \overline{a} \mid w : \overline{a} \mid \texttt{^}a.u : \overline{b} \longrightarrow w : \overline{a} \mid v : a.\texttt{^}u : \overline{b} \longrightarrow v : \overline{a} \mid w : \overline{a} \mid \texttt{^}a.u : \overline{b}$$

is actually equivalent to the empty one – the computation performing no reduction at all. Clearly the above two reductions may be repeated at will, still being equivalent to the empty computation. Therefore, it is meaningful to analyze whether a computation may be simplified, *i.e. shortened*, without altering its computational meaning. Let us discuss the problems through few examples. Consider the computation

$$\begin{aligned} v:\overline{a} \mid w:\overline{a} \mid \hat{}a.u:\overline{b} \mid \hat{}a.z:\overline{c} &\longrightarrow w:\overline{a} \mid v:a.\hat{}u:\overline{b} \mid \hat{}a.z:\overline{c} && (1)\\ &\longrightarrow v:a.\hat{}u:\overline{b} \mid w:a.\hat{}z:\overline{c} && (2)\\ &\longrightarrow v:\overline{a} \mid \hat{}a.u:\overline{b} \mid w:a.\hat{}z:\overline{c} && (3) \end{aligned}$$

The reductions (1) and (3) may be simplified because one is the reverse of the other. In order to achieve this simplification one may observe that reductions (1) and (2) involve disjoint structures – *there is no causal dependency between them* (similarly for (2) and (3)). When this happens, two consecutive reductions may be swapped, that is the second may be performed before the first. After the swapping of (1) and (2), the reduction (1) occurs immediately before (3) and they may be safely removed, thus obtaining the computation

$$v:\overline{a} \mid w:\overline{a} \mid \hat{}a.u:\overline{b} \mid \hat{}a.z:\overline{c} \longrightarrow v:\overline{a} \mid \hat{}a.u:\overline{b} \mid w:a.\hat{}z:\overline{c}$$

This equivalence between computations that swaps causally independent reductions is known in the literature as *permutation equivalence* [11, 3]. Following Lévy, permutation equivalence is defined in terms of labels that mark transitions and that allows one to retrieve reactants. The point of our structures is that labels are already available as ids of signals and gates. In particular, there is a label, noted μ, ν, $\cdots$, for every type of axiom of the reversible structures semantics: *input capture label*: $u \mid \widetilde{v}\hat{}\mathtt{A}.\widetilde{w}$, *input release label*: $\widetilde{v}u\hat{}\mathtt{A}.\widetilde{w}$, *output release label*: $\widetilde{v}.\widetilde{w}\hat{}u\widetilde{z}$, *output capture label*: $u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z}$. The labels of reductions are defined as follows. Let $id(\mathtt{A}^\perp) = \widetilde{v}$, $id(\overline{\mathtt{B}}) = \widetilde{w}$ and $id(\overline{\mathtt{C}}) = \widetilde{z}$; we write $\mu : (\mathtt{new}\ \widetilde{a})\mathtt{S} \longrightarrow (\mathtt{new}\ \widetilde{a})\mathtt{S}'$, where $\mathtt{S}$ and $\mathtt{S}'$ do not contain news, when the axiom used in the proof tree is

- (*input capture*) $u:\overline{a} \mid \mathtt{A}^\perp.\hat{}a.\mathtt{A}'.\overline{\mathtt{C}} \longrightarrow \mathtt{A}^\perp.u:a.\hat{}\mathtt{A}'.\overline{\mathtt{C}}$ and $\mu = u \mid \widetilde{v}\hat{}\mathtt{A}'.\widetilde{z}$,
- (*input release*) $\mathtt{A}^\perp.u:a.\hat{}\mathtt{A}'.\overline{\mathtt{C}} \longrightarrow u:\overline{a} \mid \mathtt{A}^\perp.\hat{}a.\mathtt{A}'.\overline{\mathtt{C}}$ and $\mu = \widetilde{v}u\hat{}\mathtt{A}'.\widetilde{z}$,
- (*output release*) $\mathtt{A}^\perp.\overline{\mathtt{B}}.\hat{}u:\overline{a}.\overline{\mathtt{C}} \longrightarrow u:\overline{a} \mid \mathtt{A}^\perp.\overline{\mathtt{B}}.u:\overline{a}.\hat{}\overline{\mathtt{C}}$ and $\mu = \widetilde{v}.\widetilde{w}\hat{}u\widetilde{z}$,
- (*output capture*) $u:\overline{a} \mid \mathtt{A}^\perp.\overline{\mathtt{B}}.u:\overline{a}.\hat{}\overline{\mathtt{C}} \longrightarrow \mathtt{A}^\perp.\overline{\mathtt{B}}.\hat{}u:\overline{a}.\overline{\mathtt{C}}$ and $\mu = u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z}$.

The definition of labels for reductions between structures in normal form and with the same bound names is necessary for addressing reactants in a unique way (up-to multiplicities). We notice that, in our case, this constraint does not bring any loss of generality.

If structures are not weak coherent then labels may fail to address reactants. For example, in $u:\overline{a} \mid u:\overline{b} \mid \hat{}a.v:\overline{c} \mid \hat{}a.v:\overline{d}$, the two reductions are both labelled $u \mid \hat{}a.v$ even if they address two different pairs of signal and gate. It is worth to notice a little notational discrepancy between labels and the terms they specify. Gates may perform two reductions: one forward and one backward. These reductions must be noted in different ways in order to separate them, even if they address the same gate. There is only one configuration that may cause ambiguity: when the symbol ^ is at the beginning of the output part of a gate. For example, the gate $u : a.\hat{}v : \overline{b}$ may reduce either into $u : a.v : \overline{b}\hat{}$ or into $\hat{}a.v : \overline{b}$ (the signals are omitted). In order to separate the two reductions, we label the former with $u\texttt{.}\hat{}v$ and the latter with $u\hat{}.v$ (the positions of "^" and "." are inverted). While these two labels are different, we agree that they specify the same gate.

Let $[\mu]^+$, read the *converse label of* μ, be the following labels (let a be the name associated to u):

$$[u \mid \widetilde{v}\hat{}a.\mathtt{A}.\widetilde{w}]^+ \stackrel{\text{def}}{=} \widetilde{v}u\hat{}\mathtt{A}.\widetilde{w} \qquad [\widetilde{v}u\hat{}\mathtt{A}.\widetilde{w}]^+ \stackrel{\text{def}}{=} u \mid \widetilde{v}\hat{}a.\mathtt{A}.\widetilde{w}$$

$$[\widetilde{v}.\widetilde{w}\hat{}u\widetilde{z}]^+ \stackrel{\text{def}}{=} u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z} \qquad [u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z}]^+ \stackrel{\text{def}}{=} \widetilde{v}.\widetilde{w}\hat{}u\widetilde{z}$$

$[\mu]^+$ is called "the converse label of μ" because the computations $\mu\,;[\mu]^+$ and $[\mu]^+\,;\mu$ do not change the initial structure (see Definition 3). In the following, with an abuse of notation, we also consider labels as the *sets* of terms they specify: input release labels and output release labels are singletons containing the gates they specify, input capture labels and output capture labels are sets of two elements, the signal and the gate they specify. Therefore we will be qualified in using set operations on labels, such as $\mu \cap \nu$.

Lemma 1. *Let* $\mu : \mathtt{S} \longrightarrow \mathtt{S}'$ *and* $\nu : \mathtt{S} \longrightarrow \mathtt{S}''$ *be such that* $\mu \cap \nu = \emptyset$*. Then there exists* $\mathtt{S}'''$ *such that* $\nu : \mathtt{S}' \longrightarrow \mathtt{S}'''$ *and* $\mu : \mathtt{S}'' \longrightarrow \mathtt{S}'''$*.*

Lemma 1 is known in the literature as "diamond lemma" because the two computations $\mu;\nu$ and $\nu;\mu$ have same initial and final structures – they are *coinitial* and *cofinal*. The condition $\mu \cap \nu = \emptyset$ means that reactants of the two reductions are disjoint, therefore reductions are not causally related and may be swapped. Contrary to other formalisms [11, 3, 8], in (weak coherent) reversible structures this condition does not completely catches reductions that may be performed concurrently. For example, in $u : \overline{a} \mid u : \overline{a} \mid \hat{}a.u : \overline{a} \mid w : c.u : \overline{a}.\hat{}v : \overline{b}$ we have the possibility of one input capture and one output capture of the same signal and Lemma 1 does not apply (even if there are two copies of the signal). Yet, the two computations $u \mid \hat{}a.u\ ;\ u \mid w.u.\hat{}v$ and $u \mid w.u.\hat{}v\ ;\ u \mid \hat{}a.u$ are coinitial and cofinal. The problem follows from the fact that labels do not convey details about multiplicities of signals and gates.

In the following, the computations $\mathtt{S} \longrightarrow^* \mathtt{S}'$ will be always noted by the sequence of labels of the corresponding reductions, separated by semicolons. For example, the computation $u : \overline{a} \mid \hat{}a.v : \overline{b} \longrightarrow^2 v : \overline{b} \mid u : a.v : \overline{b}\hat{}$ is noted $u \mid \hat{}a.v\ ;\ u\texttt{.}\hat{}v$.

Definition 3. Permutation equivalence*, written* $\sim$*, is the least equivalence relation between computations closed under composition and such that:*

$$\begin{array}{lll} \mu;[\mu]^+ \sim \varepsilon & & \\ \mu;\nu \sim \nu;\mu & \textit{if} & \mu \textit{ and } \nu \textit{ are coinitial and } \mu \cap \nu = \emptyset \end{array}$$

For example, the computation

$$\begin{aligned} u:\overline{a} \mid \hat{}a.v:\overline{b} \mid u:a.\hat{}v:\overline{b} &\longrightarrow u:a.\hat{}v:\overline{b} \mid u:a.\hat{}v:\overline{b} \\ &\longrightarrow u:a.\hat{}v:\overline{b} \mid u:a.v:\overline{b}\hat{} \mid v:\overline{b} \\ &\longrightarrow u:\overline{a} \mid \hat{}a.v:\overline{b} \mid u:a.v:\overline{b}\hat{} \mid v:\overline{b} \end{aligned}$$

that is represented by the sequence of labels $u|\hat{}a.v$; $u.\hat{}v$; $u\hat{}.v$ is permutation equivalent to $u.\hat{}v$.

Permutation equivalence as defined in Definition 3 is more discriminant than usual. For example, as already discussed, the computations $u \mid \hat{}a.u$; $u \mid w.u.\hat{}v$ and $u \mid w.u.\hat{}v$; $u \mid \hat{}a.u$ of the structure $u:\overline{a} \mid u:\overline{a} \mid \hat{}a.u:\overline{a} \mid w:c.u:\overline{a}.\hat{}v:\overline{b}$ are not equal even if the two reductions concern different terms. The reason for this discriminating power is due to multiplicities of gates and signals and the fact that labels do not distinguish different occurrences of a same term. Of course we might have defined more informative labels, in the style of [3], but this would have been a twist of mass action systems in the theory of reversible structures. In facts, in the latters, molecules have concentrations and two occurrences of a same molecule cannot be separated. Anyhow, reversible structures without multiplicities (where labels uniquely identify the terms) and their properties are studied in the next section.

Theorem 1 (Standardization theorem). *Let* S *be weak coherent and* μ_1 ; $\cdots$; μ_n *be a computation of* S *such that* μ_n *is the converse of* μ_1*. Then there is a shorter computation that is permutation equivalent to* μ_1 ; $\cdots$; μ_n*.*

The definitions of permutation equivalence and weak coherence imply that two permutation equivalent computations are cofinal. The converse direction is false, as discussed after Lemma 1 with the computations $u \mid \hat{}a.u$; $u \mid w.u.\hat{}v$ and $u \mid w.u.\hat{}v$; $u \mid \hat{}a.u$. This problem, that we will amend in the next section by refining weak coherence, is well-known in the theory of Petri nets [9].

4 Coherent structures

The mismatch between cofinality and permutation equivalence (of coinitial computations) may be eliminated by strengthening the notion of weak coherence. Following the remarks in Section 3, the refinement may be achieved by removing multiplicities from initial structures. We say that an occurrence of an id u is *positive* in a structure S if u occurs in a signal or in a gate $\mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}\overline{\mathtt{C}}$ or $\mathtt{A}^{\perp}.\hat{}\mathtt{B}.\overline{\mathtt{C}}$ in the $\mathtt{A}^{\perp}$ sequence or in the $\overline{\mathtt{C}}$ sequence. The occurrence of u is *negative* if it is in the $\overline{\mathtt{B}}$ sequence of a gate $\mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}\overline{\mathtt{C}}$. Let the *type of* g, written *type*(g), be the sequence of ids of co-names in g. For example *type*($v:a.\hat{}a.u:\overline{a}.w:\overline{c}$) $= uw$ (as usual, dots are omitted in sequences of ids). Let the *type of a label* be the type of the gate involved in the reduction.

Definition 4. *A weak coherent structure* S *is* coherent *whenever*

- *different gates in* S *have types with no id in common;*

– *ids occur at most twice: one occurrence is positive and the other is negative.*

Had we used the simpler constraint that ids occur linearly in structures, which may be reasonable for initial structures, a statement as Proposition 2 for coherent structures should have been definitely threatened. For example, the structure $u : \overline{a} \mid \hat{}a.v : \overline{b}.w : \overline{c} \mid \hat{}b.c$ reduces to $u : a.v : \overline{b}.w : \overline{c}\hat{} \mid v : b.\hat{}c \mid w : \overline{c}$ where the ids v and w occur twice – one occurrence is positive, the other is negative: the reader may verify that this last structure matches the constraints of Definition 4. It is possible to prove Proposition 2 for coherent structures.

Theorem 2. *Let $\mu_1; \mu_2; \cdots ; \mu_m$ and $\nu_1; \nu_2; \cdots ; \nu_n$ be two coinitial computations of a coherent structure. Then $\mu_1; \mu_2; \cdots ; \mu_m \sim \nu_1; \nu_2; \cdots ; \nu_n$ if and only if they terminate in the same structure, up-to structural congruence (they are cofinal).*

A coherent structure may be encoded by a 1-safe Petri net. For these nets the reachability problem is PSPACE-complete [7] and an exponential algorithm is presented in [7] (with respect to the number of gates in a structure). Below we give an algorithm whose computational complexity is quadratic with respect to the number of gates in the structure.

Let the *distance* between two gates g and g' *of the same type*, written $|g-g'|$, be the commutative operation defined as follows:

– if $g = \mathtt{A}^\perp.\mathtt{A_1}^\perp.\hat{}\mathtt{A}.\overline{\mathtt{B}}$ and $g' = \mathtt{A}^\perp.\mathtt{A_2}^\perp.\hat{}\mathtt{A}.\overline{\mathtt{B}}$, where the first id of $\mathtt{A_1}^\perp$ is different from the first id of $\mathtt{A_2}^\perp$, then

$$|g - g'| \stackrel{\text{def}}{=} \mathit{length}(\mathtt{A_1}^\perp) + \mathit{length}(\mathtt{A_2}^\perp)$$

– if $g = \mathtt{A}^\perp.\mathtt{A_1}^\perp.\hat{}\mathtt{A}.\overline{\mathtt{B}}$ and $g' = \mathtt{A}^\perp.\mathtt{A_2}^\perp.\mathtt{A}.\overline{\mathtt{B_1}}.\hat{}\overline{\mathtt{B_2}}$, where the first id of $\mathtt{A_1}^\perp$ is different from the first id of $\mathtt{A_2}^\perp$, then

$$|g - g'| \stackrel{\text{def}}{=} \mathit{length}(\mathtt{A_1}^\perp) + \mathit{length}(\mathtt{A_2}^\perp.\mathtt{A}^\perp.\overline{\mathtt{B_1}})$$

– if $g = \mathtt{A}^\perp.\mathtt{A_1}^\perp.\overline{\mathtt{B_1}}.\hat{}\overline{\mathtt{B_1'}}$ and $g' = \mathtt{A}^\perp.\mathtt{A_2}^\perp.\overline{\mathtt{B_2}}.\hat{}\overline{\mathtt{B_2'}}$, where the first id of $\mathtt{A_1}^\perp$ is different from the first id of $\mathtt{A_2}^\perp$, then

$$|g - g'| \stackrel{\text{def}}{=} \mathit{length}(\mathtt{A_1}^\perp.\overline{\mathtt{B_1}}) + \mathit{length}(\mathtt{A_2}^\perp.\overline{\mathtt{B_2}})$$

The distance between two structures $\mathtt{S}$ and $\mathtt{S'}$ containing gates of the same types, noted $|\mathtt{S} - \mathtt{S'}|$, is $\sum_{g \in \mathtt{S}, g' \in \mathtt{S'}, type(g) = type(g')} |g - g'|$.

Proposition 3. *Let $\mathtt{S}$ be a coherent structure and let $\mathtt{S} \longrightarrow \mathtt{S'} \longrightarrow^* \mathtt{S''}$ be a minimal computation (according to Theorem 1). Then $|\mathtt{S} - \mathtt{S''}| > |\mathtt{S'} - \mathtt{S''}|$.*

The algorithm takes two coherent structures $\mathtt{S}$ and $\mathtt{S'}$ such that, for every gate in $\mathtt{S}$ there is a corresponding one in $\mathtt{S'}$ with the same type, and conversely.

1. If $\mathtt{S} \equiv \mathtt{S'}$ then the algorithm terminates with success;
2. otherwise, a gate g in $\mathtt{S}$ is chosen with non-null distance from the corresponding one g' in $\mathtt{S'}$ and such that it may be reduced in $\mathtt{S}$ by decreasing its distance from g'. Let $\mathtt{S} \longrightarrow \mathtt{S''}$ be such reduction (by construction $|\mathtt{S}-\mathtt{S'}| > |\mathtt{S''}-\mathtt{S'}|$).

– if no such reduction is possible the algorithm terminates with failure;
– otherwise the algorithm returns to 1, replacing `S` with `S′`.

The data structures of the algorithm are two arrays. The first one stores the gates and is addressed using the first id of their type (by coherence, the first ids are sufficient to discriminate gates). The second array stores signals. The elements are accessed through the co-name of the signal. Every element is a boolean array that is accessed through the id and containing `true` or `false` according to the corresponding signal is present or absent, respectively. Let n be the number of gates in `S` and let k be the maximal length of a gate in `S`. The step 2 of the algorithm may require (i) a complete visit of the array of gates, that costs n, and, for each element, (ii) a gate analysis for determining the distance and the possible reduction that costs k. Since in the worst case, gates may be at distance $2k$, the algorithm may iterate $2k \times n$ times. Then its computational complexity is $O(2k^2 \times n^2)$. It is worth to remark that the computational complexity of the reachability problem in (weak coherent) reversible structures reduces to the reachability marking problem in bounded place-transition Petri nets, which is EXPSPACE complete [12, 6] and we are not aware of any better algorithm for not coherent structures.

5 The encoding of asynchronous RCCS

Coherent structures can encode a process calculus with a reversible transition relation: the *asynchronous* RCCS [8]. This allows one to establish properties of asynchronous RCCS using those of coherent structures, such as Theorem 2, which has been proved for RCCS in [8], or the above algorithm of reachability, which is original. For lack of space we give a quick overview of asynchronous RCCS and illustrate the encoding by discussing an example.

The syntax of asynchronous RCCS uses an infinite set of *names*, ranged over by $a, b, c, \ldots$, and a disjoint set of *co-names*. Names and co-names are ranged over by $\alpha, \beta, \ldots$ and are generically called *actions*. *Processes* P, *memories* m, and *run-time processes* R are defined by the following grammar:

$$\begin{array}{rcl} P & ::= & \mathbf{0} \quad | \quad \sum_{i\in I} \alpha_i.P_i \quad | \quad \prod_{i\in I} P_i \quad | \quad (\texttt{new}\ a)P \\ m & ::= & \langle\ \rangle \quad | \quad \langle i \rangle_n \bullet m \quad | \quad \langle m, \alpha, Q\rangle \bullet m \\ R & ::= & m \triangleright P \quad | \quad R \mid R \quad | \quad (\texttt{new}\ a)R \end{array}$$

The term $\mathbf{0}$ defines the terminated process; $\sum_{i\in I} \alpha_i.P_i$ defines a process that may perform one action α_i and continues as P_i; $\prod_{i\in I} P_i$ defines the parallel composition of processes P_i; finally the term $(\texttt{new}\ a)P$ defines a name with scope P. Memories are used to record the discarded alternatives in choices and the partners of synchronizations. Processes meet the following well-formed conditions: (i) continuations of co-names are empty; (ii) in $\prod_{i\in I} P_i$ the processes P_i are guarded choices.

The semantics of asynchronous RCCS is defined by a reduction relation $\longrightarrow$ that is the least relation on run-time processes satisfying the axioms:

– $m \triangleright (a.P + Q) \;|\; m' \triangleright (\overline{a} + R) \longrightarrow \langle m', a, Q\rangle \bullet m \triangleright P \;|\; \langle m, \overline{a}, R\rangle \bullet m' \triangleright \mathbf{0}$,
– $\langle m', a, Q\rangle \bullet m \triangleright P \;|\; \langle m, \overline{a}, R\rangle \bullet m' \triangleright \mathbf{0} \longrightarrow m \triangleright (a.P + Q) \;|\; m' \triangleright (\overline{a} + R)$,

and closed under the contextual rules for parallel, new and structural congruence (that, in addition to the standard rules, has also the rule $m \triangleright (\prod_{i\in 1..n} P_i) \equiv \prod_{i\in 1..n} \langle i\rangle_n \bullet m \triangleright P_i)$.

To illustrate the encoding in coherent reversible structures [**?**], consider the process $a.P + \overline{a}$ that may progress either as P by inputting a or terminate by outputting $\overline{a}$, according to the external environment offers an output or an input on a, respectively. The structure encoding this process is

$$(\texttt{new}\ c')((\hat{\ }c.a.u : \overline{c'} \;\;|\;\; [\![P]\!]_{c'}) \;\;|\;\; \hat{\ }c.v : \overline{a})$$

where $[\![P]\!]_{c'}$ is a structure implementing P. We assume that the environment may emit at most one signal with co-name $\overline{c}$. When such a signal arrives, one of the gates $\hat{\ }c.a.u : \overline{c'}$ and $\hat{\ }c.v : \overline{a}$ will react, let it be the second. Then the structure becomes $(\texttt{new}\ c')((\hat{\ }c.a.u : \overline{c'} \;\;|\;\; [\![P]\!]_{c'}) \;\;|\;\; u' : c.\hat{\ }v : \overline{a})$ that emits a signal $v : \overline{a}$. It is crucial for the correctness of the encoding that $v : \overline{a}$ cannot interact with any other branch of the choice, *i.e.* with the gate $\hat{\ }c.a.u : \overline{c'}$. At this stage, it is possible that the context offers a signal $v' : \overline{a}$ rather than accepting signals $v : \overline{a}$. That is, the local choice of the process does not matches the choice of the context. Reversibility plays a crucial role at this point. In fact, the above reductions are reverted; the signal $u' : \overline{c}$ is re-emitted, and the left branch of the above choice is chosen, thus obtaining the structure

$$(\texttt{new}\ c')((u' : c.\hat{\ }a.u : \overline{c'} \;\;|\;\; [\![P]\!]_{c'}) \;\;|\;\; \hat{\ }c.v : \overline{a})$$

that may accept the signal $v' : \overline{a}$. Notice that RCCS memories are implemented by inactive processes that are in parallel with the active ones. No ad-hoc memory management operation is used.

6 Conclusions

We have developed a reversible concurrent calculus that is amenable to biological implementations in terms of DNA circuits and is expressive enough to encode a reversible process calculus such as asynchronous RCCS.

This study can be extended in several directions. One direction is suggested to the theory of concurrency by biology. The encoding of RCCS is given in terms of coherent structures. For this reason asynchronous RCCS bears Theorem 2 (that has been already proved for RCCS in [8]), and an efficient algorithm of reachability. However coherence – a solution must contain exactly one molecule of every species – is very hard to achieve in nature, even if it will be simpler in the future. So, biology prompts a thorough study of reversible concurrent calculi where processes have multiplicities and the causal dependencies between copies may be exchanged. Section 3 is a preliminary study of this matter.

Another direction is about implementations. In this paper we have discussed the implementation of a concurrent language in biology. Since it is possible to extend reversible structures with irreversible operators, the resulting language may be used to model standard irreversible operators of programming languages in the chemistry.

Our study about reachability has been inspired by biology and retains an easy solution in reversible structures because of their simplicity. Studying other biological relevant problems, such as absence of molecules/processes, persistence of materials, etc., and designing efficient algorithms are other directions that need to be investigated in reversible structures and may bear simple solutions in this model.

References

1. C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, 1973.
2. G. Berry and G. Boudol. The chemical abstract machine. In *Proceedings of POPL'90*, pages 81–94. ACM, 1990.
3. G. Boudol and I. Castellani. Permutation of transitions: An event structure semantics for ccs and sccs. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, volume 354 of *Lecture Notes in Computer Science*, pages 411–427. Springer, 1989.
4. L. Cardelli. Strand algebras for dna computing. In *DNA 2009*, volume 5877 of *Lecture Notes in Computer Science*, pages 12–24, 2009.
5. L. Cardelli. Two-domain dna strand displacement. In *Developments in Computational Models (DCM 2010)*, volume 25 of *EPTCS*, pages 33–47, 2010.
6. E. Cardoza, R. J. Lipton, and A. R. Meyer. Exponential space complete problems for petri nets and commutative semigroups: Preliminary report. In *Eighth Annual ACM Symposium on Theory of Computing*, pages 50–54. ACM, 1976.
7. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. In *Foundations of Software Technology and Theoretical Computer Science*, volume 761 of *Lecture Notes in Computer Science*, pages 326–337. Springer, 1993.
8. V. Danos and J. Krivine. Reversible communicating systems. In *CONCUR 2004*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307, 2004.
9. P. Degano, J. Meseguer, and U. Montanari. Axiomatizing net computations and processes. In *LICS'89*, pages 175–185. IEEE Computer Society, 1989.
10. I. Lanese, C. A. Mezzina, and J.-B. Stefani. Reversing higher-order pi. In *Proceedings of CONCUR 2010*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010.
11. J.-J. Lévy. An algebraic interpretation of the *lambda beta* k-calculus; and an application of a labelled *lambda* -calculus. *Theor. Comput. Sci.*, 2(1):97–114, 1976.
12. E. W. Mayr and A. R. Meyer. The complexity of the word problems for commutative semigroups and polynomial ideals. *Adv. in Math.*, 46(3):305–329, 1982.
13. A. Phillips and L. Cardelli. A programming language for composable dna circuits. *Journal of the Royal Society Interface*, 6(S4), 2009.
14. I. Phillips and I. Ulidowski. Reversibility and models for concurrency. In *Proceedings of SOS 2007*, volume 192 of *ENTCS*, pages 93–108, 2007.
15. I. Phillips and I. Ulidowski. Reversing algebraic process calculi. *J. Log. Algebr. Program.*, 73(1-2):70–96, 2007.