Reversible structures

Luca Cardelli
Microsoft Research, Cambridge

Cosimo Laneve
Università di Bologna

November, 2010

Abstract

Reversible structures are computational units that may progress forward and backward. We study weak coherent structures that are primarily inspired by DNA circuits and may be compiled in these systems and demonstrate a standardization theorem. When units have unique id, the standardization theorem may be strengthened in a form that bears a quadratic algorithm for reachability, a problem that is EXPSPACE-complete for generic structures. We then define a compilation of a concurrent calculus – the asynchronous RCCS – to DNA *via* reversible structures, thus yielding a fine-grain implementation of memories of the past into chemistry.

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 nondeterministic, 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.

We finally discuss the integration of irreversible operators in our model.

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 $\hat{}$). 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 *id*s, 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:

- sequences of $\mathcal{N}$ are ranged over by $\mathtt{A}$, $\mathtt{B}$, $\cdots$;
- sequences of elements $u : \overline{a}$ are ranged over by $\overline{\mathtt{A}}$, $\overline{\mathtt{B}}$, $\cdots$;
- sequences of elements $u : a$ are ranged over by $\mathtt{A}^{\perp}$, $\mathtt{B}^{\perp}$, $\cdots$;

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 $\mathit{length}(\cdot)$.

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

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

$$
\begin{array}{rcll}
\mathtt{S} & ::= & & \\
 & & \mathbf{0} & \text{(null)} \\
 & | & u : \overline{a} & \text{(signal)} \\
 & | & g & \text{(gate)} \\
 & | & \mathtt{S} \mid \mathtt{S} & \text{(parallel)} \\
 & | & (\mathtt{new}\ a)\mathtt{S} & \text{(new)}
\end{array}
$$

$\mathbf{0}$ 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 $\hat{}$ 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 $\mathtt{fn}(\mathtt{S})$, 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

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

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

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 $\mathtt{S}$.

The semantics of reversible structures is defined operationally by means of a reduction relation.

Definition 2.1. *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 $v : \overline{a} \mid w : \overline{a} \mid \hat{}a.u : \overline{b} \longrightarrow w : \overline{a} \mid v : a.\hat{}u : \overline{b}$ or as $v : \overline{a} \mid w : \overline{a} \mid \hat{}a.u : \overline{b} \longrightarrow v : \overline{a} \mid w : a.\hat{}u : \overline{b}$ 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, $w : \overline{a} \mid v : a.\hat{}u : \overline{b}$ may reduce with (*output release*) as $w : \overline{a} \mid v : a.\hat{}u : \overline{b} \longrightarrow w : \overline{a} \mid v : a.u : \overline{b}\hat{} \mid u : \overline{b}$ or may backtrack with (*input release*) as follows $w : \overline{a} \mid v : a.\hat{}u : \overline{b} \longrightarrow v : \overline{a} \mid w : \overline{a} \mid \hat{}a.u : \overline{b}$. 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 2.1, there is a "converse one" $\mathtt{S}' \longrightarrow \mathtt{S}$.

Proposition 2.2. *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 2.3. *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 $u : a.v : \overline{b}\hat{} \mid v : \overline{c}$ is not weak coherent because v is associated to two different co-names, while $u : a.v : \overline{b}\hat{} \mid v : \overline{b}$ 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 2.4. *If* $\mathtt{S}$ *is weak coherent and* $\mathtt{S} \longrightarrow \mathtt{S}'$ *then* $\mathtt{S}'$ *is weak coherent.*

Reversible structures is actually a subset of a formalism for designing DNA circuits – the DSD language [13]. In this formalism it is possible to define DNA strands and gates and study the biological mechanisms for binding and unbinding of strands. We conclude this section by defining the encoding of the reversible structures into the DSD language. This part may be safely skipped by uninterested readers.

The syntax of DSD uses *domains* $\mathtt{a}$, $\mathtt{u}$, $\mathtt{t}$, $\cdots$, and *toehold domains* $\mathtt{a^{\sim}}$, $\mathtt{u^{\sim}}$, $\mathtt{t^{\sim}}$, $\cdots$. DSD terms $\mathtt{D}$ are similar to structures, except that gates and structures are replaced by strands and DNA gates. The strands and gates we use in the DSD encoding are in particular:

– *strands* are `<u t~ b>` or `<a t~ >` or `<t~ u>`;

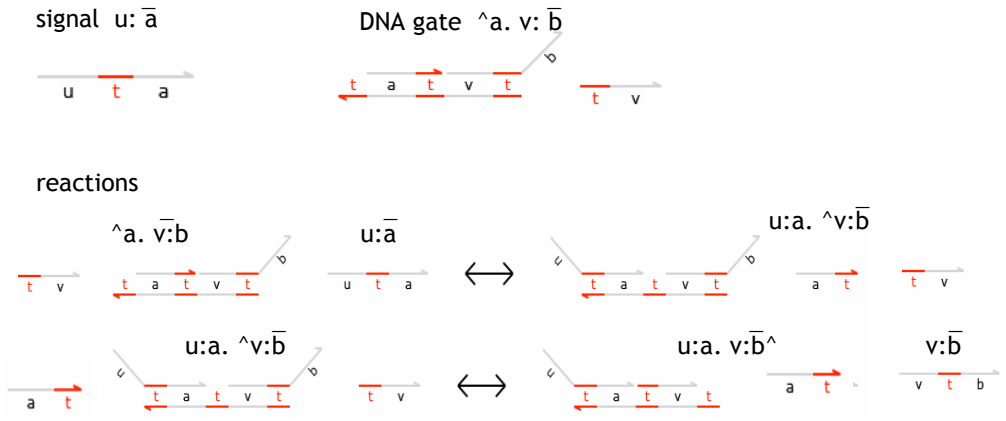


Figure 1: The DSD encoding of 1 input and 1 output gates and their reactions

– DNA *gates* are $\mathtt{G}_1\mathtt{:G}_2\mathtt{:}\ \cdots\ \mathtt{:G}_n$ where $\mathtt{G}_i$ may be either `t~` or `[a t~ ]` or `[t~ a]` or `<b>[a t~ ]` or `[a t~ ]<b>`;

Given a structural congruence definition similar to the one of reversible structures, the semantics of DSD is the least relation $\longrightarrow$ containing (structural congruence and reduction will be denoted as in reversible strand algebra):

– $\mathtt{G}_1\mathtt{:}\cdots\mathtt{:t\tilde{}\ :[a\ t\tilde{}\ ]:}\cdots\mathtt{:}\,\mathtt{G}_n \mid \mathtt{<u\ t\tilde{}\ a>} \longleftrightarrow \mathtt{G}_1\mathtt{:}\cdots\mathtt{:<u>[t\tilde{}\ a]:t\tilde{}\ :}\cdots\mathtt{:}\,\mathtt{G}_n \mid \mathtt{<a\ t\tilde{}\ >}$

– $\mathtt{G}_1\mathtt{:}\cdots\mathtt{:t\tilde{}\ :[a\ t\tilde{}\ ]<b>:}\cdots\mathtt{:}\,\mathtt{G}_n \mid \mathtt{<t\tilde{}\ a>} \longleftrightarrow \mathtt{G}_1\mathtt{:}\cdots\mathtt{:[t\tilde{}\ a]:t\tilde{}\ :}\cdots\mathtt{:}\,\mathtt{G}_n \mid \mathtt{<a\ t\tilde{}\ b>}$

(axioms are bidirectional, hence the symbol $\longleftrightarrow$) and closed under the same rules of reversible structures. Figure 1 illustrates strands, DNA gates and reductions of the DSD language. The encoding $(\!|\ \cdot\ |\!)$ of reversible structures to DSD terms is homomorphic with respect to parallel and new and it is defined on signals and gates as follows (for gates we only illustrate the encodings of configurations of $a_1.a_2.v_1 : \overline{b_1}.v_2 : \overline{b_2}$):

– $(\!|\ u : \overline{a}\ |\!) = \mathtt{<u\ t\tilde{}\ a>}$

– $(\!|\ \hat{}a_1.a_2.v_1 : \overline{b_1}.v_2 : \overline{b_2}\ |\!) =$
$\mathtt{t\tilde{}\ :[a_1\ t\tilde{}\ ]:[a_2\ t\tilde{}\ ]:[v_1\ t\tilde{}\ ]<b_1>:[v_2\ t\tilde{}\ ]<b_2>}$
$\mathtt{\mid\ <t\tilde{}\ v_1>\ \mid\ <t\tilde{}\ v_2>}$

– $(\!|\ u_1 : a_1.\hat{}a_2.v_1 : \overline{b_1}.v_2 : \overline{b_2}\ |\!) =$
$\mathtt{<u_1>[t\tilde{}\ a_1]:t\tilde{}\ :[a_2\ t\tilde{}\ ]:[v_1\ t\tilde{}\ ]<b_1>:[v_2\ t\tilde{}\ ]<b_2>}$
$\mathtt{\mid\ <a_1\ t\tilde{}\ >\ \mid\ <t\tilde{}\ v_1>\ \mid\ <t\tilde{}\ v_2>}$

– $(\!|\ u_1 : a_1.u_2 : a_2.v_1 : \overline{b_1}.\hat{}v_2 : \overline{b_2}\ |\!) =$
$\mathtt{<u_1>[t\tilde{}\ a_1]:<u_2>[t\tilde{}\ a_2]:[t\tilde{}\ v_1]:t\tilde{}\ :[v_2\ t\tilde{}\ ]<b_2>}$
$\mathtt{\mid\ <a_1\ t\tilde{}\ >\ \mid\ <a_2\ t\tilde{}\ >\ \mid\ <t\tilde{}\ v_2>}$

Figure 1 illustrates a sample encoding of 1 input and 1 output gate and its reactions. The strict correspondance between reversible structures and the DSD language is fixed by the following statement.

Proposition 2.5. $\mathtt{S} \longrightarrow \mathtt{S}'$ *implies* $(\!|\, \mathtt{S} \,|\!) \longrightarrow (\!|\, \mathtt{S}' \,|\!)$. *Additionally, if* $\mathtt{S}$ *is weak coherent then* $(\!|\, \mathtt{S} \,|\!) \longrightarrow \mathtt{S}'$ *implies there is* $\mathtt{S}''$ *such that* $\mathtt{S}' \equiv (\!|\, \mathtt{S}'' \,|\!)$ *and* $\mathtt{S} \longrightarrow \mathtt{S}''$.

The second part of Proposition 2.5 is restricted to weak coherent structures. In fact, $(\!|\, \mathtt{S} \,|\!) \longrightarrow (\!|\, \mathtt{S}' \,|\!)$ implies $\mathtt{S} \longrightarrow \mathtt{S}'$ is false in the unrestricted case. Consider the encoding of a gate $u : a.v : \overline{b}\hat{}$, namely `<u>[t~ a]:[t~ v]:t~ | <a t~ >`, and observe that, in this DNA gate, the co-name $\overline{b}$ never appears. If the (not weak coherent) structure also contained the signal $v : \overline{c}$, which is compiled into `<v t~ c>`, then the DNA structure might reduce to `<u>[t~ a]:t~ :[v t~ ]<c> | <a t~ >`. This last DNA structure encodes $u : a.\hat{}v : \overline{c}$ and cannot be obtained from the structure $u : a.v : \overline{b}\hat{} \mid v : \overline{c}$.

The correspondence between the DSD language and reversible structures has been crucial in the design of the latter ones. However, at this point a reader may wonder whether ids are really needed in these two formalisms: is it possible to define an id-free reversible structure and an encoding in the DSD language? The answer is positive. In this case signals are encoded in two-domains strands and gates have no overhangs [5]. However, in two-domains DSD structures it is not possible to define coherence (see Section 4) and to encode (in a causally consistent way) process calculi such as asynchronous RCCS. Said in a more effective way: the three-domains DSD (sub)language has the shortest domains that correctly implement reversibility of reversible process calculi.

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

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

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*: $u \,|\, \widetilde{v}\texttt{\^{}A.}\widetilde{w}$, *input release label*: $\widetilde{v}u\texttt{\^{}A.}\widetilde{w}$, *output release label*: $\widetilde{v}\texttt{.}\widetilde{w}\texttt{\^{}}u\widetilde{z}$, *output capture label*: $u \,|\, \widetilde{v}\texttt{.}\widetilde{w}u\texttt{\^{}}\widetilde{z}$. The labels of reductions are defined as follows. Let $id(\texttt{A}^{\perp}) = \widetilde{v}$, $id(\overline{\texttt{B}}) = \widetilde{w}$ and $id(\overline{\texttt{C}}) = \widetilde{z}$; we write $\mu : (\texttt{new}\ \widetilde{a})\texttt{S} \longrightarrow (\texttt{new}\ \widetilde{a})\texttt{S}'$, where $\texttt{S}$ and $\texttt{S}'$ do not contain news, when the axiom used in the proof tree is

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

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) and for defining the notion of residual of a gate (*cf.* the paragraph before Proposition 3.3). We notice that, in our case, this constraint does not bring any loss of generality.

In weak coherent structures, writing labels or reactants is almost the same. Let us write $\widehat{\widetilde{u}}$ and $\overline{\widetilde{u}}$, where $u = u_1 \cdots u_n$, be the sequences $u_1 : a_1\texttt{.}\cdots\texttt{.}u_n : a_n$ and $u_1 : \overline{a_1}\texttt{.}\cdots\texttt{.}u_n : \overline{a_n}$, respectively, where the structure associates to u_i a name a_i or a co-name $\overline{a_i}$ (by definition, the associations are uniquely defined in weak coherent structures). Then

- the label $u \,|\, \widetilde{v}\texttt{\^{}A.}\widetilde{w}$ specifies that the signal $\overline{u}$ and the gate $\widehat{\widetilde{v}}\texttt{.\^{}A.}\overline{\widetilde{w}}$ are the moving terms;
- the label $\widetilde{v}u\texttt{\^{}A.}\widetilde{w}$ specifies that $\widehat{\widetilde{v}u}\texttt{.\^{}A.}\overline{\widetilde{w}}$ is the moving gate;
- the label $\widetilde{v}\texttt{.}\widetilde{w}\texttt{\^{}}u\widetilde{z}$ specifies that $\widehat{\widetilde{v}}\texttt{.}\overline{\widetilde{w}}\texttt{.\^{}}\overline{u\widetilde{w}}$ is the moving gate;
- the label $u \,|\, \widetilde{v}\texttt{.}\widetilde{w}u\texttt{\^{}}\widetilde{z}$ specifies that the signal $\overline{u}$ and the gate $\widehat{\widetilde{v}}\texttt{.}\overline{\widetilde{w}u}\texttt{.\^{}}\overline{\widetilde{w}}$ are the moving terms.

If structures are not weak coherent then labels may fail to address reactants. For example, in $u : \overline{a} \mid u : \overline{b} \mid \texttt{\^{}}a.v : \overline{c} \mid \texttt{\^{}}a.v : \overline{d}$, the two reductions are both labelled $u \,|\, \texttt{\^{}}a\texttt{.}v$ 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 $\texttt{\^{}}$ is at the beginning of the output part of a gate. For example, the gate $u : a\texttt{.\^{}}v : \overline{b}$ may reduce either into $u : a.v : \overline{b}\texttt{\^{}}$ or into $\texttt{\^{}}a.v : \overline{b}$ (the signals are omitted). In order to separate the two reductions, we label the former with $u\texttt{.\^{}}v$ and the latter with $u\texttt{\^{}.}v$ (the positions of "$\texttt{\^{}}$"

and "$\cdot$" 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):

$$\begin{array}{rcl@{\qquad\qquad}rcl} [u \mid \widetilde{v}\hat{}a.\mathtt{A}.\widetilde{w}]^+ & \stackrel{\text{def}}{=} & \widetilde{v}u\hat{}\mathtt{A}.\widetilde{w} & [\widetilde{v}u\hat{}\mathtt{A}.\widetilde{w}]^+ & \stackrel{\text{def}}{=} & u \mid \widetilde{v}\hat{}a.\mathtt{A}.\widetilde{w} \\ [\widetilde{v}.\widetilde{w}\hat{}u\widetilde{z}]^+ & \stackrel{\text{def}}{=} & u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z} & [u \mid \widetilde{v}.\widetilde{w}u\hat{}\widetilde{z}]^+ & \stackrel{\text{def}}{=} & \widetilde{v}.\widetilde{w}\hat{}u\widetilde{z} \end{array}$$

$[\mu]^+$ is called "the converse label of μ" because the computations $\mu\,;\,[\mu]^+$ and $[\mu]^+\,;\,\mu$ do not change the initial structure (see Definition 3.2). 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 3.1. *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 3.1 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 3.1 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 $u \mid \hat{}a.v\ ;\ u.\hat{}v$.

Definition 3.2. Permutation equivalence*, written* $\sim$*, is the least equivalence relation between computations closed under composition and such that:*

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

For example, the computation

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

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 3.2 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.

Weak coherence guarantees the soundness of Definition 3.2. The (not weak coherent) structure $u : \overline{a} \mid \hat{}a.v : \overline{b} \mid u : a.\hat{}v : \overline{c}$ has a computation

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

whose labels are $u \mid \hat{}a.v \;;\; u\hat{}.v$, with $[u \mid \hat{}a.v]^+ = u\hat{}.v$. However, the two labels specify different gates and the above computation is not equivalent to ε.

Let the *gate of a label* be the gate, up-to structural congruence, involved in the reduction. Let $\mu : \mathtt{S} \longrightarrow \mathtt{S}'$ and let g be a gate in $\mathtt{S}$. We define g/μ, the *residual* of g after μ, the following gate in $\mathtt{S}'$:

$$g/\mu \stackrel{\text{def}}{=} \begin{cases} g & \text{if } g \text{ is not the gate of } \mu \\ g' & \text{if } g \text{ is the gate of } \mu \text{ and } g' \text{ is the gate of } [\mu]^+ \end{cases}$$

Proposition 3.3. *Let* $\mathtt{S}$ *be weak coherent and* $\mu : \mathtt{S} \longrightarrow \mathtt{S}'$. *(1) For every gate* g *in* $\mathtt{S}$ *there is a gate* g/μ *in* $\mathtt{S}'$*; (2) for every gate* g' *in* $\mathtt{S}'$ *there is a gate* g *in* $\mathtt{S}$ *such that* $g' = g/\mu$.

Theorem 3.4 (Standardization theorem)**.** *Let* $\mathtt{S}$ *be weak coherent and* $\mu_1 \;;\; \cdots \;;\; \mu_n$ *be a computation of* $\mathtt{S}$ *such that* μ_n *is the converse of* μ_1. *Then there is a shorter computation that is permutation equivalent to* $\mu_1 \;;\; \cdots \;;\; \mu_n$.

Proof. By induction on n. The cases $n \leq 2$ are either vacuous or obvious. Let $n \geq 3$. The argument for the inductive case analyzes the sequence $\mu_1 \;;\; \cdots \;;\; \mu_n$. Since μ_1 and μ_n are one the converse of the other, let μ_1 be the reduction whose label specifies the set $\{g_1\}$, where g_1 is a gate (it does not specify any signal). The argument is by cases on μ_2. Let $\mu_1 : \mathtt{S} \longrightarrow \mathtt{S}'$ and g_2 be the gate of μ_2 in $\mathtt{S}'$. By Proposition 3.3, let g_2' in $\mathtt{S}$ be such that $g_2 = g_2'/\mu$. If $g_1 \neq g_2'$ then μ_1 and μ_2 are coinitial and $\mu_1 \cap \mu_2 = \emptyset$. Therefore $\mu_1 \;;\; \mu_2 \;;\; \cdots \;;\; \mu_n \sim \mu_2 \;;\; \mu_1 \;;\; \cdots \;;\; \mu_n$ and we may apply the inductive hypothesis to $\mu_1 \;;\; \mu_3 \;;\; \cdots \;;\; \mu_n$. If $g_1 = g_2'$ and $\mu_2 = [\mu_1]^+$ then $\mu_1 \;;\; \mu_2 \;;\; \cdots \;;\; \mu_n \sim \mu_3 \;;\; \cdots \;;\; \mu_n$ and we are done. It remains the case $g_1 = g_2'$ and $\mu_1 = \mu_2$ then μ_n is the converse of μ_2 as well. We therefore use the inductive hypothesis on $\mu_2 \;;\; \cdots \;;\; \mu_n$. □

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 3.1 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].

We conclude this section with a comment about the computational complexity of the reachability problem in reversible structures – the existence of a

computation from one structure to another –, which is a relevant practical issue when structures represent DNA solutions. It is straightforward to encode reversible structures into symmetric (*a.k.a.* reversible) and bounded place-transition Petri nets. However, for these nets, the reachability marking problem is EXPSPACE complete [12, 6]. We are not aware of any better algorithm that improve, in our case, this limit. It is worth to remark that, even restricting our analysis to weak coherent structures, the conditions do not change very much because the problem reduces (by Theorem 3.4) to find the shortest computation in symmetric and bounded place-transition Petri net, which is the one returned by the algorithms in [12, 6]. In the next section we will study a refinement of coherence that retains better reachability algorithms.

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 $\mathtt{S}$ 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 $\mathit{type}(v : a.\hat{}\,a.u : \overline{a}.w : \overline{c}) = uw$ (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 4.1. *A weak coherent structure* $\mathtt{S}$ *is* coherent *whenever*

- *different gates in* $\mathtt{S}$ *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 2.4 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 4.1. It is possible to prove Proposition 2.4 for coherent structures.

Back to the mismatch between cofinality and permutation equivalence, if we weaken Definition 4.1 by admitting unconstrained occurrences of ids (the second constraint of coherence), then the problem remains. For example, the (not coherent) structure $u : \overline{a} \mid u : \overline{a} \mid \hat{}\,a.v : \overline{b} \mid \hat{}\,a.w : \overline{c}$ has two cofinal computations $u \,|\, \hat{}\,a.v \;;\; u \,|\, \hat{}\,a.w$ and $u \,|\, \hat{}\,a.w \;;\; u \,|\, \hat{}\,a.v$ that are not permutation equivalent.

Theorem 4.2. *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).*

Proof. The only-if direction is immediate by definition of $\sim$ and (strong) coherence. The if-direction is demonstrated by induction on $m+n$. The base case ($m+n=0$) is immediate. Assume the theorem holds when $m+n=h$, we prove the case $m+n=h+1$.

The interesting case is when Theorem 3.4 cannot be applied to the (sub)computations $\mu_1;\mu_2;\cdots;\mu_m$ and $\nu_1;\nu_2;\cdots;\nu_n$. Otherwise we use the inductive hypothesis.

So assume that $\mu_1;\mu_2;\cdots;\mu_m$ and $\nu_1;\nu_2;\cdots;\nu_n$ have no pair of converse labels. By coherence, if the types of two labels are equal then the corresponding reductions have the same direction (they are both forward or backward). By cofinality, the two computations must address the same gates and every gate appears the same number of times in labels. (Therefore $m=n$.) Since the computations do not contain converse labels then, projecting out labels of a same type, we obtain either sequences of input captures followed by output releases or sequences of output captures followed by input releases – therefore subsequences are *monotone*. Additionally, the projections of $\mu_1;\mu_2;\cdots;\mu_m$ and of $\nu_1;\nu_2;\cdots;\nu_n$ corresponding to a same type must be pairwise equal.

There are two cases: (a) one of the sequences has empty subsequence of captures, (b) every sequence has nonempty subsequence of captures. In case (a), the first label of the empty subsequence of captures may be permuted till the beginning of the sequences $\mu_1;\cdots;\mu_m$ and of $\nu_1;\cdots;\nu_n$, therefore we are reduced to shorter computations (because the first two labels are equal) and we may apply inductive hypothesis. In case (b), assume μ_1 is an input capture of a signal $u:\overline{a}$ and let ν_h be the first occurrence in $\nu_1;\cdots;\nu_n$ of the gate in μ_1. Because of cofinality, $\mu_1=\nu_h$. We demonstrate that ν_h may be permuted with $\nu_1;\cdots;\nu_{h-1}$ and the argument is as in case (a). If no label ν_1, $\cdots$, ν_{h-1} is an input capture of $u:\overline{a}$ then the permutation is possible. Otherwise, take the reduction ν_i, $1\leq i\leq h-1$, with largest i that performs an input capture. Then ν_i is either equal to $u\,|\,\widetilde{v}\hat{}a.\mathtt{A}.\widetilde{w}$ or equal to $u\,|\,\widetilde{v}.\widetilde{v'}u\hat{}\widetilde{w}$. In both cases, after the reduction ν_i the occurrence of u is positive. Since $u:\overline{a}$ cannot be released by the gate of ν_i (because this would contradict the monotony) and by any other gate in ν_{i+1} ; $\cdots$; ν_{h-1} (because this would contradict coherence) then it is impossible that ν_h performs an input capture.

The case when μ_1 is an output capture is similar. □

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.

We use the notion of *distance* between two gates with the same type as the least number of reductions to convert one gate into the other (it is infinite, otherwise). Formally, 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 $\sum_{g \in \mathtt{S}, g' \in \mathtt{S}', \mathit{type}(g) = \mathit{type}(g')} |g - g'|$.

Proposition 4.3. *Let* $\mathtt{S}$ *be a coherent structure and let* $\mathtt{S} \longrightarrow \mathtt{S}' \longrightarrow^* \mathtt{S}''$ *be a minimal computation (according to Theorem 3.4). Then* $|\mathtt{S} - \mathtt{S}''| > |\mathtt{S}' - \mathtt{S}''|$.

Proof. The proof is by contradiction and a case analysis on the reduction $\mathtt{S} \longrightarrow \mathtt{S}'$. One shows that, for every type of reduction, if $|\mathtt{S} - \mathtt{S}''| \leq |\mathtt{S}' - \mathtt{S}''|$ (actually, the equality is not possible) then $\mathtt{S}' \longrightarrow^* \mathtt{S}''$ must revert the reduction $\mathtt{S} \longrightarrow \mathtt{S}'$ thus contradicting the assumption of minimality. □

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 $\mathtt{S}$ with $\mathtt{S}'$.

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 $\mathtt{S}$ and let k be the maximal length of a gate in $\mathtt{S}$. 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 $O(2k^2 \times n^2)$.

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 4.2, which has been proved for RCCS in [8], or the above algorithm of reachability, which is original.

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 are defined by the following grammar (we are assuming that I are finite sets):

$$P \ ::= \quad \mathbf{0} \quad | \quad \textstyle\sum_{i\in I} \alpha_i.P_i \quad | \quad \prod_{i\in I} P_i \quad | \quad (\mathtt{new}\ a)P$$

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 $(\mathtt{new}\ a)P$ defines a name with scope P. Processes meet the following well-formed conditions:

- continuations of co-names are empty;
- in $\prod_{i\in I} P_i$ the processes P_i are guarded choices.

The semantics of asynchronous RCCS is defined in terms of a *transition relation* that uses

- *memories* m:

$$m \ ::= \quad \langle\,\rangle \quad | \quad \langle i\rangle_n \bullet m \quad | \quad \langle m, \alpha, Q\rangle \bullet m$$

- *run-time processes* R:

$$R \ ::= \quad m \triangleright P \quad | \quad R \mid R \quad | \quad (\mathtt{new}\ a)R$$

- *structural congruence* $\equiv$, defined in the standard way (see Section 2), plus the rules

$$m \triangleright (\textstyle\prod_{i\in 1..n} P_i) \ \equiv \ \prod_{i\in 1..n} \langle i\rangle_n \bullet m \triangleright P_i$$

$$m \triangleright (\mathtt{new}\ a)P \ \equiv \ (\mathtt{new}\ a)(m \triangleright P) \qquad (a \notin \mathsf{fn}(m))$$

The reduction relation $\longrightarrow$ is the least relation on run-time processes satisfying the axioms:

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

and closed under the contextual rules for parallel, new and structural congruence.

Let us have a short discussion that illustrates the main ideas of our encoding of RCCS before going into the details.

Consider the asynchronous RCCS process $a.P + \overline{a}$ that may progress either as P or terminate according to the external environment offers an output or an input on a, respectively. The structure encoding this process is

$$(\mathtt{new}\ c')((\hat{}\,c.a.u : \overline{c'} \ \mid \ [\![P]\!]_{c'}) \ \mid \ \hat{}\,c.v : \overline{a})$$

where $[\![P]\!]_{c'}$ is a structure encoding P such that every subterm is a gate performing an initial input capture on C'. 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 $\hat{}c.v : \overline{a}.w : \overline{c''}$ will react, let it be the second. Then the structure becomes

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

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

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

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.

Our encoding uses environments Γ that map memories to signals $u : \overline{c}$ such that:

1. (Γ is injective) if $m \neq m'$ then the signals $\Gamma(m)$ and $\Gamma(m')$ are different (they have different ids and co-names);
2. (Γ is prefix closed) if $s \bullet m \in \mathtt{dom}(\Gamma)$ then $m \in \mathtt{dom}(\Gamma)$ and
   - if $s = \langle i \rangle_n$ then $\langle 1 \rangle_n \bullet m, \cdots, \langle n \rangle_n \bullet m \in \mathtt{dom}(\Gamma)$;
   - if $s = \langle m', \alpha, P \rangle$ then $m' \in \mathtt{dom}(\Gamma)$.

Let $\mathsf{fn}(\Gamma) \stackrel{\text{def}}{=} \{a \mid \exists\, m, u.\ \Gamma(m) = u : \overline{a}\}$.

Let Γ be an environment such that, for every $i \in I$, $\Gamma(m_i) = u_i : \overline{c_i}$, for some u_i. The encoding $[\![\cdot]\!]^\Gamma$ is defined by

$$[\![\prod_{i\in I} m_i \triangleright P_i]\!]^\Gamma \ = \ (\mathtt{new}\ \mathsf{fn}(\Gamma))\Big(\prod_{i\in I}([\![P_i]\!]_{c_i} \ \mid\ \Gamma(m_i)) \ \mid\ \mathcal{U}(\{m_i \mid i \in I\}, \Gamma)\Big)$$

where the auxiliary functions $[\![P]\!]_c$ and $\mathcal{U}(M, \Gamma)$ are:

$$\begin{array}{rcl}
[\![\mathbf{0}]\!]_c & = & \hat{}c \\
[\![\sum_{i\in I} a_i.P_i + \sum_{j\in J} \overline{a_j}]\!]_c & = & (\mathtt{new}\ {c_i}^{i\in I})(\prod_{i\in I}(\hat{}c.a_i.u_i : \overline{c_i} \ \mid\ \prod_{i\in I}[\![P_i]\!]_{c_i}) \\
 & & \qquad \mid\ \prod_{j\in J} \hat{}c.u_j : \overline{a_j}) \\
 & & \qquad \text{where, for every } \ell \in I \cup J,\ u_\ell, u'_\ell \text{ are fresh} \\
[\![\prod_{i\in 1..n} P_i]\!]_c & = & (\mathtt{new}\ {c_i}^{i\in 1..n})(\hat{}c.u_1 : \overline{c_1}.\cdots.u_n : \overline{c_n} \ \mid\ \prod_{i\in 1..n}[\![P]\!]_{c_i}) \\
 & & \qquad \text{where } u_1, \cdots, u_n \text{ are fresh} \\
[\![(\mathtt{new}\ a)P]\!]_c & = & (\mathtt{new}\ a)([\![P]\!]_c)
\end{array}$$

$$\begin{array}{rcl}
\mathcal{U}(\{\langle 1\rangle_n \bullet m, \cdots, \langle n\rangle_n \bullet m\} \uplus M, \Gamma) & = & u : c.v_1 : \overline{c_1}.\cdots.v_n : \overline{c_n}\hat{} \\
& & \quad \mid \mathcal{U}(\{m\} \uplus M, \Gamma) \\
& & \text{where } \Gamma(m) = u : \overline{c} \\
& & \text{and } \Gamma(\langle i\rangle_n \bullet m) = v_i : \overline{c_i} \\
\\
\mathcal{U}(\{\langle m', a, P\rangle \bullet m, \langle m, \overline{a}, Q\rangle \bullet m'\} \uplus M, \Gamma) & = & u : c.w : a.v : \overline{c_1}\hat{} \mid [\![P]\!]_c \\
& & \quad \mid u' : c'.w : \overline{a}\hat{} \mid [\![Q]\!]_{c'} \\
& & \quad \mid \mathcal{U}(\{m, m'\} \uplus M, \Gamma) \\
& & \text{where } \Gamma(m) = u : \overline{c} \\
& & \text{and } \Gamma(m') = u' : \overline{c'} \\
& & \text{and } \Gamma(\langle m', a, P\rangle \bullet m) = v : \overline{c_1} \\
& & \text{and } w \text{ fresh}
\end{array}$$

As a consequence of the definition, the function $[\![\cdot]\!]^\Gamma$ always returns a coherent structure.

Lemma 5.1. *If $R \longrightarrow R'$ then, for suitable Γ and Γ', $[\![R]\!]^\Gamma \longrightarrow^* [\![R']\!]^{\Gamma'}$, with $id(\mu_i) \cap id(\Gamma(m) \mid \Gamma(m')) \neq \emptyset$, .*

Proof. Without loss of generality, let

$$R = m \triangleright \sum_{i\in I} a_i.P_i + \sum_{j\in J} \overline{a_j} \quad \mid \quad m' \triangleright \sum_{i\in I'} b_i.Q_i + \sum_{j\in J'} \overline{b_j} \quad \mid \quad m'' \triangleright P''$$

and let $\overline{a_h} = b_k = \overline{a}$ and $P' = \sum_{i\in I\setminus\{h\}} a_i.P_i + \sum_{j\in J} \overline{a_j}$ and $Q' = \sum_{i\in I'} b_i.Q_i + \sum_{j\in J'\setminus\{k\}} \overline{b_j}$. Then $R \longrightarrow \langle m', a, P'\rangle \bullet m \triangleright P_h \mid \langle m, \overline{a}, Q'\rangle \bullet m' \triangleright \mathbf{0} \mid m'' \triangleright P''$.

Let Γ be such that $\Gamma(m) = w : \overline{c}$, $\Gamma(m') = w' : \overline{c'}$ and $\Gamma(m'') = w'' : \overline{c''}$. Then

$$\begin{array}{rcl}
[\![R]\!]^\Gamma & \longrightarrow & (\mathtt{new}\ \widetilde{c})(\quad w : c.\hat{}a.u_k : \overline{c_k} \mid [\![P_k]\!]_{c_k} \mid \prod_{i\in I\cup J\setminus\{h\}} [\![P_i]\!]_c \\
& & \qquad \mid \quad w' : \overline{c'} \mid \prod_{i\in I'\cup J'} [\![Q_i]\!]_{c'} \\
& & \qquad \mid \quad w'' : \overline{c''} \mid [\![P'']\!]_{c''} \mid \mathcal{U}(\{m, m', m''\}, \Gamma) \quad) \\
& \longrightarrow & (\mathtt{new}\ \widetilde{c})(\quad w : c.\hat{}a.u_k : \overline{c_k} \mid [\![P_k]\!]_{c_k} \mid \prod_{i\in I\cup J\setminus\{h\}} [\![P_i]\!]_c \\
& & \qquad \mid \quad w' : c'.\hat{}v'_h : \overline{a} \mid \prod_{i\in I'\cup J'\setminus\{h\}} [\![Q_i]\!]_{c'} \\
& & \qquad \mid \quad w'' : \overline{c''} \mid [\![P'']\!]_{c''} \mid \mathcal{U}(\{m, m', m''\}, \Gamma) \quad) \\
\\
& \longrightarrow & (\mathtt{new}\ \widetilde{c})(\quad w : c.\hat{}a.u_k : \overline{c_k} \mid [\![P_k]\!]_{c_k} \mid \prod_{i\in I\cup J\setminus\{h\}} [\![P_i]\!]_c \\
& & \qquad \mid \quad v'_h : \overline{a} \mid w' : c'.v'_h : \overline{a}.\hat{} \mid \prod_{i\in I'\cup J'\setminus\{h\}} [\![Q_i]\!]_{c'} \\
& & \qquad \mid \quad w'' : \overline{c''} \mid [\![P'']\!]_{c''} \mid \mathcal{U}(\{m, m', m''\}, \Gamma) \quad) \\
\\
& \longrightarrow & (\mathtt{new}\ \widetilde{c})(\quad w : c.v'_h : a.\hat{}u_k : \overline{c_k} \mid [\![P_k]\!]_{c_k} \mid \prod_{i\in I\cup J\setminus\{h\}} [\![P_i]\!]_c \\
& & \qquad \mid \quad w' : c'.v'_h : \overline{a}.\hat{} \mid \prod_{i\in I'\cup J'\setminus\{h\}} [\![Q_i]\!]_{c'} \\
& & \qquad \mid \quad w'' : \overline{c''} \mid [\![P'']\!]_{c''} \mid \mathcal{U}(\{m, m', m''\}, \Gamma) \quad) \\
\\
& \longrightarrow & (\mathtt{new}\ \widetilde{c})(\quad u_k : \overline{c_k} \mid [\![P_k]\!]_{c_k} \mid w : c.v'_h : a.u_k : \overline{c_k}\hat{} \mid \prod_{i\in I\cup J\setminus\{h\}} [\![P_i]\!]_c \\
& & \qquad \mid \quad w' : c'.v'_h : \overline{a}.\hat{} \mid \prod_{i\in I'\cup J'\setminus\{h\}} [\![Q_i]\!]_{c'} \\
& & \qquad \mid \quad w'' : \overline{c''} \mid [\![P'']\!]_{c''} \mid \mathcal{U}(\{m, m', m''\}, \Gamma) \quad) \\
\\
& = & [\![R']\!]^{\Gamma'} \quad \text{where } \Gamma' = \Gamma[\langle m', a, P'\rangle \bullet m \mapsto u_k : \overline{c_k}\ ;\ \langle m, \overline{a}, Q'\rangle \bullet m' \mapsto v''_h : \overline{c_h}]
\end{array}$$

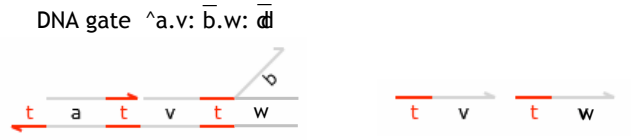


Figure 2: The encoding of $a.v : \overline{c}.w : \overline{\mathbf{del}}$ in the DSD language

Since our structures and asynchronous RCCS are both reversible, the above computation also demonstrates that $R' \longrightarrow R$ implies $[\![R']\!]^{\Gamma'} \longrightarrow^* [\![R]\!]^{\Gamma}$. □

6 Irreversible combinators

Having developed a theory for reversible structures, we now analyze how to extend our calculus in order to integrate (irreversible) strand algebra [4].

Let us add an *irreversible co-name*, noted $\overline{\mathbf{del}}$, to the set of co-names. There is no name corresponding to $\overline{\mathbf{del}}$ and we assume that occurrences of $\mathbf{del}$ in gates, if any, are in the last position. For example, $\hat{}a.u : \overline{b}.v : \overline{\mathbf{del}}$ and $w : a.u : \overline{b}.v : \overline{\mathbf{del}}\hat{}$ are valid gates. We also assume that structures never retain signals with co-name $\overline{\mathbf{del}}$.

The reduction relation of Definition 2.1 is then expanded with

$$\mathtt{A}^{\perp}.\overline{\mathtt{B}}.\hat{}u : \overline{\mathbf{del}} \longrightarrow \mathtt{A}^{\perp}.\overline{\mathtt{B}}.u : \overline{\mathbf{del}}\hat{}$$

that, contrary to the other output release rules, does not emit any signal $u : \overline{\mathbf{del}}$ and has no associated converse reduction. The intended meaning is that, while reversibility is admitted up-to the signal preceding $\overline{\mathbf{del}}$, it is forbidden once $\overline{\mathbf{del}}$ has been consumed.

This extension of reversible structures is expressive enough to encode

– the (irreversible) strand algebra in [4]. We only illustrate the encoding $[\![\cdot]\!]$ of an irreversible strand $a.\overline{b}$ (v and w are fresh ids):

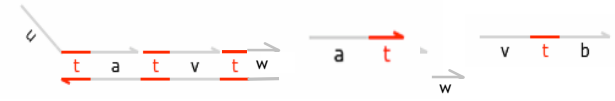
$$[\![a.\overline{b}]\!] \stackrel{\text{def}}{=} a.v : \overline{b}.w : \overline{\mathbf{del}}\,.$$

– the (irreversible) guarded choice in asynchronous CCS. We only illustrate the encoding $\llbracket \cdot \rrbracket$ of $a.P + \overline{b}$ (u, u', v, v' are fresh ids):

$$\llbracket a.P + \overline{b} \rrbracket \stackrel{\text{def}}{=} (\hat{}c.a.u : \overline{c'}.u' : \overline{\mathbf{del}} \;\mid\; \llbracket P \rrbracket_{c'}) \;\mid\; (\hat{}c.v : \overline{b}.v' : \overline{c''} \;\mid\; \llbracket \mathbf{0} \rrbracket_{c''})$$

where the irreversible reduction is performed once the input continuation has been triggered. The encoding of output has no tailing $\overline{\mathbf{del}}$ combinator.

Figure 2 defines the translation of the gate $a.v : \overline{b}.w : \overline{\mathbf{del}}$ in the DSD language. According to the semantics of the DSD language, if a strand `<u t~ a>` arrives we will obtain the DSD term



that cannot be reverted to the initial one. We ignore the minor issue of the inert "w" segment that is left over by the DSD reduction.

7 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 4.2 (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. The presence of irreversible combinators makes this implementation more interesting because it paves the way for modelling standard irreversible operators of programming languages.

Comparing *in vivo* and *in silico* implementations of programming languages is an exciting research direction both for biology and computer science, definitely.

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.