Open access • Book Chapter • DOI:10.1007/978-3-642-36315-3_18

# A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway



Iain Phillips, Irek Ulidowski, Shoji Yuen

**Institutions:** Imperial College London, University of Leicester, Nagoya University

**Published on:** 02 Jul 2012 - Reversible Computation

# A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway

Iain Phillips Irek Ulidowski Shoji Yuen

**Abstract**

We introduce a reversible process calculus with a new feature of execution control that allows us to change the direction and pattern of computation. This feature allows us to model a variety of modes of reverse computation, ranging from strict backtracking to reversing which respects causal ordering of events, and even reversing which violates causal ordering. The SOS rules that define the operators of the new calculus employ communication keys to handle communication correctly and key identifiers to control execution.

As an application of our calculus, we model the ERK signalling pathway which delivers mitogenic and differentiation signals from the membrane of a cell to its nucleus. The proteins participating in the pathway are represented by reversible processes in such a way that the pathway's bio-chemical reactions are simply interactions between the processes.

## 1 Introduction

Reversing computation of a concurrent system poses a number of conceptual and technical questions. How is the forward and reverse computation performed and controlled? When reversing, in what order are computation steps undone? We answer the last question first. Consider a computation where the event $a$ causes the event $b$, written $a < b$, and the event $c$ occurs at another location independently of $a$ and $b$. The three traces of this computation that preserve causality are $abc$, $acb$ and $cab$: note that $a$ always precedes $b$. There are several conceptually different ways of undoing these events. *Backtracking* is undoing in precisely the reverse order in which they happened. So, undo $b$ undo $c$ undo $a$ is a backtrack of $acb$.

*Reversing* is a more general form of undoing: here events can be undone in any order as long as causality is preserved, meaning that causes cannot be undone before effects. For example, undo $c$ undo $b$ undo $a$ is a reversal of $acb$ for $a, b$ and $c$ as defined above. However, and quite surprisingly, there are situations where events happen, or are undone, *out of causal order*. The creation and breaking of molecular bonds between the proteins involved in the ERK signalling pathway described in Section 3 is a good example. Simplifying, let us assume that the creation of molecular bonds is represented by events $a, b, c$

where, as above, $a < b$ and $c$ is independent of $a$ and $b$. In the ERK pathway, the molecular bonds are broken in the following order: undo $a$ undo $b$ undo $c$, which seems to undo the cause $a$ before the effect $b$. Similarly, an execution of multi-threaded programs under weak memory models or under the out-of-order regime may result in traces which contradict program (thread) order; this is a result of well-known hardware or compiler optimisations. In this paper we propose a reversible process calculus in which we can model reversibility and out-of-order computation. To the best of our knowledge this is the first such calculus.

We return to the question of how to control the direction of computation. Reversible process calculi RCCS [7] and CCS with communication Keys (CCSK) [15, 16] use a memory with a history of past computation and communication keys, respectively, to reverse computation so that causality is preserved. Reversible systems modelled in RCCS and CCSK choose the direction of computation spontaneously. When an execution of a fault-tolerant system encounters an error, the system recovers by undoing execution to a state where the error can be eliminated. The instruction when and how far to reverse is a part of the system's software and such reversibility control mechanism can be modelled by the rollback construct of the higher-order $\pi$-calculus [10]. In this paper we propose a different and more expressive mechanism for controlling reversibility. Its operational formulation and usefulness compares favourably with that of the rollback construct, and it allows us to model additionally out-of-order forward and reverse computation which we believe has not been done before in the process calculi setting.

Our calculus is an extension of CCSK [15, 16], a reversible process calculus based on Milner's CCS [13], with prefixing by multisets of actions and with an execution control mechanism (controllers). The generalised prefixing gives us the ability to represent a loose relationship between events of out-of-order computation and, more specifically, it allows us to model more faithfully the structure and reactions of bio-chemical molecules. This form of prefixing was previously employed in [9]. Controllers permit us to manage the pattern and the direction of computation and together with the multiset prefixing they are able to model out-of-order computation (note that weaker forms of prefixing are not sufficient). It is a different form of the rollback construct of the higher-order $\pi$ calculus [10].

Processes and controllers are quite strongly contrasted: processes (without controllers) can compute freely either forwards or in reverse, whereas controllers can only compute forwards (even when the process under control is reversing). We envisage a wide variety of uses for controllers, ranging from handling error recovery to providing the main focus of the computation, as in the bio-chemical example we present later.

We give SOS rules for the operators of our calculus in Section 2. The rules for reversing computation are simply symmetric versions of the forward rules. In order to manage correctly both communication and the reversing of communication we employ communication keys [15, 16]. The new notion of key identifiers is introduced to mark the actions of processes that are to be performed or undone,

thus giving us the ability not only to reverse specific past actions, as achieved by the rollback [10], but also to specify which forward action to compute and when to compute them. In this way, we achieve a more general mechanism for controlling computation. To illustrate this, consider a process that can perform actions $a$ and $b$ in parallel. We can define a controller that forces $b$ to execute always after $a$, effectively setting $a$ as the cause of $b$. In the standard setting, this means that reversing $a$ must be proceeded by undoing $b$. However, our control mechanism gives the ability to reverse $a$ and $b$ 'out of order': first $a$ and then $b$. Such patterns of computation, seemingly breaking the causal relationships between actions, are common in the bio-chemical setting as can be seen in our model of the ERK signalling pathway.

The usefulness of the execution control mechanism in exhibited in several examples. In Section 2.2 we consider the modelling of long-running transactions with compensations and we re-work the example from [10] of a system with complex causal dependencies between executing and reversing communications. The first example shows the need for the new key identifiers, whereas in the second example communication keys alone suffice. The second part of the paper (Section 3) is devoted solely to the modelling of the ERK signalling pathway [5, 20], which delivers mitogenic and differentiation signals from the membrane of a cell to its nucleus, and how it is regulated by RKIP proteins. There, the execution control mechanism and prefixing with multisets of actions play a vital rôle.

The research on reversing process calculi can be traced back perhaps to the work by Berry and Boudol on the Chemical Abstract Machine [1]. We were inspired to look at reversible computation by, among others, the paper of Danos and Krivine on reversing CCS [9] and the subsequent [7, 8]. We then proposed an alternative, more algebraic method for reversing CCS in [15, 16], and recently provided both bisimulation and modal logic semantics for reversible concurrency [17, 18]. Lanese, Mezzina, Schmitt and Stefani proposed a reversible version of a higher-order $\pi$ calculus and equipped it with a rollback construct [11, 10]. They also studied other forms of reversibility for defining programming abstractions for dependable distributed systems, and discussed the need for compensations [12]. Finally, reversible structures that compute forwards and backwards in an asynchronous manner were proposed by Cardelli and Laneve [4].

# 2 A reversible process calculus with execution control

In this section we extend CCSK with an execution control mechanism which allows us to control the direction and the pattern of computation. The extended calculus is given an operational semantics and its usefulness is illustrated in several examples including long-running transactions with compensations.

$$\frac{\mathsf{std}(X)}{\alpha[v].X \overset{\alpha[n,v]}{\to} \alpha[n,v].X} \qquad \frac{X \overset{\mu[n,v]}{\to} X'}{\alpha[m,u].X \overset{\mu[n,v]}{\to} \alpha[m,u].X'}\, m \neq n$$

$$\frac{X \overset{\mu[n,v]}{\to} X' \quad \mathsf{fsh}[n](Y)}{X \,|\, Y \overset{\mu[n,v]}{\to} X' \,|\, Y} \qquad \frac{X \overset{\alpha[n,v]}{\to} X' \quad Y \overset{\overline{\alpha}[n,u]}{\to} Y'}{X \,|\, Y \overset{\tau[n]}{\to} X' \,|\, Y'}$$

$$\frac{X \overset{\mu[n,v]}{\to} X' \quad \mathsf{std}(Y)}{X + Y \overset{\mu[n,v]}{\to} X' + Y} \qquad \frac{X \overset{\mu[n,v]}{\to} X'}{X \backslash A \overset{\mu[n,v]}{\to} X' \backslash A}\, \mu, \overline{\mu} \notin A \qquad \frac{X \overset{\mu[n,v]}{\to} X'}{X[f] \overset{f(\mu)[n,v]}{\to} X'[f]}$$

$$\frac{\mathsf{std}(X)}{\alpha[n,v].X \overset{\alpha[n,v]}{\rightsquigarrow} \alpha[v].X} \qquad \frac{X \overset{\mu[n,v]}{\rightsquigarrow} X'}{\alpha[m,u].X \overset{\mu[n,v]}{\rightsquigarrow} \alpha[m,u].X'}\, m \neq n$$

$$\frac{X \overset{\mu[n,v]}{\rightsquigarrow} X' \quad \mathsf{fsh}[n](Y)}{X \,|\, Y \overset{\mu[n,v]}{\rightsquigarrow} X' \,|\, Y} \qquad \frac{X \overset{\alpha[n,v]}{\rightsquigarrow} X' \quad Y \overset{\overline{\alpha}[n,u]}{\rightsquigarrow} Y'}{X, Y \overset{\tau[n]}{\rightsquigarrow} X' \,|\, Y'}$$

$$\frac{X \overset{\mu[n,v]}{\rightsquigarrow} X' \quad \mathsf{std}(Y)}{X + Y \overset{\mu[n,v]}{\rightsquigarrow} X' + Y} \qquad \frac{X \overset{\mu[n,v]}{\rightsquigarrow} X'}{X \backslash A \overset{\mu[n,v]}{\rightsquigarrow} X' \backslash A}\, \mu, \overline{\mu} \notin A \qquad \frac{X \overset{\mu[n,v]}{\rightsquigarrow} X'}{X[f] \overset{f(\mu)[n,v]}{\rightsquigarrow} X'[f]}$$

Figure 1: Forward and reverse SOS rules.

## 2.1 CCSK

We define the (forward) actions of CCS as usual: let $\mathcal{A}$ be a set of actions $a$, let $\overline{a}$ be the *complement* of $a$, and let $\overline{\mathcal{A}} = \{\overline{a} : a \in \mathcal{A}\}$. Also, let $\overline{\overline{a}} = a$ for $a \in \mathcal{A}$. We assume that $\alpha, \beta$ range over $\mathcal{A} \cup \overline{\mathcal{A}}$, and $\mu, \nu$ range over all actions, namely $\mathsf{Act} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$, where $\tau \notin \mathcal{A}$ is the silent action and $\overline{\overline{\tau}} = \tau$. Let $\mathcal{K}$ be an infinite set of *communication keys* (or just *keys* for short), ranged over by $k, m, n$. And, let $\mathcal{I}$ be an infinite set of key identifiers, ranged over by $v, u, w$. We also have a set of process identifiers $\mathcal{PI}$ , with typical elements $S, T$, and a set of variables, ranged over by $X, Y$. $\mathcal{PI}$ contains the deadlocked process $\mathbf{0}$.

The syntax of CCSK is given below, where $A \subseteq \mathsf{Act} \setminus \{\tau\}$ and $f : \mathsf{Act} \to \mathsf{Act}$ with $f(\tau) = \tau$. The set of CCSK closed terms is $\mathcal{P}$, and we shall refer to closed terms as processes. We let $P, Q$ to range over processes.

$$P \quad ::= \quad X \;\mid\; S \;\mid\; \alpha[v].P \;\mid\; \alpha[n,v].P \;\mid\; P + Q \;\mid\; P \,|\, Q \;\mid\; P \backslash A \;\mid\; P[f]$$

The prefixing with forward actions operator is $\alpha[v].X$ where $v$ is a key identifier and is optional. Each $\{\alpha, \overline{\alpha}\}$ (and $\{\alpha, \overline{\alpha}, \underline{\alpha}, \underline{\overline{\alpha}}\}$ in Section 2.2) has a set of key identifiers associated with it, and we assume that all such sets are disjoint. Prefixing with past actions has the form $\alpha[n,v].X$ where $n$ is the specific key for performing this $\alpha$ and $v$ is drawn from the set of key identifiers for $\alpha$, and may be omitted if it plays no rôle (but the key $n$ must occur). There is no

prefixing with $\tau$. We often omit trailing $\mathbf{0}$s so, for example, $a.\mathbf{0}$ is written as $a$.

$X \,|\, Y$ represents two systems $X$ and $Y$ that can perform actions or reverse actions on their own or they can interact with each other on complementary actions, for example $a$ and $\overline{a}$. The choice, restriction and relabelling operators, namely '$\cdot + \cdot$', '$\cdot \setminus A$' and '$\cdot[f]$', are as in CCS except that $+$ is now static in process terms. Each process identifier $S$ has a defining equation $S \stackrel{\text{df}}{=} P$.

The SOS forward and reverse SOS rules for CCSK are given in Figure 1. Note that the reverse rules are simply the reversals of the forward rules. We associate with each term $X$ the set of its keys, written as $\mathsf{keys}(X)$. A term $X$ is standard, written $\mathsf{std}(X)$, if it contains no prefixing with past actions. A key $n$ is fresh in $Y$, written $\mathsf{fsh}[n](Y)$, if $n$ is not used in $Y$.

Structural congruence $\equiv$ on terms is defined by $X \,|\, Y \equiv Y \,|\, X$, $X \,|\, (Y \,|\, Z) \equiv (X \,|\, Y) \,|\, Z$ and $X \,|\, \mathbf{0} \equiv X$. Also, $X + Y \equiv Y + X$, $X + (Y + Z) \equiv X + (Y + Z)$ and $X + \mathbf{0} = X$. And $S \equiv P$ for all $S$ and $P$ such that $S \stackrel{\text{df}}{=} P$. We also have the Structural Congruence Rule:

$$\frac{X \equiv Y \quad Y \stackrel{\mu[n,v]}{\rightarrow} Y' \quad Y' \equiv X'}{X \stackrel{\mu[n,v]}{\rightarrow} X'}$$

**Example 2.1** *In CCSK we keep track of the identities of actions that communicate so that when we reverse we undo the correct past actions. Consider* $P \stackrel{\text{df}}{=} (a \,|\, a.c \,|\, \overline{a} \,|\, \overline{a}.e) \setminus a$. *Here the restriction of* $a$ *prevents* $a$ *and* $\overline{a}$ *being performed except as part of a communication. Suppose that* $a$ *communicates with* $\overline{a}$ *and then* $a.c$ *with* $\overline{a}.e$. *In CCSK we write this as follows:*

$$P \equiv \stackrel{\tau[m]}{\rightarrow} (a[m] \,|\, a.c \,|\, \overline{a}[m] \,|\, \overline{a}.e) \setminus a \stackrel{\tau[n]}{\rightarrow} (a[m] \,|\, a[n].c \,|\, \overline{a}[m] \,|\, \overline{a}[n].e) \setminus a$$

*Note that the process* $a[m] \,|\, a.c \,|\, \overline{a}[m] \,|\, \overline{a}.e$ *cannot regress by reversing* $a[m]$ *alone because key* $m$ *is not fresh in* $a.c \,|\, \overline{a}[m] \,|\, \overline{a}.e$. *The fact that* $m$ *appears in* $a.c \,|\, \overline{a}[m] \,|\, \overline{a}.e$ *which is in parallel with* $a[m]$ *proves that the processes communicated with* $a$ *and* $\overline{a}$ *rather than performed them independently.*

*Our notation does not allow us to backtrack by undoing a different pair of actions, but clearly we can change the order of reversing* $\tau[m]$ *and* $\tau[n]$*:*

$$(a[m] \,|\, a[n].c \,|\, \overline{a}[m] \,|\, \overline{a}[n].e) \setminus a \stackrel{\tau[m]}{\rightsquigarrow} (a \,|\, a[n].c \,|\, \overline{a} \,|\, \overline{a}[n].e) \setminus a \stackrel{\tau[n]}{\rightsquigarrow} \equiv P$$

CCSK processes are fully reversible because the reverse SOS rules in Figure 1 are obtained by simply reversing the forward SOS rules in Figure 1 [15, 16]. We have $P \stackrel{\mu[n,v]}{\rightarrow} Q$ iff $Q \stackrel{\mu[n,v]}{\rightsquigarrow} P$ for all processes $P, Q$ and all $\mu \in \mathsf{Act}, n \in \mathcal{K}, v \in \mathcal{I}$. Moreover, CCSK is a conservative extension of CCS [15, 16].

## 2.2 Execution control operator

We add a new operator '$\cdot\langle\cdot\rangle$' to CCSK for controlling the execution of processes. We shall need new actions that control the reversing of the forward actions: $\underline{a}$

$$\text{(cf1)}\ \frac{X \overset{\alpha[n,v]}{\rightarrow} X' \quad Y \overset{\alpha[n,v]}{\rightarrow} Y'}{X\langle Y\rangle \overset{\alpha[n,v]}{\rightarrow} X'\langle Y'\{n,v/v\}\rangle} \quad > \quad \text{(cf2)}\ \frac{X \overset{\beta[m,u]}{\rightarrow} X' \quad Y \overset{\alpha'[k,w]}{\rightarrow} Y'}{X\langle Y\rangle \overset{\beta[m,u]}{\rightarrow} X'\langle Y\{m,u/u\}\rangle}$$

$$\text{(cr1)}\ \frac{X \overset{\alpha[n,v]}{\rightsquigarrow} X' \quad Y \overset{\underline{\alpha}[n,v]}{\rightarrow} Y'}{X\langle Y\rangle \overset{\alpha[n,v]}{\rightsquigarrow} X'\langle Y'\{v/n,v\}\rangle} \quad > \quad \text{(cr2)}\ \frac{X \overset{\beta[m,u]}{\rightsquigarrow} X' \quad Y \overset{\underline{\alpha'}[k,w]}{\rightarrow} Y'}{X\langle Y\rangle \overset{\beta[m,u]}{\rightsquigarrow} X'\langle Y\{u/m,u\}\rangle}$$

Figure 2: SOS rules for the control operator.

and $\underline{\overline{a}}$ prompt reversing of the past versions of $a$ and $\overline{a}$ respectively. Thus, we have two further sets $\underline{\mathcal{A}}$ and $\underline{\overline{\mathcal{A}}}$. We let $\underline{\alpha}, \underline{\beta}$ range over $\underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}}$, $\kappa$ range over $\mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}}$ and, from now on, we let $\mu, \nu$ range over all actions, namely $\mathsf{Act} = \mathcal{A} \cup \overline{\mathcal{A}} \cup \underline{\mathcal{A}} \cup \underline{\overline{\mathcal{A}}} \cup \{\tau\}$.

$X\langle Y\rangle$ is the process $X$ controlled by $Y$. The behaviour of $X\langle Y\rangle$ is a subset of the behaviour of $X$ as prescribed by $Y$ according to the rules in Figure 2 which are in the Ordered SOS format [19, 14]. Before we explain these rules and how the control operator works, we define control terms and update the definition of processes. The syntax for control terms is given below. Closed control terms, or simply control terms or controllers, are ranged over by $C, D$.

$$C ::= X \ \mid\ c \ \mid\ \kappa[v].C \ \mid\ \kappa[n,v].C \ \mid\ C + D \ \mid\ C \,|\, D$$

Terms $c$ are typical elements of a set of control identifiers. By abuse of notation we shall often use $C, D$ for control identifiers. Every control identifier has a defining equation $c \overset{\text{df}}{=} C$; we extend the definition of $\equiv$ by $c \equiv C$ for all $c, C$ such that $c \overset{\text{df}}{=} C$. The SOS rules for the operators of control terms are the standard SOS rules for CCS, except that we have prefixing with new actions and prefixing carries keys or key identifiers. Note that the prefixing and $+$ operators are dynamic operators as in CCS. Thus, controllers compute forwards only so, for example, $\kappa[v].C \overset{\kappa[k,v]}{\rightarrow} C$, for some $k$, and $\kappa[v].C + D \overset{\kappa[k,v]}{\rightarrow} C$.

The class of processes is extended to include terms $P\langle C\rangle$ for all $P$ and $C$.

Returning to Figure 2, the notation (cf1) $>$ (cf2) means that (cf2) can be applied to derive a transition of $P\langle C\rangle$ if no rules higher in the ordering $>$ can be applied, namely the rules (cf1) are not applicable for all $\alpha, n, v$. So, if $C$ can perform any forward $\alpha'[k,w]$ and $P$ cannot perform any of the forward actions $\alpha[n,v]$ of $C$, then (cf2) can be used to derive $P\langle C\rangle \overset{\beta[m,u]}{\rightarrow} P'\langle C\{m,u/u\}\rangle$ if $P' \overset{\beta[m,u]}{\rightarrow} P'$. We note that $C\{m,u/u\}$ means that every occurrence of $u$ in $C$ is replaced with $m, u$. The controller keeps track of which actions to reverse or to perform by recording keys and key identifiers shared with its process.

Actions $\underline{\alpha}$ of the controller require $X$ to reverse until $\alpha$ is undone. The rules (cr1) and (cr2) play the dual rôle to (cf1) and (cf2). Here, we replace the key and the key identifier in the controller with the key identifier alone, thus wiping out the record of the keys of the reversed transitions.

Terms such as $a[v].b[v]$ are not well formed as different actions cannot share identifiers. Some well formed terms are not very useful, for example only the first $a$ can execute in $a[v].a[v]$.

**Example 2.2** *Consider* $P \stackrel{\text{df}}{=} a.a.b.b$. *If* $C' \stackrel{\text{df}}{=} b.\underline{a}.b$ *then* $P\langle C'\rangle$ *computes until after the first* $b$ *of* $P$, *then reverses until the second* $a$ *is undone and finally it computes until after the first* $b$. *If we wish to compute or reverse other occurrences of actions* $a$ *and* $b$ *in* $P$, *for example the first* $a$ *and the second* $b$, *then we use key identifiers. The controller* $C \stackrel{\text{df}}{=} b[v].\underline{a}[u].b[v]$ *achieves this provided that the appropriate actions* $a$ *and* $b$ *in* $P$ *are marked with* $u$ *respectively* $v$. *Let* $P \stackrel{\text{df}}{=} a[u].a.b.b[v]$. *Then, using rule (cf2), we obtain* $P\langle C\rangle \stackrel{a[1,u]}{\rightarrow} a[1,u].a.b.b[v]\ \langle b[v].\underline{a}[1,u].b[v]\rangle$. *Note that prefixing with* $\underline{a}[u]$ *in the controller has been updated with the key 1. After another forward* $a$ *and a* $b$, *we use rule (cf1) to perform* $b[4,v]$ *(*$b[v]$ *with the key 4); note that the second* $b[v]$ *in* $C$ *is updated to* $b[4,v]$*:*

$$\stackrel{a[2]}{\rightarrow}\stackrel{b[3]}{\rightarrow}\stackrel{b[4,v]}{\rightarrow} \quad a[1,u].a[2].b[3].b[4,v]\ \langle \underline{a}[1,u].b[4,v]\rangle.$$

*Then we reverse until we have undone the* $a[1,u]$ *using (cr2) and (cr1):*

$$\stackrel{b[4,v]}{\rightsquigarrow}\stackrel{b[3]}{\rightsquigarrow}\stackrel{a[2]}{\rightsquigarrow}\stackrel{a[1,u]}{\rightsquigarrow} a[u].a.b.b[v]\ \langle b[v]\rangle.$$

*Note that this reversal wipes out all the keys. Finally, we can compute forwards.*

The control operator is very expressive. Consider a process $P$. Process $P\langle \mathbf{0}\rangle$ behaves as $\mathbf{0}$. If $a$ is not in the sort of $P$ (the set of actions that $P$ can ever perform), then $P\langle a \,|\, \underline{a}\rangle$ and $P\langle a + \underline{a}\rangle$ behave exactly as $P$. If we allowed prefixing with $\tau$ in control terms, then $C \stackrel{\text{df}}{=} \tau.C$ would force communications in $P$ thus acting as the restriction operator of CCS.

The control operator can be used to make the forward actions of processes irreversible. Consider $a.b$ and $C \stackrel{\text{df}}{=} b.\underline{b}.C$. Then $(a.b)\langle C\rangle \stackrel{a[1]}{\rightarrow} (a[1].b)\langle C\rangle \stackrel{a[1]}{\not\rightsquigarrow}$ since $C$ insists on computing forwards with $b$. Also we can find examples where $Q \stackrel{\mu[n,v]}{\rightsquigarrow} P$ holds but there is no $Q'$ such that $P \stackrel{\mu[n,v]}{\rightarrow} Q'$. Consider $(a \,|\, b)\langle C\rangle$ where $C \stackrel{\text{df}}{=} a.b.\underline{a}.\underline{b}.C$. We have $(a \mid b)\langle C\rangle \stackrel{a[1]}{\rightarrow}\stackrel{b[2]}{\rightarrow} (a[1] \mid b[2])\langle \underline{a}.\underline{b}.C\rangle \stackrel{a[1]}{\rightsquigarrow} (a \mid b[2])\langle \underline{b}.C\rangle \stackrel{b[2]}{\rightsquigarrow} (a \,|\, b)\langle C\rangle$ but not $(a \,|\, b)\langle C\rangle \stackrel{b[2]}{\rightarrow} Q'$ since $C$ insists on performing $a$ first. This is an example of a computation that reaches a state after a reversal that cannot be reached by computing forwards only.

**Example 2.3** *A long-running transaction consists of many atomic steps which are represented here by* $a$. *A step may succeed, and then it is followed by the next step (or success* $s$*; this action never fails), or fail which results in the action* $f$. *When all steps are successfully completed the transaction succeeds and is irreversible. When* $f$ *takes place all steps* $a$ *performed successfully need to be undone. The transaction is modelled by* $T_0$ *as follows:*

$$T_i \stackrel{\text{df}}{=} a[v_{i+1}].T_{i+1} + f[u] \quad \textit{for } 0 \le i < n, \qquad T_n \stackrel{\text{df}}{=} s$$

*The required controller is* $C \stackrel{\text{df}}{=} a[v_1].(a[v_n] + f[u].\underline{a}[v_1].C) + f[u].\underline{f}[u].C$. *Let us see how* $T_0\langle C\rangle$ *computes. If the transaction fails immediately by performing* $f[1,u]$*, then this triggers the outermost action* $f$ *in the controller:*

$$T_0\langle C\rangle \stackrel{f[1,u]}{\rightarrow} (a[v_1].T_1 + f[1,u])\ \langle \underline{f}[1,u].C\rangle$$

*The controller then requires undoing* $f$*:* $\stackrel{f[1,u]}{\rightsquigarrow} (a[v_1].T_1 + f[u].\mathbf{0})\langle C\rangle \equiv T_0\langle C\rangle$.

*If the transaction does not fail immediately, then* $a[1,v_1]$ *is performed (and is matched by the controller):*

$$T_0\langle C\rangle \stackrel{a[1,v_1]}{\rightarrow} (a[1,v_1].T_1 + f[u])\ \langle a[v_n] + f[u].\underline{a}[1,v_1].C\rangle$$

*The process then computes until the last step* $a[v_n]$*, or else it fails in the meantime by performing* $f[k,u]$*, for some key* $k$*. This is matched by the controller which becomes* $\underline{a}[1,v_1].C$*. Next, the execution is reversed until* $a[1,v_1]$ *is undone, thus returning to the original configuration:* $\cdots \stackrel{a[1,v_1]}{\rightsquigarrow} T_0\langle C\rangle$.

*In some transactions it may not be necessary to undo all successful steps* $a$ *in case of failure. If these steps can be grouped into sequences, then only the steps of the most recently performed sequence need undoing. Let there be two such sequences, the first finishing with* $a_k$ *with* $2 \leq k$ *and* $k+2 \leq n$*. Then the controller* $D$ *is defined as follows:*

$$\begin{aligned} D &\stackrel{\text{df}}{=} a[v_1].(a[v_k].D' + f[u].\underline{a}[v_1].D) + f[u].\underline{f}[u].D \\ D' &\stackrel{\text{df}}{=} a[v_{k+1}].(a[v_n] + f[u].\underline{a}[v_{k+1}].D') + f[u].\underline{f}[u].D' \end{aligned}$$

*We easily can check that* $T_0\langle D\rangle$ *works properly.*

**Example 2.4** *Assume a long-running transaction has a compensation* $K$ *which is triggered by action* $c$ *and which completes with* $s$*, where both* $c$ *and* $s$ *never fail. We model this by adjusting the definition of* $T_1$ *from Example 2.3 and leaving other* $T_i$*s unchanged:* $T_1 \stackrel{\text{df}}{=} a[v_1].T_2 + f[u] + c.K$*. The controller is* $C \stackrel{\text{df}}{=} a[v_1].(a[v_n] + f[u].\underline{a}[v_1].c.s) + f[u].\underline{f}[u].c.s$*. When a failure occurs the controller reverses all actions* $a$ *that took place so far (or just the initial* $f$*), triggers* $c$ *and insists that the compensation* $K$ *computes forwards by demanding* $s$*.*

**Example 2.5** *Consider the following system taken from [10], where (undo* $a$*) forces reversing of computation until* $a$ *is undone (similarly for (undo* $b$*)).*

$$(\overline{a}\,|\,a.\overline{d}\,|\,c.(\textit{undo } a)\ \ |\ \ \overline{b}\,|\,b.\overline{c}\,|\,d.(\textit{undo } b))\backslash\{a,d,c,b\}$$

*Inspecting the causal dependencies between actions, we note that undoing* $a$ *is possible only after* $c, \overline{c}$ *have communicated, which requires* $b, \overline{b}$ *to communicate first. And, of course, after* $a, \overline{a}$ *have happened. If in the meantime a communication on* $d, \overline{d}$ *takes place, it disables undoing* $a$*. This is because* $a$ *causes* $\overline{d}$ *and the cause* $a$ *cannot be undone prior to undoing the effect* $d$*. Causal dependencies*

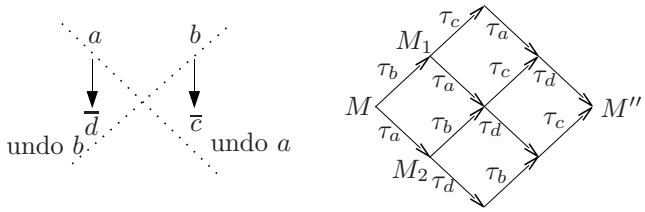


Figure 3: Example 2.5.

*between these communications are shown in the left-hand diagram in Figure 3, where $b \to \overline{c}$ means that a communication involving $b$ must precede a communication involving $\overline{c}$. The dashed line labelled 'undo a' indicates that reversing the communication involving $a$ is possible only after the communications involving the actions that appear above the line (here $a, b, \overline{c}$) have taken place.*

*In our calculus, the left component is $\overline{a} \,|\, (a.\overline{d} \,|\, c)\langle(c.\underline{a}.\underline{e}) \,|\, e\rangle$. Since $e$ cannot happen, the controller $(c.\underline{a}.\underline{e}) \,|\, e$ requires that 'after a forward $c$ reverse $a$ and keep reversing, or independently compute forward'. Since all actions are distinct there is no need here for key identifiers. The system is*

$$M \stackrel{\text{df}}{=} (\overline{a} \,|\, (a.\overline{d} \,|\, c)\langle(c.\underline{a}.\underline{e}) \,|\, e\rangle \;|\; \overline{b} \,|\, (b.\overline{c} \,|\, d)\langle(d.\underline{b}.\underline{e}) \,|\, e\rangle) \setminus \{a, d, c, b\}.$$

*In order to make transitions representing communications more readable we shall decorate labels $\tau$ with action labels: we shall write, for example, $\tau_b[1]$ instead of $\tau[1]$ for the communication on $b, \overline{b}$ with key 1. We shall also omit restriction. A communication on $b, \overline{b}$ leads to*

$$M \stackrel{\tau_b[1]}{\to} \overline{a} \,|\, (a.\overline{d} \,|\, c)\langle(c.\underline{a}.\underline{e}) \,|\, e\rangle \;|\; \overline{b}[1] \,|\, (b[1].\overline{c} \,|\, d)\langle(d.\underline{b}[1].\underline{e}) \,|\, e\rangle \equiv M_1$$

*Then we perform communications involving $a$ and then $c$:*

$$M_1 \stackrel{\tau_a[2]\tau_c[3]}{\to} \overline{a}[2] \;|\; (a[2].\overline{d} \,|\, c[3])\langle(\underline{a}[2].\underline{e}) \,|\, e\rangle \;|\; \overline{b}[1] \;|\; (b[1].\overline{c}[3] \,|\, d)\langle(d.\underline{b}[1].\underline{e}) \,|\, e\rangle.$$

*Next, a communication on $d, \overline{d}$ can take place or we can undo the communication involving $a$. Note that although the communication on $b$ took place, it cannot be undone at this point since the communication on $d$ has not taken place yet.*

*Consider a communication on $d, \overline{d}$:*

$$\stackrel{\tau_d[4]}{\to} \overline{a}[2] \;|\; (a[2].\overline{d}[4] \,|\, c[3])\langle(\underline{a}[2].\underline{e}) \,|\, e\rangle \;|\; \overline{b}[1] \;|\; (b[1].\overline{c}[3] \,|\, d[4])\langle(\underline{b}[1].\underline{e}) \,|\, e\rangle) \equiv M''$$

*The right-hand diagram in Figure 3 shows other sequences of communications involving $a, b, c, d$ from $M$ to $M''$. Controllers of $M''$ ask to undo $a$ and undo $b$. But, since both $d$ and $c$ have now taken place, $a$ and $b$ can be reversed only after reversing other actions. Overall, our mechanism for controlling execution works well with this example and its operational formulation is simpler than the formulation of the rollback construct [10].*

We finish this section with a remark on suitable behavioural equivalences and modal logics for our reversible calculus. A reverse interleaving bisimulation [15, 16, 18], which extends the standard bisimulation [13] with reverse transitions, seems a suitable behavioural equivalence. Also, a reverse pomset bisimulation may be very useful [18] as it talks directly about forward and reverse behaviour in terms of pomsets (partially ordered multisets) of actions. Event Identifier Logic [17], a modal logic with both forward and reverse modalities, is the appropriate logic for our calculus since it characterises the mentioned above equivalences and many safety properties, such as precedence and exception, are naturally expressible with reverse modalities.

# 3 The ERK Signalling Pathway

The ERK signalling pathway is a realistic example of computation that comprises forward and reverse steps where some of the reverse steps violate the causal ordering established by the forward steps. We show how the new execution control and prefixing with multisets of actions allow us to represent naturally this form of out-of-order reversible computation. Signalling pathways were modelled more fully by PEPA [3] and by rule-based languages BioNetGen [2] and Kappa [6]. We shall comment on the PEPA model below.

We shall now define prefixing with multisets of actions. The actions of a given multiset of actions can execute in any order, and the computation progresses to the next multiset of actions only if all of the actions from the first multiset have taken place. Process terms are extended with $(\alpha[v], s).P$ and $(\alpha[n, v], s).P$ where $s$ is a sequence of any actions or past actions. $s'$ is a typical sequence consisting entirely of past actions. For simplicity, we do not allow prefixing with multisets of actions in control terms. The SOS rules are as follows:

$$\frac{\mathsf{std}(X)}{(\alpha[v], s).X \overset{\alpha[n,v]}{\rightarrow} (\alpha[n, v], s).X} \qquad \frac{X \overset{\mu[n,v]}{\rightarrow} X' \quad \mathsf{fsh}[n](s')}{(s').X \overset{\mu[n,v]}{\rightarrow} (s').X'}$$

$$\frac{\mathsf{std}(X)}{(\alpha[n, v], s).X \overset{\alpha[n,v]}{\rightsquigarrow} (\alpha[v], s).X} \qquad \frac{X \overset{\mu[n,v]}{\rightsquigarrow} X' \quad \mathsf{fsh}[n](s')}{(s').X \overset{\mu[n,v]}{\rightsquigarrow} (s').X'}$$

The Ras/Raf-1/MEK/ERK signalling pathway (ERK pathway for short) delivers mitogenic and differentiation signals from the membrane of a cell to its nucleus. This pathway is regulated by the protein RKIP. We borrow the description of the pathway and its reactions from [5, 20].

The ERK pathway is spatially organised in such a way that a signal that arrives at the cell's membrane can be transmitted to the cell's nucleus via a cascade of reactions that involve proteins Ras, Raf-1, MEK and ERK. Initially, a G protein Ras is activated near a receptor on the cell's membrane. Ras then activates a kinase Raf-1 which becomes Raf*-1 (represented here by $F$). We shall not model Ras and its reactions here. Raf*-1 can then activate the MEK protein ($M$ here) which gets phosphorylated to become $pM$. Or, this binding

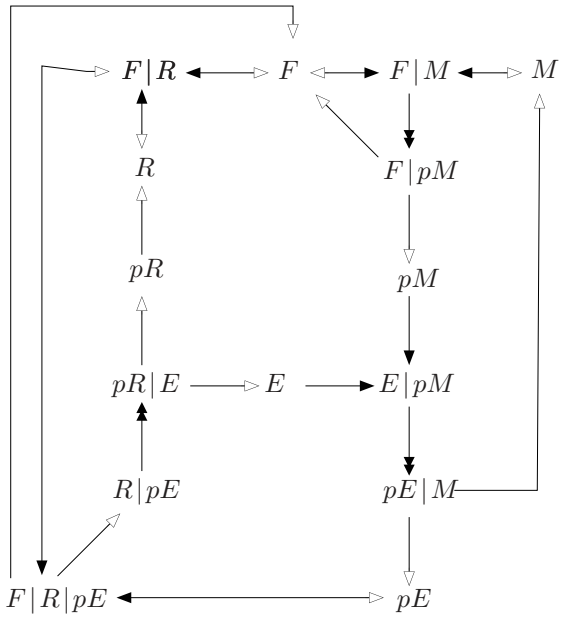


Figure 4: The ERK pathway.

of Raf*-1 to MEK can be inhibited by RKIP, which binds to Raf*-1; we shall return to this sequence of reactions below. The phosphorylated MEK ($pM$) then activates ERK protein ($E$ here) which, in turn becomes phosphorylated (represented by $pE$). Finally, at the end this cascade $pE$ can translocate to the nucleus and pass the signal. Or, it binds to RKIP thus deactivating it temporarily (see below).

When RKIP binds Raf*-1 and thus inhibits the activation of MEK, the resulting complex binds to a phosphorylated ERK ($pE$). Then the complex breaks releasing $F$, which can get involved in the cascade, $E$ and a phosphorylated $R$.

Figure 4 represents the described reactions. A black-headed arrow represents a reaction that binds two molecules into a complex molecule, an open-headed arrow represents a reaction that breaks a complex into its component molecules and a bi-directional arrow represents a pair of forwards/reverse reactions: a binding and unbinding. A two-headed arrow represents a reaction that involves phosphorylation/de-phosphorylation of its reactants. The nodes in the diagram are the molecules or complexes of molecules.

We note that the ERK pathway was previously modelled in the setting of the stochastic process algebra PEPA in [3]. There, the states of the pathway as in Figure 4 are represented as indivisible processes so, for example, $F \,|\, R \,|\, pE$ is represented by a single process and not as a composition of three separate processes. These processes perform forward actions that represent creation and breaking of bonds, and the system evolves from one state to another via multi-way syn-

chronisation of these actions. The transitions are timed and their durations are expressed as exponentially distributed random variables.

We represent the individual molecules of the ERK pathway as processes, for example $F, M, E$ and $R$, and the pathway is modelled by a composition of these processes. The reactions between the molecules are represented by forward and reverse synchronisations between processes. We define $F, M, E$ and $R$ as follows using the new multiset prefixing operator (key identifiers are not necessary here):

$$F \stackrel{\text{df}}{=} \overline{a}.F' \qquad M \stackrel{\text{df}}{=} (a, p, \overline{c}).M' \qquad E \stackrel{\text{df}}{=} (c, p, \overline{b}, \overline{n}).E' \qquad R \stackrel{\text{df}}{=} (a, b, p).R'$$

We also have molecules $P \stackrel{\text{df}}{=} \overline{p}.P'$ which represent phosphate groups that bind with $M, E$ and $R$ and phosphorylate them. These phosphorylated molecules are denoted by $pM, pE$ and $pR$ respectively. The ERK pathway is

$$(F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, pE) \setminus \{a, b, c, p\}$$

where we have a single copy of each molecule $F, M, E, P, R$ and $pE$. Next, we list those synchronisations between the processes of the pathway that represent valid reactions; there are many other synchronisations that have no bio-chemical meaning and we shall see later how controllers can be employed to prune them.

The molecule $F$ can bind with $M$ and start a signal cascade or it can bind with a copy of inhibitor $R$. In order to show how $F$ is released from the control of $R$ we have included a copy of a phosphorylated ERK ($pE$). (A more realistic model would be a composition of a large numbers of copies of $F, M, E, R, P, pM, pE$ and $pR$). These reactions start two alternative sequences of reactions, which we shall call the *cascade* and *regulation* sequences. We consider the cascade sequence first. For simplicity we omit restriction from now on. The binding of $F$ and $M$ is reversible; it is represented by blue arrows in Figure 4. The system evolves to

$$\stackrel{\tau_a[1]}{\rightarrow} \overline{a}[1].F' \;|(a[1], p, \overline{c}).M' \;|E \,|\, P \,|\, R \,|\, pE$$

where transition $\tau_a[1]$ indicates that a binding between $\overline{a}$ of $F$ and $a$ of $M$ took place and $\overline{a}$ and $a$ were marked with 1. Note that this binding can be immediately reversed.

$$\stackrel{\tau_a[1]}{\rightsquigarrow} \overline{a}.F' \,|\, (a, p, \overline{c}).M' \;|E \,|\, P \,|\, R \;\equiv F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, pE$$

$M$ gets phosphorylated and then releases $F$ by reversing the binding on $a$:

$$\stackrel{\tau_p[2]}{\rightarrow} \overline{a}[1].F' \,|\, (a[1], p[2], \overline{c}).M' \,|\, \overline{p}[2].P' \,|\, E \,|\, R \,|\, pE$$

$$\stackrel{\tau_a[1]}{\rightsquigarrow} \overline{a}.F' \,|\, (a, p[2], \overline{c}).M' \,|\, \overline{p}[2].P' \,|\, E \,|\, R \,|\, pE \equiv F \,|\, (a, p[2], \overline{c}).M' \,|\, \overline{p}[2].P' \,|\, E \,|\, R \,|\, pE$$

Then, $pM$ (which is $(a, p[2], \overline{c}).M' \,|\, \overline{p}[2].P'$) binds with $E$ and phosphorylates it;

$M$ is released and $pE$ is ready to convey the signal to the cell's nucleus:

$$\begin{aligned}
&\overset{\tau_c[3]}{\rightarrow} F \,|\, (a, p[2], \overline{c}[3]).M' \,|\, \overline{p}[2].P' \,|\, (c[3], p, \overline{b}, \overline{n}).E' \;\; | \, R \,|\, pE \\
&\overset{\tau_p[2]}{\rightsquigarrow} F \,|\, (a, p, \overline{c}[3]).M' \,|\, (c[3], p, \overline{b}, \overline{n}).E' \;\; | \, P \,|\, R \,|\, pE \\
&\overset{\tau_p[4]}{\rightarrow} F \,|\, (a, p, \overline{c}[3]).M' \,|\, (c[3], p[4], \overline{b}, \overline{n}).E' \,|\, \overline{p}[4].P' \,|\, R \,|\, pE \\
&\overset{\tau_c[3]}{\rightsquigarrow} F \,|\, (a, p, \overline{c}).M' \,|\, (c, p[4], \overline{b}, \overline{n}).E' \,|\, \overline{p}[4].P' \,|\, R \,|\, pE
\end{aligned}$$

The last process is $\equiv$ equivalent to $F \,|\, M \,|\, (c, p[4], \overline{b}, \overline{n}).E' \,|\, \overline{p}[4].P' \,|\, R \,|\, pE$ which is $\equiv$ equivalent to $F \,|\, M \,|\, pE \,|\, R \,|\, pE$. Now, the newly created $pE$ can communicate the signal with the nucleus via action $\overline{n}$ (we do not show this reaction). Note that there is now an extra copy of $pE$ created out of $E$ and $P$.

We return to the regulation sequence. We assume the binding in $pE$ has the key 8. Instead of combining with $M$, the protein $F$ can be inhibited by binding with $R$; this reaction is immediately reversible. Then the $R \,|\, F$ complex binds with $pE$. The system $F \mid M \mid E \mid P \mid R \mid pE \equiv R \mid F \mid pE \mid M \mid E \mid P$ evolves as follows:

$$\begin{aligned}
&\overset{\tau_a[5]}{\rightarrow} (a[5], b, p).R' \,|\, \overline{a}[5].F' \;\; | pE \,|\, M \,|\, E \,|\, P \\
&\overset{\tau_b[6]}{\rightarrow} (a[5], b[6], p).R' \,|\, \overline{a}[5].F' \,|\, (c, p[8], \overline{b}[6], \overline{n}).E' \,|\, \overline{p}[8].P' \,|\, M \,|\, E \,|\, P
\end{aligned}$$

Next, $F$ is released and then $pE$ phosphorylates $R$:

$$\begin{aligned}
&\overset{\tau_a[5]}{\rightsquigarrow} \equiv F \,|\, (a, b[6], p).R' \,|\, (c, p[8], \overline{b}[6], \overline{n}).E' \,|\, \overline{p}[8].P' \,|\, M \,|\, E \,|\, P \\
&\overset{\tau_p[8]}{\rightsquigarrow} F \,|\, (a, b[6], p).R' \,|\, (c, p, \overline{b}[6], \overline{n}).E' \,|\, \overline{p}.P' \,|\, M \,|\, E \,|\, P \\
&\overset{\tau_p[7]}{\rightarrow} F \,|\, (a, b[6], p[7]).R' \,|\, (c, p, \overline{b}[6], \overline{n}).E' \,|\, \overline{p}[7].P' \,|\, M \,|\, E \,|\, P
\end{aligned}$$

Finally, $E$ and $pR$ are disassociated and $R$ is de-phosphorylated:

$$\begin{aligned}
&\equiv \overset{\tau_b[6]}{\rightarrow} F \,|\, (a, b, p[7]).R' \,|\, \overline{p}[7].P' \,|\, (c, p, \overline{b}, \overline{n}).E' \,|\, M \,|\, E \,|\, P \\
&\overset{\tau_p[7]}{\rightsquigarrow} F \,|\, (a, b, p).R' \,|\, \overline{p}.P' \,|\, (c, p, \overline{b}, \overline{n}).E' \,|\, M \,|\, E \,|\, P \equiv F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, E \,|\, P
\end{aligned}$$

Note that this segment of the pathway deactivates $pE$ into $E$ and $P$.

A high level view of the behaviour of the ERK system $F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, pE$ is represented abstractly by the cascade and regulation sequences:

$$F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, E \,|\, P \;\; \begin{matrix} \leftarrow reg \leftarrow \\ \rightarrow cas \rightarrow \end{matrix} \;\; F \,|\, M \,|\, E \,|\, P \,|\, R \,|\, pE \;\; \begin{matrix} \rightarrow cas \rightarrow \\ \leftarrow reg \leftarrow \end{matrix} \;\; F \,|\, M \,|\, pE \,|\, R \,|\, pE$$

The cascade produces $pE$ which can signal the nucleus and the regulation sequence consumes $pE$ in order to stop $R$ regulating $F$.

$M, E$ and $R$ exhibit the following patterns of behaviour (putting aside the undoing of immediately reversible reactions on $a$ and $b$, and $\overline{n}$) which we write

with controller actions: $M : a.p.\underline{a}.\overline{c}.\underline{p}.\overline{\underline{c}}$; $E : c.p.\underline{c}.\overline{b}.\underline{p}.\overline{\underline{b}}$ and $R : a.b.\underline{a}.p.\underline{b}.\underline{p}$. We note the common pattern (modulo action names) and also behaviours that break causal dependencies: for example $\underline{a}$ happens before $\underline{p}$ in $M$ although $a$ causes $p$.

Finally, we define controller terms for the proteins $M, E$ and $R$ that will ensure that reactions follow the order of the cascade and regulation sequences.

$$C_M \stackrel{\text{df}}{=} a.C'_M \qquad C'_M \stackrel{\text{df}}{=} \underline{a}.C_M + p.\underline{a}.\overline{c}.\underline{p}.\overline{\underline{c}}.C_M$$

$$C_E \stackrel{\text{df}}{=} c.p.\underline{c}.C'_E \qquad C'_E \stackrel{\text{df}}{=} \overline{n}.N + \overline{b}.C''_E \qquad C''_E \stackrel{\text{df}}{=} \overline{\underline{b}}.C'_E + \underline{p}.\overline{\underline{b}}.C_E$$

$$C_R \stackrel{\text{df}}{=} a.C'_R \qquad C'_R \stackrel{\text{df}}{=} \underline{a}.C_R + b.C''_R \qquad C''_R \stackrel{\text{df}}{=} \underline{b}.C'_R + \underline{a}.p.\underline{b}.\underline{p}.C_R$$

**Claim 3.1** $(F \mid M\langle C_M\rangle \mid E\langle C_E\rangle \mid P \mid R\langle C_R\rangle \mid pE\langle C'_E\rangle) \setminus \{a, b, c, p\}$ *exhibits precisely the cascade and regulation reactions of* $(F \mid M \mid E \mid P \mid R \mid pE)\setminus\{a, b, c, p\}$.

# 4 Conclusion

We have presented a reversible process calculus with a new execution control operator and illustrated its usefulness and expressiveness with several examples, including long-running transactions with simple compensations and the ERK signalling pathway. The new operator allows us to model a variety of modes of reverse computation, ranging from strict backtracking to reversing which respects causal ordering of events, and even reversing which violates causal ordering. This last form of reversing has not been studied before and in our view it deserves further investigation. The execution control operator can also be used to encode irreversible actions, it can act as the restriction operator of CCS in contexts involving communicating processes, and it allows us to construct terms that reach a state after a reversal that cannot be reached by computing forwards only.


## Acknowledgements

We are grateful to the Reversible Computation 2012 referees and participants for their helpful comments and suggestions. The second author acknowledges partial support by EPSRC grant EP/G039550/1 and the Japan Society for the Promotion of Science (JSPS) grants S-09053 and FU-019.


# References


[1] G. Berry and G. Boudol. The chemical abstract machine. *Theoretical Computer Science*, 96(1):217–248, 1992.

[2] M.L. Blinov, J. Yang, J.R. Faeder, and W.S. Hlavacek. Graph theory for rule-based modeling of biochemical networks. In *T. Comp. Sys. Biology*, volume 4230 of *Lecture Notes in Computer Science*, pages 89–106. Springer, 2006.

[3] M. Calder, S. Gilmore, and J. Hillston. Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA. In *Transactions on Computational Systems Biology VII*, volume 4230 of *Lecture Notes in Computer Science*, pages 1–23. Springer, 2006.

[4] L. Cardelli and C. Laneve. Reversible structures. In *9th International Conference on Computational Methods in Systems Biology*, pages 131–140. ACM, 2011.

[5] K-H. Cho, S-Y. Shin, H.W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of RKIP on the ERK signaling pathway. In *CMSB*, volume 2602 of *Lecture Notes in Computer Science*, pages 127–141. Springer, 2003.

[6] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling of cellular signalling. In *Proceedings of the 18th International Conference on Concurrency Theory CONCUR 2007*, volume 4703 of *Lecture Notes in Computer Science*, pages 17–41. Springer, 2007.

[7] V. Danos and J. Krivine. Reversible communicating systems. In *Proceedings of the 15th International Conference on Concurrency Theory CONCUR 2004*, volume 3170 of *Lecture Notes in Computer Science*, pages 292–307. Springer, 2004.

[8] V. Danos and J. Krivine. Transactions in RCCS. In *Proceedings of the 16th International Conference on Concurrency Theory CONCUR 2005*, volume 3653 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 2005.

[9] V. Danos and J. Krivine. Formal molecular biology done in CCS-R. In *Proceedings of the 1st Workshop on Concurrent Models in Molecular Biology BioConcur 2003*, volume 180 of *ENTCS*, pages 31–49, 2007.

[10] I. Lanese, C.A. Mezzina, A. Schmitt, and J-B. Stefani. Controlling reversibility in higher-order pi. In *Proceedings of the 22nd International Conference on Concurrency Theory CONCUR 2011*, volume 6901 of *Lecture Notes in Computer Science*, pages 297–311. Springer, 2011.

[11] I. Lanese, C.A. Mezzina, and J-B. Stefani. Reversing higher-order pi. In *Proceedings of the 21st International Conference on Concurrency Theory CONCUR 2010*, volume 6269 of *Lecture Notes in Computer Science*, pages 478–493. Springer, 2010.

[12] I. Lanese, C.A. Mezzina, and J-B. Stefani. Controlled reversibility and compensations. In *Proceedings of Reversible Computation 2012*, volume 7581 of *Lecture Notes in Computer Science*. Springer, 2012.

[13] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[14] M. Mousavi, I.C.C. Phillips, M.A. Reniers, and I. Ulidowski. Semantics and expressiveness of Ordered SOS. *Information and Computation*, 207(2):85–119, 2009.

[15] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. In *Proceedings of 9th International Conference on Foundations of Software Science and Computation Structures, FOSSACS 2006*, volume 3921 of *LNCS*, pages 246–260. Springer, 2006.

[16] I.C.C. Phillips and I. Ulidowski. Reversing algebraic process calculi. *Journal of Logic and Algebraic Programming*, 73:70–96, 2007.

[17] I.C.C. Phillips and I. Ulidowski. A logic with reverse modalities for history-preserving bisimulations. In *Proceedings 18th International Workshop on Expressiveness in Concurrency*, volume 64 of *EPTCS*, pages 104–118, 2011.

[18] I.C.C. Phillips and I. Ulidowski. A hierarchy of reverse bisimulations on stable configuration structures. *Mathematical Structures in Computer Science*, 22:333–372, 2012.

[19] I. Ulidowski and I.C.C. Phillips. Ordered SOS rules and process languages for branching and eager bisimulations. *Information and Computation*, 178(1):180–213, 2002.

[20] J. Vera, O. Rath, E. Balsa-Canto, J.R. Banga, W. Kolch, and O. Wolkenhauer. Investigating dynamics of inhibitory and feedback loops in ERK signalling using power-law models. *Molecular BioSystems*, 6:2174–2191, 2010.