Reversing Computations Modelled by Coloured Petri Nets

Kamila Barylska[1], Anna Gogolińska[1], Łukasz Mikulski[1], Anna Philippou[2], Marcin Piątkowski[1], Kyriaki Psara[2]

[1] Faculty of Mathematics and Computer Science
Nicolaus Copernicus University, 87-100 Toruń, Poland
{kamila.barylska,anna.gogolinska,lukasz.mikulski,marcin.piatkowski}@mat.umk.pl
[2] Department of Computer Science, University of Cyprus
{annap,kpsara01}@cs.ucy.ac.cy

Abstract. Reversible computation is an unconventional form of computing where any sequence of performed operations can be executed in reverse order at any point during computation. It has recently been attracting increasing attention as on the one hand it promises low-power computation and on the other hand it is inherent or of interest in a variety of applications. In this paper we propose a structural way of translating reversing Petri nets (RPNs), a formalism that embeds the three main forms of reversibility (backtracking, causal reversing and out-of-causal-order reversing), to Coloured Petri Nets (CPNs), an extension of traditional Petri Nets, where tokens carry data values. The translation into the CPN model uses additional places and transitions in order to capture the machinery employed in the RPN framework and demonstrates that the abstract model of RPNs, and thus the principles of reversible computation, can be emulated in CPNs. The transformation can be automated and utilized for the analysis of reversible systems using CPN Tools.

1 Introduction

Reversible computation is an emerging computational paradigm that allows computation to be executed in the backward direction as effortlessly as it can be executed in the standard forward direction. Physicist Rolf Landauer [10] was the first to argue that only irreversible computation generates heat spawning a strong line of research towards the creation of reversible logic gates and circuits. Besides, reversible computing attracts much interest for its potential in a growing number of applications ranging from biological processes, where computation may be carried out in forward or backward direction [16, 9], to the field of system reliability, where reversibility can be used as a means of recovering from failures [6, 11].

Various strategies for reversing computation have been identified to date, the most prominent being backtracking, causal reversing and out-of-causal-order reversing. These approaches have been studied within a variety of frameworks including process calculi and Petri nets. The main

challenge in these studies was to maintain the information needed to reverse computation, addressed by including "memories" of past behaviour.

The first reversible process calculus RCCS [5] is a causal-consistent reversible extension of CCS that uses memory stacks in order to keep track of past communications, further developed in [6, 7]. A general method for reversing process calculi was subsequently proposed in [15] with CCSK being a special instance of the methodology. This proposal introduces keys to bind synchronised actions together. Constructs for controlling reversibility were also proposed in reversible extensions of the π-calculus in [11, 12], where the authors rely on simple thread tags, which act as unique identifiers. Furthermore, reversible computation was studied within event structures in [20, 17], whereas a reversible computational calculus for modelling chemical systems, composed of signals and gates, was proposed in [3].

Petri nets (PNs) are a graphical mathematical language associated with a rich mathematical theory and a variety of tools. They have been used extensively for modelling and reasoning about a wide range of applications [19]. The first study of reversible computation within Petri nets was proposed in [1, 2]. In these works, the authors investigate the effects of adding *reversed* versions of selected transitions in a PN, where these transitions are obtained by reversing the directions of arcs connected to them. They then explore decidability problems regarding reachability and coverability in the resulting PNs.

In another line of work, [14] proposes a reversible approach to Petri nets which implements an explicit approach for modelling reversible computation that does not require the addition of transitions for modelling reverse execution but, instead, allows to execute transitions in both forward and backward direction. The proposed formalism, *reversing Petri nets* (RPNs), introduces machinery and associated operational semantics to tackle the challenges of the three main forms of reversibility. In particular, RPNs are acyclic Petri nets where tokens are persistent and are distinguished from each other by an identity which allows transitions to be reversed spontaneously in or out of causal order and a history function is employed to track which transitions have been executed and in which order. The model appears to be closely related to that of coloured Petri nets (CPNs) [8], a backward compatible extension of traditional Petri nets that allows tokens to have a data value/colour attached and hence be distinguishable.

Contribution. The objective of this paper is to study the precise relation between RPNs and CPNs, the main challenge being the presence of the history notion in RPNs. Specifically, histories in RPNs impose a form of

global control during transition execution that comes in contrast to the nature of Petri nets where transition enabledness and execution can be determined merely based on local conditions.

Our study of this relation has demonstrated that RPNs can be encoded into standard CPNs. We utilize bounded CPNs only, hence even if some inhibitors are indirectly present (used to encode some functions of CPN ML), the boundedness of a net assures that the state space is finite and all inhibitors can be dropped by a standard complementary net construction [2, 13]. As a result we conclude that the principles of reversible computation can be encoded directly in the traditional model. Note that inhibitor nets are in general Turing powerful and decision problems (including reachability) are for them undecidable. However, in the case of bounded inhibitor nets, which we are using, this is not an issue. The more typical challenges are related with the complexity and the cost of increasing (exponentially) the size of the net. Specifically, we propose a structural translation from RPNs to CPNs, where for each transition we consider both forward and backward instances. Furthermore, the translation relies on storing histories and causal dependencies of executed transition sequences in additional places. The translation has been tested on a number of examples from all three forms of reversible computation, where the CPN-tools [18] was employed to illustrate that the translations conform to the semantics of reversible computation. Note that in addition to the acyclicity assumption, we have also excluded forks from the present discussion in order to keep the presentation simpler, though we believe that both of these assumptions can be lifted at the cost of a more complex translation.

Paper organisation. In the next section we give an overview of reversing Petri nets followed by Section 3, which describes the operational semantics for backtracking, causal and out-of-causal-order reversibility. In Section 4 we recall the notion of coloured Petri nets, whereas the sections following describe machinery for encoding local history (Section 5), encoding of backtracking and causal reversing (Section 6) and encoding of out-of-causal-order reversing (Section 7). Section 8 concludes the paper.

A working example of CPN that utilizes all constructions for out-of-causal reversing is available: http://folco.mat.umk.pl/papers/reversing/.

2 Reversing Petri Nets

In this section we recall the notion of reversing Petri nets (RPNs) defined in [14] together with the basic definitions and the semantic relations for reversing computation in RPNs (see [14] for details).

RPNs support the main three categories of reversing computations. The first approach, the most restrictive one, called *backtracking*, allows reversing only the transition which was last executed in the ongoing computation. The second approach, less restrictive, called *causal reversing*, allows a transition to rollback only if all its effects, if any, have been undone beforehand. The last form of undoing computation is the concept of *out-of-causal-order reversing*, where any previously executed transition can be undone. To capture this latter form of reversibility requires close monitoring of token manipulation within a net. Indeed, undoing an arbitrary transition during the execution of a Petri net calls for a clear enunciation of each transition's effect and undoing such an effect involves distinguishing each token on a net along with its causal path, i.e. the places and transitions it has traversed before reaching its current state. This requirement comes in conflict with transitions which might consume multiple tokens but release only a subset or even a set of new tokens. To resolve this, a reversing Petri net is built on the basis of a set of bases or simply tokens that correspond to the basic entities that occur in a system. These tokens are persistent, cannot be consumed, and can be combined together as the effect of transitions via so-called *bonds* into coalitions (also called *molecules*) that record the computational history of each token. This approach is similar to reaction systems from biochemistry but can be applied to a wide range of systems that feature reversible behaviour. Based on this intuition, reversing Petri nets are defined as follows:

Definition 1. A *reversing Petri net* is a tuple (P, T, F, A, B) where:

1. P and T are finite sets of *places* and *transitions*, respectively.
2. A is a finite set of *bases* or *tokens*. The set $\overline{A} = \{\overline{a} \mid a \in A\}$ contains "negative" instances and we write $\mathcal{A} = A \cup \overline{A}$.
3. $B \subseteq \{\{a, b\} \mid a \neq b \in A\}$ is a set of *bonds*. We use the notation $a-b$ for a bond $\{a, b\} \in B$. The set $\overline{B} = \{\overline{\beta} \mid \beta \in B\}$ contains a "negative" instance for each bond in B, $\mathcal{B} = B \cup \overline{B}$.
4. $F : (P \times T \cup T \times P) \to 2^{\mathcal{A} \cup \mathcal{B}}$ is a set of directed *arcs* labelled by a subset of $\mathcal{A} \cup \mathcal{B}$.

In the definition of a reversing Petri net, the sets of *places* and *transitions* are understood in a standard way (see [19]). For a label $l = F(p, t)$

or $l = F(t,p)$ we assume that each token appears in l at most once (either a or $\overline{a}$), if a bond $\{a,b\} \in l$ then $a,b \in l$ and for $l = F(t,p)$ we have $l \cap \overline{A} = \emptyset$ and $l \cap \overline{B} = \emptyset$.

For transition $t \in T$ we introduce ${}^\bullet t = \{p \in P \mid F(p,t) \neq \emptyset\}$, $t^\bullet = \{p \in P \mid F(t,p) \neq \emptyset\}$ (sets of input and output places of t), and $\mathsf{pre}(t) = \bigcup_{p \in P} F(p,t)$, $\mathsf{post}(t) = \bigcup_{p \in P} F(t,p)$ (unions of labels of the incoming/outgoing arcs of t), as well as $\mathsf{effect}(t) = \mathsf{post}(t) \setminus \mathsf{pre}(t)$.

The following restrictions give rise to the notion of well-formed RPNs.

Definition 2. A reversing Petri net is *well-formed*, if it satisfies the following conditions for all $t \in T$:

1. $A \cap \mathsf{pre}(t) = A \cap \mathsf{post}(t)$,
2. if $a{-}b \in \mathsf{pre}(t)$ then $a{-}b \in \mathsf{post}(t)$,
3. for every $t \in T$ we have: ${}^\bullet t \neq \emptyset$ and $|t^\bullet| = 1$,
4. if $a,b \in F(p,t)$ and $\beta = a{-}b \in F(t,q)$ then $(\beta \in F(p,t) \vee \overline{\beta} \in F(p,t))$.

Clause (1) indicates that transitions do not erase tokens and clause (2) indicates that transitions do not destroy bonds. In (3) forks are prohibited in order to avoid duplicating tokens that fork into different outgoing places but are already bonded in the incoming places. Finally, clause (4) indicates that tokens/bonds cannot be cloned into more than one outgoing places.

Remark 1. Note that the above definition differs from Definition 2 of [14], in that it additionally imposes condition *3.* that every transition has at least one incoming place and exactly one outgoing place. However, conflict situations and concurrency might still occur.

A marking is a distribution of tokens and bonds across places, $M : P \to 2^{A \cup B}$, where for $p \in P$ if $a{-}b \in M(p)$ then $a,b \in M(p)$. A *history* assigns a memory to each transition, $H : T \to \mathbb{N}$. The value 0 associated with a transition $t \in T$ means that t has not been executed yet or it has been reversed and not executed again, while a value $k > 0$ indicates that t was the k^{th} transition executed in the computation (and not reversed until this moment) H_0 denotes the initial history where $H_0(t) = 0$ for every $t \in T$. A *state* is a pair $\langle M, H \rangle$ of a marking and history.

For $a \in A$ and $C \subseteq A \cup B$ we capture the set of tokens connected with a via bonds as $\mathsf{con}(a,C) = (\{a\} \cap C) \cup \{b,c,\{b,c\} \mid \exists_w\ \mathsf{path}\{a,w,C\},\{b,c\} \in w\}$, where $\mathsf{path}\{a,w,C\}$ if $w = \langle \beta_1, \ldots, \beta_n \rangle$, and for all $1 \leq i \leq n$ we have $\beta_i = \{a_{i-1}, a_i\} \in C \cap B$, $a_i \in C \cap A$, and $a_0 = a$.

As seen in Figure 1, reversing Petri nets admit a natural graphical representation with places indicated by circles and transitions by boxes

connected to each other via labelled arcs. Tokens are indicated by $\bullet$ and bonds by lines between relevant tokens. A history is represented over the respective transition t as $[m]$, where $m = H(t)$, and is omitted when $m = 0$.

From now on we assume RPNs to be well-formed and acyclic (in the meaning of graph theory). Furthermore, as in [14], we assume that in the initial marking of a RPN, M_0, there exists exactly one base of each type, i.e. $|\{x \mid a \in M_0(x)\}| = 1$, for all $a \in A$. Now we can indicate the conditions that must be met for a transition of a RPN to be enabled.

Definition 3. Consider a reversing Petri net (P, T, F, A, B), a transition $t \in T$, and a state $\langle M, H\rangle$. We say that t is *(forward) enabled* in $\langle M, H\rangle$ if the following hold:

1. if $a \in F(p,t)$, resp. $\beta \in F(p,t)$, for $p \in {}^\bullet t$, then $a \in M(p)$, resp. $\beta \in M(p)$,
2. if $\overline{a} \in F(p,t)$, resp. $\overline{\beta} \in F(p,t)$ for $p \in {}^\bullet t$, then $a \notin M(p)$, resp. $\beta \notin M(p)$,
3. if $\beta \in F(t,p)$ for $p \in t^\bullet$ and $\beta \in M(q)$ for $q \in {}^\bullet t$ then $\beta \in F(q,t)$.

A transition t is enabled in a state $\langle M, H\rangle$ if all tokens from $F(p,t)$ for every $p \in {}^\bullet t$ (i.e. tokens required by the action for execution) are available, and none of the tokens whose absence is required exists in an incoming place of the transition (clauses *1.* and *2.*). Clause *3* indicates that if a pre-existing bond appears in an outgoing arc of a transition then it is also a precondition for the transition to fire. This enables us to define the effect of a transition as $\mathsf{effect}(t) = \mathsf{post}(t) - \mathsf{pre}(t)$.

Remark 2. Note that the above definition contains one less clause than the associated definition from [14] (definition 3 ibidem). This clause has become obsolete by the adoption of Clause *4.* in the definition of well-formedness of the present paper, forbidding forks in RPNs.

Definition 4. Given a reversing Petri net (P, T, F, A, B), a state $\langle M, H\rangle$, and a transition t enabled in $\langle M, H\rangle$, we write $\langle M, H\rangle \xrightarrow{t} \langle M', H'\rangle$ where:

$$M'(p) = \begin{cases} M(p) - \bigcup_{a \in F(p,t)} \mathsf{con}(a, M(p)), & \text{if } p \in {}^\bullet t \\ M(p) \cup F(t,p) \cup \bigcup_{a \in F(t,p), q \in {}^\bullet t} \mathsf{con}(a, M(q)), & \text{if } p \in t^\bullet \\ M(p), & \text{otherwise} \end{cases}$$

and $H'(t') = \max\{k | k = H(t''), t'' \in T\} + 1$ if $t' = t$, and $H(t')$ otherwise.

After the execution of transition t, all suitable (according to Definition 4) tokens and bonds occurring in its incoming arcs are transferred

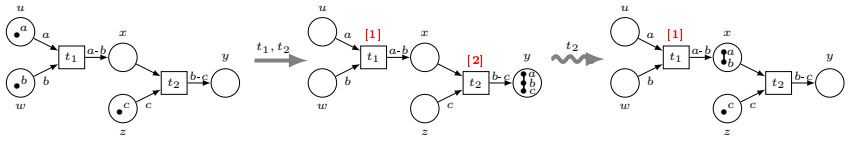

Fig. 1. Forward and backtracking execution

from the input places to the output places of t. Moreover, the history function H is changed by assigning the next available integer number to the transition. An example of forward transitions can be seen in the first two steps of Figure 1 where transitions t_1 and t_2 take place with the histories of the two transitions becoming [1] and [2], respectively. We also note that the tokens have been transferred from the incoming places of t_1 to the outgoing places of t_2 whilst creating a bond between $a{-}b{-}c$

3 Forms of reversibility – semantics

We now present the semantics for the three forms of reversibility as proposed in [14].

3.1 Backtracking

A transition is backward enabled if the following holds:

Definition 5. Consider a reversing Petri net $N = (P, T, F, A, B)$, a state $\langle M, H\rangle$ and a transition $t \in T$. We say that t is *bt-enabled* in $\langle M, H\rangle$ if $H(t) = k > 0$ with $k \geq k'$ for all $k' \in \mathbb{N}$, $k' = H(t')$ and $t' \in T$.

Thus, only the last executed transition can be backward executed in this semantics. The effect of backtracking a transition in a reversing Petri net is as follows:

Definition 6. Given a RPN $N = (P, T, F, A, B)$, a state $\langle M, H\rangle$, and a transition t *bt*-enabled in $\langle M, H\rangle$, we write $\langle M, H\rangle \overset{t}{\rightsquigarrow}_b \langle M', H'\rangle$ where:

$$M'(p) = \begin{cases} M(p) \cup \bigcup_{a \in F(p,t) \cap F(t,q)} \mathsf{con}(a, M(q) - \mathsf{effect}(t)), & \text{if } p \in {}^\bullet t \\ M(p) - \bigcup_{a \in F(t,p)} \mathsf{con}(a, M(p)), & \text{if } p \in t^\bullet \\ M(p), & \text{otherwise} \end{cases}$$

and $H'(t') = 0$ if $t' = t$, and $H(t)$ otherwise.

After the reversal of $t \in T$ all tokens and bonds from $F(t,p)$ for $p \in t^\bullet$, as well as their connected components, are removed, and transferred to places from $F(p,t)$. Note that in the final step of Figure 1, the bonds created by the transition are broken. Moreover the history function H of t is changed to 0 to capture that the transition has been reversed.

3.2 Causal Reversing

Reversal enabledness in causal reversing semantics is defined as follows.

Definition 7. Consider RPN $N = (P, T, F, A, B)$, a state $\langle M, H\rangle$ and a transition $t \in T$. Then t is *co-enabled* in $\langle M, H\rangle$ if $H(t) > 0$, and, for all $a \in F(t,p)$, if $a \in M(q)$ for some q and $\mathsf{con}(a, M(q)) \cap \mathsf{pre}(t') \neq \emptyset$ for some $t' \in T$ then either $H(t') = 0$ or $H(t') \leq H(t)$.

Note that the causal reversing of a transition $t \in T$ is allowed if all transitions executed causally after t have been reversed. The effect of causally reversing a transition in a reversing Petri net is as follows:

Definition 8. Given a RPN $N = (P, T, F, A, B)$, a state $\langle M, H\rangle$, and a transition t *co*-enabled in $\langle M, H\rangle$, we write $\langle M, H\rangle \overset{t}{\rightsquigarrow}_c \langle M', H'\rangle$ for M' and H' as in Definition 6.

We reverse transitions in causal reversing semantics in the same manner as in the backtracking semantics (Definition 6).

3.3 Out-of-causal-order Reversing

We are now ready to define out-of-causal-order reversing enabledness.

Definition 9. Consider a RPN $N = (A, P, B, T, F)$, a state $\langle M, H\rangle$ and a transition $t \in T$. We say that t is *o-enabled* in $\langle M, H\rangle$, if $H(t) > 0$.

According to the above definition, in this setting, every transition which has been executed can be reversed. The following notion helps to define the last executed transition manipulating a given set of tokens, where we write $\perp$ to express that the value is undefined.

Definition 10. Given a RPN N, an initial marking M_0, a current marking M, a history H, and a set of bases and bonds C we write:

$$\mathsf{last}(C,H) = \begin{cases} t, & \text{if } \exists_t\ \mathsf{post}(t) \cap C \neq \emptyset,\ H(t) > 0, \\ & \quad \forall_{t'}\ \mathsf{post}(t') \cap C \neq \emptyset, H(t') > 0, H(t') \leq H(t) \\ \perp, & \text{otherwise} \end{cases}$$

The effect of reversing a transition in out-of-causal order is as follows:

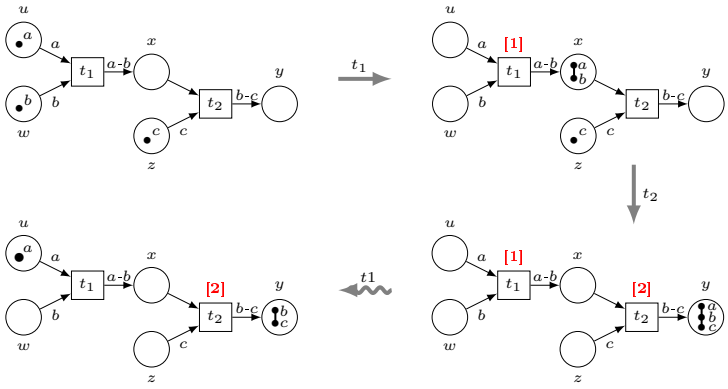

Fig. 2. Forward and out-of-causal-order execution

Definition 11. Given a RPN $N = (A, P, B, T, F)$, an initial marking M_0, a state $\langle M, H \rangle$ and a transition t that is o-enabled in $\langle M, H \rangle$, we write $\langle M, H \rangle \overset{t}{\rightsquigarrow}_o \langle M', H' \rangle$ where H' is defined as in Definition 6 and:

$$\begin{aligned} M'(p) = M(p) - \mathsf{effect}(t) - \{C_{a,p} \mid \exists_{a \in M(p)} p \in t'^{\bullet}, t' \neq \mathsf{last}(C_{a,p}, H')\} \\ \cup \{C_{a,q} \mid \exists_{a,q} a \in M(q), \mathsf{last}(C_{a,q}), H) = t', F(t', p) \cap \mathsf{last}(C_{a,q}) \neq 0\} \\ \cup \{C_{a,q} \mid \exists_{a,y} a \in M(q) \mathsf{last}(C_{a,q}), H') = \bot, C_{a,q} \subseteq M_0(p)\}, \end{aligned}$$

where $C_{a,p} = \mathsf{con}(a, M(p) - \mathsf{effect}(t))$.

As a result of out-of-causal-order reversing, a transition $t \in T$ results in breaking all bonds produced by t. For this reason in Figure 2 we can see that it is possible for tokens to move backwards in the net. If the destruction of a bond divides a component into smaller connected components, then those components should be relocated. They should be transferred to the places in which they would have been located if transition t had never been executed, which could imply the places in which they occurred in the initial marking.

Note that out-of-causal-order reversing semantics may create new states that were formerly unreachable by forward-only execution. For example consider the process of catalysis from biochemistry where a catalyst a helps the otherwise inactive molecules b and c to bond. Initially, a bonds with b which then enables the bonding between b and c. Next (see final step of Figure 2), the catalyst is no longer needed and its bond to the

other two molecules is released in an out-of-causal manner which creates the formerly unreachable state containing a bond between (only) b and c.

4 Coloured Petri Nets

Recall that RPNs constitute a model in which transitions can be reversed according to three semantics: backtracking, causal, and out-of-causal-order reversing. A main characteristic of RPNs is the concept of a *history*, which assigns natural numbers to transitions. In general, this mechanism may lead to the use of arbitrarily large numbers, thus requiring an infinite memory, hence resulting in an unbounded model. At the same time, it imposes the need of a global control in order to reverse computation. Our goal is to recast the model of [14] into one without the need for a potentially infinite history while establishing the expressiveness relation between RPNs and the model of bounded coloured Petri nets. In this section we recall the notion of coloured Petri nets and we propose a model that performs transition reversal according to the same semantics, but needing only local memory to perform.

Definition 12 ([8]). A (non-hierarchical) *coloured Petri net* is a nine-tuple $CPN = (P, T, D, \Sigma, V, C, G, E, I)$, where:

- P and T are finite, disjoint sets of *places* and *transitions*;
- $D \subseteq P \times T \cup T \times P$ is a set of *directed arcs*;
- Σ is a finite set of non-empty *colour sets*;
- V is a finite set of *typed variables* such that $Type(V) \in \Sigma$ for all $v \in V$;
- $C : P \to \Sigma$ is a *colour set function* that assigns colour sets to places;
- $G : T \to EXPR_V$ is a *guard function* that assigns a guard to each transition t such that $Type[G(t)] = Bool$;
- $E : D \to EXPR_V$ is an *arc expression function* that assigns an arc expression to each arc $d \in D$ such that $Type[E(d)] = \mathbb{N}^{C(p)}$, where p is the place connected with the arc d;
- $I : P \to EXPR_\emptyset$ is an *initialisation function* that assigns an initialisation expression to take each place p such that $Type[I(p)] = \mathbb{N}^{C(p)}$.

Note that, according to the utilised CPN Tools [4], $EXPR_V$ is the set of *net inscriptions* (over a set of variables V, possibly empty, i.e. using only constant values) provided by CPN ML. Moreover, by $Type[e]$ we denote the type of values obtained by the evaluation of expression e. The set of *free variables* in an expression e is denoted by $Var[e]$, while the type of v by $Type[v]$. The setting of a particular value to free variable v is called a

binding $b(v)$. We require that $b(v) \in Type[v]$ and denote with the use of $\langle\rangle$ filled by the list of valuations and written next to the element to whom it relates. The set of bindings of t is denoted by $B(t)$. The *binding element* is a transition t together with a valuation $b(t)$ of all the free variables related to t. We denote it by (t, b), for $t \in T$ and $b \in B(t)$.

A *marking* M in coloured Petri nets is a function which assigns to each $p \in P$ a multiset of tokens $M(p) \in \mathbb{N}^{C(p)}$. An initial marking is denoted by M_0 and defined for each $p \in P$ as follows: $M_0(p) = I(p)\langle\rangle$.

A binding element (t, b) is enabled at a marking M if $G(t)\langle b\rangle$ is true and at each place $p \in P$ there are enough tokens in M to fulfil the evaluation of the arc expression function $E(p, t)\langle b\rangle$. The resulting marking is obtained by removing from $M(p)$ the tokens given by $E(p, t)\langle b\rangle$ and adding those given by $E(t, p)\langle b\rangle$ for each $p \in P$.

In the following sections we show how to utilize coloured Petri nets to reverse transitions in accordance with the three considered semantics: backtracking, causal reversing and out-of-causal-order reversing. In fact, we indicate how to obtain CPNs behaving analogously to RPNs.

5 Transformation – local history

In this section we embark on constructing a procedure to transform an acyclic RPN, as presented in Definition 1, to an equivalent CPN. As a first step we consider the history function H, which we transform into a set of additional places. Specifically, we introduce one place for every transition and one place for every pair of transitions. This way, for each pair of transitions in a path π we are able to reconstruct the history of the pair and using it we may reconstruct the whole history of RPN computations. The proposed mechanism is local and distributed over the Petri net. Furthermore, it is universal for all types of reversing, with the assumption that for *causal-order reversing* some places are not used. This is explained in detail at the end of this section.

We construct the transformation of a RPN $P_R = (P_R, T_R, F_R, A_R, B_R)$ to CPN, namely the new coloured Petri net $C_R = (P_C, T_C, D_C, \Sigma_C, V_C, C_C, G_C, E_C, I_C)$ as follows.

The set of places P_C contains all elements from P_R, *transition history places* assigned to each transition and *connection history places* assigned to each pair of transitions: $P_C = P_R \cup \{h_i \mid t_i \in T_R\} \cup \{h_{ij} \mid t_i, t_j \in T_R, i < j\}$. The set of transitions of the net C_R is the same as in a RPN, namely $T_C = T_R$. New arcs have to be added to C_R to connect newly added places. Each transition is connected with its history place - as an input and an

output place. Every transition t_i is also connected with every connection history place h_{ij} (or h_{ji} depending on the order of i and j) with two arcs in opposite directions (they are at the same time input and output places of the transition), where j is a number of transition, different from i.

$$\begin{aligned} D_C = Domain(F_R) \cup \{(t_i, h_i) \mid t_i \in T_R\} \cup \{(h_i, t_i) \mid t_i \in T_R\} \\ \cup \{(t_i, h_{ij}) \mid t_i \in T_R, i < j\} \cup \{(t_i, h_{ji}) \mid t_i \in T_R, j < i\} \\ \cup \{(h_{ij}, t_i) \mid t_i \in T_R, i < j\} \cup \{(h_{ji}, t_i) \mid t_i \in T_R, j < i\}. \end{aligned}$$

Bases in CPNs are the basic entities in a system which can also be considered to be the atoms in a biochemical system. The set of colours Σ_C contains $Base = A_R$; $Bond = B_R$; $Bases$ (lists of distinct bases – we can treat them as subsets of $Base$); $Bonds$ (lists of distinct bonds – we can treat them as subsets of $Bond$); $Molecule = Bases \times Bonds$; $HIST = \{(n, i, j) \mid i, j, n \in \mathbb{N}\}$ (local history for a pair of transitions) and Int (natural numbers). Here molecules, as in a biochemical system, are considered to be a set of bases or atoms with the corresponding bond between them.

The set of variables V_C should contain all elements necessary to describe each input token of a transition.

The colour function C_C assigns a *molecule* to every $p \in P_R$, a natural number to every $h_{ij} \in P_C$, and a history $HIST$ to every $h_i \in P_C$.

Markings of connection-history places are numbers, which describe how many times transitions from the pair were executed. Namely, $M(h_{ij}) = n$ means that transitions t_i and t_j were executed n times – however the marking does not specify how many times an individual transition was executed. The markings of every transition history place h_i are sets of triples (n, j, i), where i and j are the numbers of two distinct transitions and t_i was n^{th} in a sequence of executions of t_i and t_j.

The guard function G_C has to be equivalent to the labels of input arcs defined in the reversing Petri net P_R. Consequently, if $a \in F_R(p, t_i)$ ($\beta \in F_R(p, t_i)$, respectively) for a transition t_i and its input place p, then $G_C(t_i)$ should contain a condition, assuring that the bonding of an input token for place p contains a (β, respectively).

The arc expression function E_C for arcs between transitions $t_i \in T_R$ and places $p_i \in P_R$ should be analogous to $F_R(t_i, p_i)$. If t_i only transfers tokens then $E_C(t_i, p_i)$ should be a union of bonds and bases of all inputs for t_i. If t_i creates a bond β, then $E_C(t_i, p_i)$ should be a union of bonds and bases of all inputs for t_i, together with the newly created bond β.

Places h_{ij} and h_i control the history. The arc expression function is defined as: $E_C(h_{ij}, t_i) = history_{ij}$ ($E_C(h_{ij}, t_j) = history_{ij}$), $E_C(t_i, h_{ij}) =$

$history_{ij} + 1$ ($E_C(t_j, h_{ij}) = history_{ij} + 1$), where $history_{ij} \in V_C$ and it represents the value obtained from place h_{ij} by t_i (t_j). Hence, the marking of the connection history place h_{ij} denotes the next history value for the pair of transitions t_i and t_j. For the transition history place h_i, the following arc expressions should be assigned: $E_C(h_i, t_i) = list_i$, where $list_i$ is a list of triples, which describes the previous history of the transition t_i and $E_C(h_i, t_i) = list_i \cup \{(history_{ij}, j, i) \mid t_j \in T_R, M(h_{ij}) = history_{ij}\}$. Since we assumed that each path π in a RPN is finite (as the net is acyclic), values in places h_i and h_{ij} are bounded.

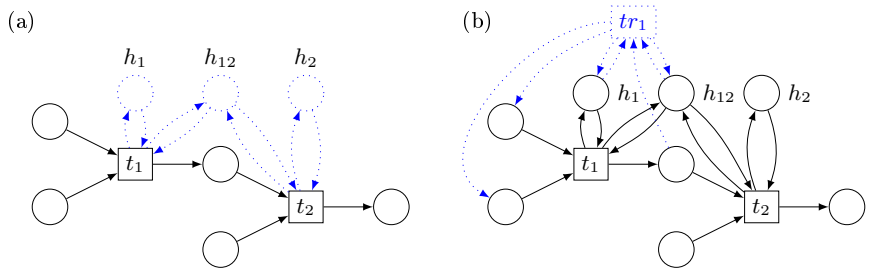

Fig. 3. Adding history places to a CPN – the basic construction for a CPN containing transition history places h_i, h_j and connected history place $h_{i,j}$ (a). Introducing a reverse transition of t_i (b). Construction appropriate for backtracking and causal-order reversing (see Examples 1 and 2).

Example 1. Let us look at Figure 3(a). The transition t_i is connected by a self-loop with its history place h_i, as well as t_j is connected by a self-loop with its history place h_j. Moreover, both of them are connected with their connection history place h_{ij} (assuming $i < j$). The newly created places and connections are depicted with the use of blue dashed arcs.

Transformation – causal-order reversing. The construction described to this point requires a refinement for causal-order reversing. In the case of causal-order reversing, history places are created only for pairs of dependent transitions, not for every possible transition pair.

Definition 13. Transitions $t_i, t_j \in T_R$ are dependent (we use the notation: $(t_i, t_j) \in Dep$) if an input place of one of them is an output place of the other: $(t_i, t_j) \in Dep \Rightarrow (t_i^\bullet \cap {}^\bullet t_j \neq \emptyset)$ *or* $({}^\bullet t_i \cap t_j^\bullet \neq \emptyset)$.

6 Transformation – adding reverses: backtracking, causal-order

The coloured Petri net C_R described in Section 5 is prepared for reversing. This can be achieved by adding supplementary reversal transitions. The new CPN $C_R' = (P_C, T_C', D_C', \Sigma_C, V_C', C_C, G_C', E_C', I_C)$ is based on C_R. The set of places P_C and colours Σ_C, the function C_C and the initialization expression I_C are the same as in C_R.

For every transition in C_R, a new reversal transition tr is added to the net. Hence $T_C' = T_C \cup \{tr_i \mid t_i \in T_R\}$. The execution of tr_i is equivalent to a rollback of an execution of t_i corresponding to tr_i. Each transition tr_i is connected to the same set of places as $t_i \in T_C$ but in opposite directions, namely $D_C' = D_C \cup \{(tr_i, p) \mid (p, t_i) \in D_C\} \cup \{(p, tr_i) \mid (t_i, p) \in D_C\}$.

The set of variables V_C' should contain all elements necessary to describe each input token of all transitions (also reversal transitions).

The guard function G_C' has to be modified to take into account the newly created reversal transitions. The conditions used in the guard function for transitions tr_i should guarantee that the transition t_i was the last one executed in a system (for backtracking) or no other dependent transition was executed after t_i (for causal-order reversing). All necessary information about an execution of t_i can be obtained from the history place h_i assigned to t_i and connection history places h_{ij} and h_{ji}.

The arc expression function E_C' for arcs between transitions tr_i and places $p_i \in P_R$ should describe the reversal of the execution of t_i. Hence, if t_i is just a transfer transition, then the arc description should contain only the transfer of molecules. If t_i creates a bond $\beta = a-b$ then during the execution of tr_i the bond should be broken. This results in the production of two separate molecules, one of them including a while the other one including b. Hence $E_C(p_i, tr_i)$ should contain only transfer of a molecule, and $E_C(tr_i, p_j)$ should contain transfer of a molecule obtained after breaking bond $\beta = a-b$, which includes a (b, respectively) if a (b, respectively) has been transferred from the place p_j during execution of t_i. It can be done using the CPN semantics in combination with the use of functions, allowed in CPN ML.

The arc expression function for the pair (h_i, tr_i) (i.e. $E_C(h_i, tr_i)$) should assign only transfer of the t_i execution history. For arcs in the opposite direction the arc expression function should return the history token to h_i with the last execution of t_i excluded. Be aware that for acyclic RPNs the last execution of a transition is the only execution of that transition. For connection history places h_{ij} (h_{ji} respectively) values in those places

should be decreased by one during tr_i execution, and this operation should be described by the arc expression function.

Example 2. Figure 3 depicts a net containing the newly created history places (a), as well as the same net with the addition of the reverse tr_i of the transition t (b). As one can see, the reverse transition is connected to exactly the same places as t, but the arrows turn in the opposite direction. The newly created part is depicted in dashed blue.

7 Transformation - adding reverses: out-of-causal-order

In the previous section constructions for backtracking and causal-order reversing have been presented. Below we provide a similar construction for out-of-causal-order reversing. Based on C_R, let us define another CPN $C_R'' = (P_C, T_C'', D_C'', \Sigma_C'', V_C'', C_C'', G_C'', E_C'', I_C'')$. Only the set P_C remains untouched, all other components of the CPN C_R are adjusted.

A new colour *smol* – set of molecules (represented as a list) is added to Σ_C''. Hence $\Sigma_C'' = \Sigma_C \cup \{smol\}$. C_C'' assigns *smol* to places $p \in P_R$. For places from $P_C \setminus P_R$, functions C_C'' and C_C are the same. The change is caused by the need to distinguish between an empty place and a place with an empty molecule, represented by the empty set. In some cases the empty list of molecules has to be transferred to or from a place. We need extra arcs from and to those places, in directions opposite to the original ones (the arcs are depicted with red dashed lines in Figure 4(a) since the colour set function C_C'' for places of the original RPN has changed (see below).

For every transition t_i in C_R, a new reversal transition tr_i is added to the net. The execution of tr_i is equivalent to the rollback of the execution of transition t_i. Hence $T_C'' = T_C \cup \{tr_i \mid t_i \in T_C\}$.

As in the previous case, each transition tr_i is connected in opposite direction to all places connected with t_i. However, we also need to add connections which allow to place accurately all molecules obtained during breaking a bond that has been created by t_i (or reversing a transfer). For every $t_i \in T_C$ we define a set of atoms $Atoms_{t_i}$ which contains all bases related to t_i, such that $a \in Atoms_{t_i}$ if a is present in the set of arc function values $E_C({}^\bullet t_i, t_i)$ (recall that $a \in E_C(t_i, q) \Rightarrow a \in E_C({}^\bullet t_i, t_i)$). Then, for every $t_i \in T_C$ we define a set of places TP_{t_i} (*Transition's Places*) as follows: $q \in TP_{t_i}$ if some base from $Atoms_{t_i}$ is present in q during some computation of the net. It is possible to define the TP_{t_i} set using the structure of RPNs. The reversal transition tr_i is connected in both directions with every place from TP_{t_i}.

Transition tr_i is also connected in both directions with every transition history place and every connection history place. Those places are necessary to give the reversal transition access to the complete history of the RPN computations. Since C_C'' is different than in C_R, additional arcs should be added from every transition $t_i \in T_R$ to all its input places. This is required to maintain correctly a token consisting of a list of molecules. If a transition consumes a molecule from its input place, it has to transfer the whole list and then put back the list without the respective molecule.

The similar goes for its output places. Summarising, in C_R'' every transition $t \in T_R$ is connected with its original input and output places with two arcs in opposite directions.

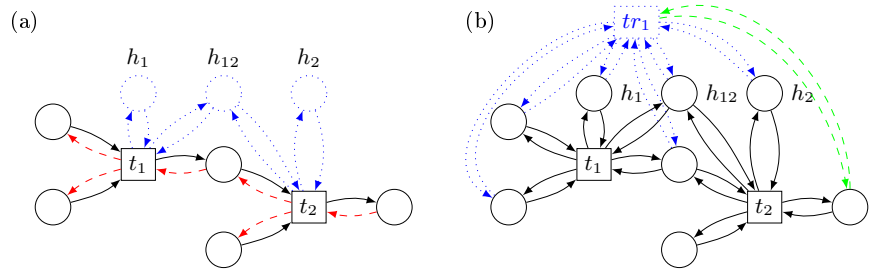

Fig. 4. Adding a reverse tr_1 to transition t_1. Construction appropriate for out-of-causal-order reversing (see Example 3).

Example 3. Consider the nets depicted in Figure 4. Part (a) presents a net with additional history places and connections (dotted blue), as well as the additional arcs from t_1 and t_2 to their input places, and arcs directed to t_1 and t_2 from their output places (dashed red). Part (b) shows the net obtained in (a) (black), with the addition of tr_1 being a reverse of t_1 and all needed connections from and to tr_1. Note that the connections marked in dashed green need to be added according to the definition of the set TP_{t_1}.

The set of variables V_C'' should contain all elements necessary to describe each input token of every transition (also the reversal ones). The guard function G_C'' has to be modified to take into account reversal transitions. The conditions used for transitions tr_i should guarantee that the transition t_i has been executed before reversing – its history cannot be empty, hence its transition history place h_i cannot contain an empty list.

During out-of-causal reversing we have to analyse a set of transitions which use a particular base. Hence, for every base $a \in A$, we define a set of transitions $AT_a = \{t \in T_C \mid \exists_{q \in P_R} a \in E_C(q,t)\}$ (*Atom's Transitions*). Therefore AT_a contains transitions which require base a to be executed, i.e. base a is included in the description of at least one input arc of every transition from AT_a.

The function E_C'' requires the largest adjustment. It has to describe the whole procedure of reversing. If transition t_i creates a bond $a{-}b$ then we need to find a molecule Z, which contains base a. By the construction of RPN, only one base of each type can be present. The molecule has to be found among places from TP_{t_i} due to the construction of TP_{t_i}. For every $p{\in}TP_{t_i}$, the arc expressions $E_C''(p, tr_i)$ must be defined to transfer a token from p. If the molecule transferred from p does not contain a, then it is moved back to p – a suitable expression should be added to $E_C''(tr_i, p)$. If the molecule contains a, then it is the sought molecule Z. The following step is to determine a set of bases from the found molecule $C{=}Z \setminus \{a, b\}$, connected to a after breaking the bond $a{-}b$ – recall that it is denoted by $\mathsf{con}(a, C)$. Subsequently the last transition executed in the considered computation among transitions from $\bigcup_{c\in\mathsf{con}(a,C)} AT_c \setminus \{t_i\}$ has to be determined (if the set $\bigcup_{c\in\mathsf{con}(a,C)} AT_c \setminus \{t_i\}$ is not empty). This can be done using a total order represented in a distributed way as markings of transition history places and connection history places. Later on we take the maximal element from the ordered set $\bigcup_{c\in\mathsf{con}(a,C)} AT_c \setminus \{t_i\}$ and denote it as t_a. We need also to take into account the output place of t_a (from P_R) and we denote it as p_a (notice that every transition from T_C has only one output place from P_R). If $\bigcup_{c\in\mathsf{con}(a,C)} AT_c \setminus \{t_i\}$ is empty we take as p_a the initial place for base a (a place where a was present in the initial marking). The molecule $\mathsf{con}(a, C)$ should be moved into p_a. If $b \notin \mathsf{con}(a, C)$ then we repeat the same procedure for b ($\mathsf{con}(b, C)$ from Z and t_b should be determined and $\mathsf{con}(b, C)$ should be placed into p_b). The case when t_i only transfers base a is almost the same as the previous case, but $\mathsf{con}(a, Z)$ is equivalent to the whole molecule which contains a, hence similar procedure can be used. Although it seems complicated, all those operations can be expressed using CPN semantics with the use of graph operations and included in $E_C''(tr_i, p)$, where $p \in P_R$.

One more thing has to be explained in details. Transition tr_i transfers $\mathsf{con}(a, Z)$ to p_a, but how can we be sure that transition tr_i is connected to p_a? We know that tr_i is connected with every place from TP_{t_i} and the following reasoning justifies that $p_a \in TP_{t_i}$. We assume that transition t_i creates a bond $a{-}b$ or transfers a base a, and it is reversed. Note that $a \in Atoms_{t_i}$ – according to the definition of $Atoms_{t_i}$. If p_a is the initial place for base a then naturally $p_a \in TP_{t_i}$. Let us assume that p_a is an output place of transition t_a.

If $\exists_{q\in P_R} a \in E_C(q, t_a)$ then transition t_a requires base a to be executed and it cannot consume any base (according to definition of RPNs), hence

it has to put base a into its output place p_a. According to the construction of TP_{t_i}, base a can be in place p_a, hence $p_a \in TP_{t_i}$.

Now let us consider the case when $\forall_{q \in P_R} a \notin E_C''(q, t_a)$. Since $t_a \in \bigcup_{c \in \mathsf{con}(a,C)} AT_c \setminus \{t_i\}$, it must be that $\exists_{d \in \mathsf{con}(a,C)} d \in E_C''(q, t_a)$. The molecule in C_R'' can be seen as a graph, where vertices represent bases and edges – bonds. Because $d \in \mathsf{con}(a, C)$ there exists a path e between a and d in molecule $\mathsf{con}(a, C)$. Let us assume that transition t_k has created the last bond α in e (it has added the last edge to the path). If transition t_a was executed before t_k (according to the total order), then t_a cannot be the maximal in the set $\bigcup_{c \in \mathsf{con}(a,C)} AT_c \setminus \{t_i\}$, because $t_k \in \bigcup_{c \in \mathsf{con}(a,C)} AT_c \setminus \{t_i\}$ (t_k has created α between some bases in $\mathsf{con}(a, C)$). If transition t_a was executed after t_k, then path e between a and d was present in the molecule, when transition t_a was executed. During its execution, transition t_a transfers base d because $d \in E_C''(q, t_a)$ (it may also transfer it and create a bond between d and other base). Hence, in the considered execution, it had to transfer also base a to p_a (because a and d are connected). Since base a was present in p_a, according to the definition, $p_a \in TP_{t_i}$. If $t_a = t_k$, transition t_a has transferred both bases a and d to p_a to add the last bond present in path e, hence $p_a \in TP_{t_i}$.

For all connection history places h_{jk} the arc expression $E_C''(h_{jk}, tr_i)$ should include the relocation of a token from h_{jk}, because it is necessary to extract the total order of transitions' executions. If $i \neq j$ and $i \neq k$ then the token is not changed and put back into h_{jk} – it should be expressed by $E_C''(tr_i, h_{jk})$. If $i = j$ or $i = k$ then $E_C''(tr_i, h_{jk})$ has to contain an expression decreasing the value of the token from h_{jk} by 1 and transferring it to h_{jk}.

For the transition history place h_i related to transition t_i (the transition to be reversed) information about the last execution of the transition should be removed from the token and this operation should be described by $E_C''(tr_i, h_i)$ and $E_C(h_i, tr_i)$. Since the considered PN is acyclic, the last execution is the only one. Hence, after the execution of tr_i, h_i has to contain only an empty list. For the other transition history places h_j, $i \neq j$, the arc expression $E_C''(h_j, tr_i)$ should include transfer of a token from h_j. The expression $E_C''(tr_i, h_j)$ should contain the modification of the token value. Since the last execution of transition t_i has not yet been reversed, the triple (n, i, j) is for sure present in the token. We also need to find a corresponding triple (m, j, i) in the local history of transition t_i obtained from place h_i. If n is larger than m, the arc expression $E_C''(tr_i, h_j)$ should exchange the value of triple (n, i, j) by $(n - 1, i, j)$. No matter whether

the value of the token is modified or not, it needs to be transfer back to the place h_j.

8 Conclusions

In this paper we deal with two models of concurrent computations allowing action reversing. Namely, we provide a transformation from reversing Petri nets into coloured Petri nets. This way we enable backtracking, causal reversing and out-of-causal-order reversing with the use of the well known and widely investigated model of coloured Petri nets. The model can encode reversing Petri nets without the need of a history utilizing arbitrarily large values nor global control.

As future work we plan on implementing an algorithmic translation that transforms RPNs to CPNs in an automated manner using the transformation techniques discussed in this paper. Another goal is to investigate systems which allow cycles. Note that the addition of cycles can in fact be achieved within RPNs by adopting "stack histories" for each transition that record all previous occurrences of a transition that is inside a cycle and it remains to be seen how this type of histories can be encoded within the CPN model. We also aim to explore how our framework applies in fields outside computer science, since the expressive power and visual nature offered by Petri nets coupled with reversible computation has the potential of providing an attractive setting for analysing systems (for instance in biology, chemistry or hardware engineering).

References

1. K. Barylska, M. Koutny, Ł. Mikulski, and M. Piątkowski. Reversible computation vs. reversibility in Petri nets. *Science of Computer Programming*, 151:48–60, 2018.
2. K. Barylska, Ł. Mikulski, M. Piątkowski, M. Koutny, and E. Erofeev. Reversing transitions in bounded Petri nets. *Fundamenta Informaticae*, 157:341–357, 2018.
3. L. Cardelli and C. Laneve. Reversible structures. In *Proceedings of CMSB 2011*, pages 131–140. ACM, 2011.
4. CPN Tools project website, http://cpntools.org/.
5. V. Danos and J. Krivine. Reversible communicating systems. In *Proceedings of CONCUR 2004*, LNCS 3170, pages 292–307. Springer, 2004.
6. V. Danos and J. Krivine. Transactions in RCCS. In *Proceedings of CONCUR 2005*, LNCS 3653, pages 398–412. Springer, 2005.
7. V. Danos and J. Krivine. Formal molecular biology done in CCS-R. *Electronic Notes in Theoretical Computer Science*, 180(3):31–49, 2007.
8. K. Jensen and L. M. Kristensen. Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer, 2009.
9. S. Kuhn and I. Ulidowski. A calculus for local reversibility. In *Proceedings of RC 2016*, LNCS 9720, pages 20–35. Springer, 2016.

10. R. Landauer. Irreversibility and heat generation in the computing process. *IBM Journal of Research and Development*, 5(3):183–191, 1961.
11. I. Lanese, M. Lienhardt, C. A. Mezzina, A. Schmitt, and J. Stefani. Concurrent flexible reversibility. In *Proceedings of ESOP 2013*, LNCS 7792, pages 370–390. Springer, 2013.
12. I. Lanese, C. A. Mezzina, and J. Stefani. Reversibility in the higher-order π-calculus. *Theoretical Computer Science*, 625:25–84, 2016.
13. T. Murata. Petri nets: Properties, analysis and applications. In*Proceedings of the IEEE 77.4*, 541–580, 1989.
14. A. Philippou and K. Psara. Reversible computation in Petri nets. To appear in the *Proceedings of RC 2018*.
15. I. Phillips and I. Ulidowski. Reversing algebraic process calculi. In *Proceedings of FOSSACS 2006*, LNCS 3921, pages 246–260. Springer, 2016.
16. I. Phillips, I. Ulidowski, and S. Yuen. A reversible process calculus and the modelling of the ERK signalling pathway. In *Proceedings of RC 2012*, LNCS 7581, pages 218–232. Springer, 2012.
17. I. Phillips, I. Ulidowski, and S. Yuen. Modelling of bonding with processes and events. In *Proceedings of RC 2013*, LNCS 7947, pages 141–154. Springer, 2013.
18. A. V. Ratzer, L. Wells, H. M. Lassen, M. Laursen, J. F. Qvortrup, M. S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. CPN tools for editing, simulating, and analysing coloured Petri nets. In *Proceedings of ICATPN 2003*, LNCS 2679, pages 450–462. Springer, 2003.
19. W. Reisig. *Understanding Petri Nets - Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.
20. I. Ulidowski, I. Phillips, and S. Yuen. Concurrency and reversibility. In *Proceedings of RC 2014*, LNCS 8507, pages 1–14. Springer, 2014.