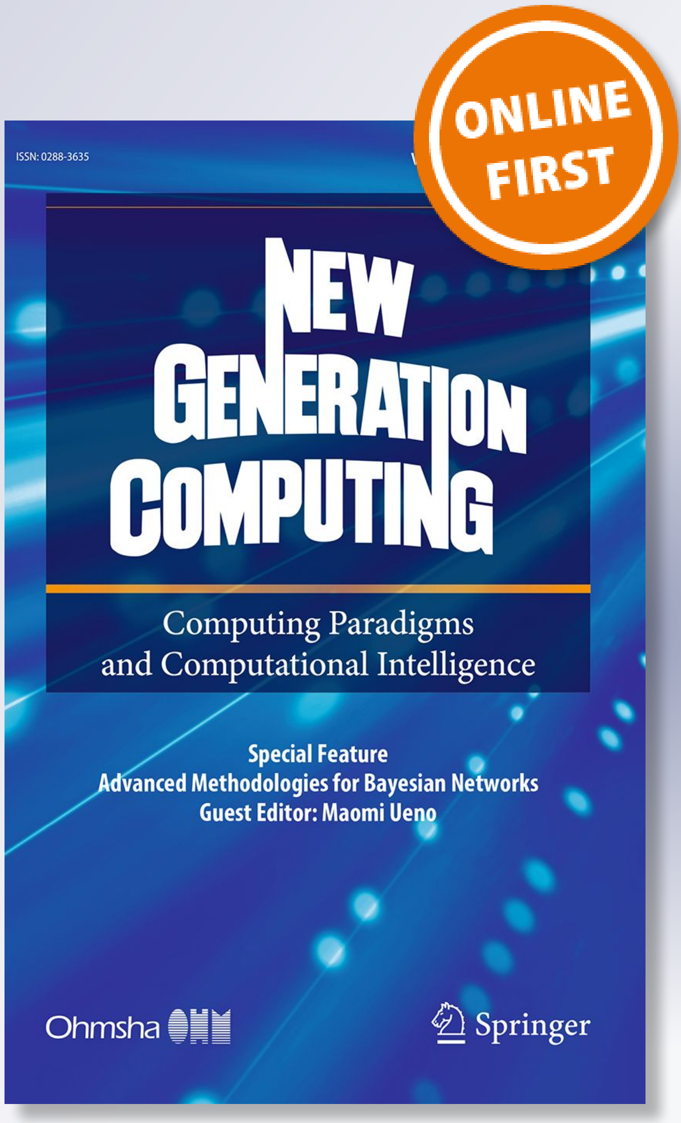

ISSN: 0288-3635
ONLINE FIRST
NEW GENERATION COMPUTING
Computing Paradigms and Computational Intelligence
Special Feature
Advanced Methodologies for Bayesian Networks
Guest Editor: Maomi Ueno
Ohmsha
Springer

New Generation Computing
https://doi.org/10.1007/s00354-018-0040-8

RESEARCH PAPER

Reversing Event Structures

Irek Ulidowski[1] · Iain Phillips[2] · Shoji Yuen[3]

Received: 5 March 2018 / Accepted: 25 July 2018

Abstract
Reversible computation has attracted increasing interest in recent years. In this paper, we show how to model reversibility in concurrent computation as realised abstractly in terms of event structures. Two different forms of event structures are considered, namely event structures defined by causation and prevention relations and event structures given by an enabling relation with prevention. We then show how to reverse the two kinds of event structures, and discuss causal as well as out-of-causal order reversibility.

Keywords Reversible computation · Configuration system · Reversible asymmetric event structure · Enabling with prevention relation

Introduction

Reversing computation of concurrent and distributed systems has many novel and promising applications and, since it is a relatively new area of research, it has as many technical and conceptual challenges. Several different styles of undoing of computation have been identified recently. *Backtracking* and reversing of computation that preserves *causal order* were considered in, for example, [7, 8, 12, 13, 24, 25, 28] with applications including recovery-oriented systems and reversible debugging. Reversing *out-of-causal order*, however, which is a very common mode of operation in biochemical systems, has not been studied very widely. The

✉ Irek Ulidowski
iu3@le.ac.uk

Iain Phillips
i.phillips@imperial.ac.uk

Shoji Yuen
yuen@i.nagoya-u.ac.jp

[1] Department of Informatics, University of Leicester, Leicester, UK

[2] Department of Computing, Imperial College London, London, UK

[3] Graduate School of Informatics, Nagoya University, Nagoya, Japan

first study of out-of-causal order reversibility was carried out by Phillips et al. [34] where an extension of the process calculus CCSK [28, 29], a reversible version of Milner's CCS, with the execution control operator was proposed. This was followed by a study of a form of reversible event structure [35] based on a generalisation of Winskel's enabling relation [38]. Phillips and Ulidowski proposed then in [31, 32] reversible event structures that focused on analysing conflict and causation as first-class notions in the setting of reversible computation. Both forms of reversible event structures were studied further in [36]. Subsequently, Aubert and Cristescu modelled a subcalculus of RCCS [10, 11], another reversible extension of Milner's CCS, in configuration structures [1]. Then Graversen, Phillips and Yoshida studied categories of reversible event structures in [16], including those mentioned above. Moreover, they developed a category of reversible bundle event structures in [17], and used the causal subcategory to model CCSK. They modified CCSK to control the reversibility with a rollback primitive, and gave semantics of CCSK with rollback by exploiting the capacity for non-causal reversibility of reversible bundle event structures. Join inverse categories were developed by Kaarsgaard et al. to model reversible recursion in reversible functional programming in [21].

The last decade has produced a good understanding of how causal reversibility can be described in the settings of operational semantics and process calculi, and how to model reversibility logically and in terms of behavioural equivalences. Research on reversing process calculi can be traced back perhaps to Berry and Boudol's Chemical Abstract Machine [6]. Danos and Krivine developed RCCS, a reversible version of CCS, in [10, 11], and Phillips and Ulidowski proposed a general method for reversing process calculi in [28, 29]. The reversible flowchart computation model developed by Yokoyama et al. [40] showed how to reverse low-level machine code for microprocessors as well as high-level block-structured languages. Reversible structures that compute forwards and backwards asynchronously were developed by Cardelli and Laneve [7]. Mechanisms for controlling reversibility based on a rollback construct were devised by Lanese et al. [23] for a reversible higher order π calculus [24]. An alternative mechanism based on the execution control operator was proposed in [34]. A mechanism for triggering reversal of actions inspired by covalent bonding in chemistry was developed by Kuhn and Ulidowski in [22]. Cristescu et al. gave a compositional semantics for the reversible pi-calculus [8], and further developed rigid families, a causal model based on configuration structures, for the reversible pi-calculus [9]. Event Identifier Logic (EIL), which extends Hennessy–Milner logic [18] with reverse modalities, was introduced in [33]. EIL corresponds to hereditary history-preserving bisimulation equivalence [5] within a particular true-concurrency model of stable configuration structures [14]. Moreover, natural sublogics of EIL correspond to coarser equivalences, several of them defined in terms of reversible events, sets of concurrent reversible events or pomsets of reversible events. These equivalences and other behavioural equivalences based in the reversible setting were studied for the first time in [30].

The aim of the paper is to explain how to model reversibility in concurrent computation as realised abstractly in terms of event structures. We adopt a descriptive and fairly informal style of presentation, with many examples and directions for further reading. In the next section, we introduce the notions of events, configurations, computation and configuration systems followed by which we present two different forms of event structures and show how to extend them with reversibility. We explain that Reversible Asymmetric Event Structures (RAESs) are easier to work with and are suitable for the modelling of many reversible systems which reverse either causally or out-of-causal order. We show that a version of the Reverse Diamond property holds for causal RAESs. However, not all concurrent systems can be modelled with RAESs, so we propose more expressive Reversible Event Structures (RESs). We prove that RAESs can be encoded into RESs. Since the sets of events computed by RESs can grow non-monotonically we define configurations of RESs as limits of infinite sequences of sets of events. We show that such configurations correspond to the traditional configurations when there are no reversible events. Moreover, we prove that the enabling relations of RESs are powerful enough that we do not need the traditional notion of the consistency relation. Numerous examples are used to illustrate our approach.

For the convenience of the reader we provide in the conclusion a table of all relations on events we use in this paper.

Events and Configurations

The behaviour of concurrent systems is represented abstractly by event structures where units of behaviour are modelled by *events*. We aim to cover many different systems, so the events will represent a wide range of activities such as, for example, receiving or sending a message, assigning a value to a variable, or creating a bond between two chemical compound. Events have names, denoted here abstractly as a, b, c, d, e, f, and no two different events share the same name. A concurrent system or process is then modelled as an *event structure* which is a set of events and a number of relations on events such as, for example, causality and conflict. Event structures were defined by Winskel [38] following earlier work by Nielsen, Plotkin and Winskel [26]. They were developed further in [15, 37, 39].

There are a number of ways to relate events. For example, an event *a causes* an event b, so when they both happen we know that a happened before b. Events can be *independent* from each other, or some events may be in *conflict* with other events. An alternative way to define how events relate to each other is via an *enabling* relation. In this paper, we shall present event structures defined by a number of relations, such as causality and precedence, as well as event structures given by an enabling relation.

Event structures compute or execute by either performing events or undoing previously performed events, thus moving from one state to another state. A state is a set of events called a *configuration* that have occurred and have not been undone yet. The act of moving from a configuration to another configuration is a computation step and is represented by a transition relation $C \rightarrow C'$. It means that configuration C evolves to configuration C' by doing and/or undoing some events. A sequence of computation steps is called an execution, or a computation.

Example 1 Consider a situation where two events a and b are to take place. Initially, no event has occurred yet. This is represented by the empty configuration $\emptyset$. Then the event a takes place, resulting in configuration $\{a\}$, and finally the event b occurs producing $\{a, b\}$. This computation is represented by $\emptyset \rightarrow \{a\} \rightarrow \{a, b\}$. Events can also be undone (reversed). We shall use the notation $\underline{a}$ and $\underline{b}$ to denote actions of undoing executed events a and b, respectively. We also call $\underline{a}$ and $\underline{b}$ reverse a and reverse b events, respectively. More generally, we take the view that undoing an event e means that e is removed from the current configuration, and it is as if e had never occurred. That is apart possibly from indirect effects, such as e having caused another event f before e was reversed. If e is putting a coin into a vending machine, then undoing e is withdrawing that coin. Returning to our example, performing $\underline{b}$ in $\{a, b\}$ regresses the computation to $\{a\}$: this is often written as $\{a, b\} \dashrightarrow \{a\}$ instead of $\{a, b\} \rightarrow \{a\}$ to indicate that a is undone and to match the notation used in our figures.

The computation of event structures, where we can also undo performed events, can be represented by a *configuration system*. Configuration systems are closely related to *configuration structures*, which have a notion of configuration and a notion of concurrent or *step* transition. These were introduced by van Glabbeek and Goltz in [14] and were later extended by van Glabbeek and Plotkin in [15].

Definition 1 Let $\mathcal{P}(E)$ denote the powerset of a set E. A *configuration structure* is a pair $\mathcal{C} = (E, \mathsf{C})$ where E is a set of events and $\mathsf{C} \subseteq \mathcal{P}(E)$ is a set of configurations. For configurations X, Y, we let $X \rightarrow Y$ if $X \subseteq Y$ and for every Z, if $X \subseteq Z \subseteq Y$ then Z is a configuration.

Let X and Y be configurations. Since all the events in $Y \setminus X$ are independent, they can happen concurrently as a single *step*. We sometimes write $X \xrightarrow{A} Y$ instead of $X \rightarrow Y$ where $A = Y \setminus X$. If $Y = X \cup \{a\}$ then $X \rightarrow Y$. This may no longer hold when we reverse events. Consider $E = \{a, b\}$ where a causes b, so that b cannot occur unless a has occurred previously. Then $\{b\}$ is not a possible configuration using forwards computation. However, if a is reversible, we can do a, written as $\emptyset \rightarrow \{a\}$, then we can follow with b, written as $\{a\} \rightarrow \{a, b\}$, and finally we reverse a, namely $\{a, b\} \dashrightarrow \{b\}$, and thus we reach $\{b\}$. Thus, both $\emptyset$ and $\{b\}$ are configurations, but we do not have $\emptyset \xrightarrow{b} \{b\}$.

The first definition of configuration systems suitable for reversing events was given in [31]. We let $A, B, X, Y, Z, \ldots$ range over sets of events. If an event e is reversible, we have a corresponding reverse event $\underline{e}$. We write $\underline{B}$ for $\{\underline{e} : e \in B\}$. Not all events need to be reversible, so we define the set $F \subseteq E$ of reversible events, and we insist that $\underline{F} \cap E = \emptyset$.

Definition 2 A *configuration system* is a quadruple $\mathcal{C} = (E, F, \mathsf{C}, \rightarrow)$ where E is a set of events, $F \subseteq E$ are the reversible events, $\mathsf{C} \subseteq \mathcal{P}(E)$ is the set of *configurations* and $\rightarrow \subseteq \mathsf{C} \times \mathcal{P}(E \cup \underline{F}) \times \mathsf{C}$ is a labelled transition relation such that if $X \xrightarrow{A \cup \underline{B}} Y$ then:

- $A \cap X = \emptyset$ and $B \subseteq X \cap F$ and $Y = (X \backslash B) \cup A$;
- $X \xrightarrow{A' \cup \underline{B'}} Z \xrightarrow{(A \backslash A') \cup (\underline{B \backslash B'})} Y$ (where $Z = (X \backslash B') \cup A' \in \mathsf{C}$) for every $A' \subseteq A$ and $B' \subseteq B$.

We say that $A \cup \underline{B}$ is *enabled* at X if there is Y such that $X \xrightarrow{A \cup \underline{B}} Y$. A transition $X \xrightarrow{A \cup \underline{B}} Y$ is *mixed* if both A and B are non-empty. If $B = \emptyset$ we say the transition is *forwards*, and if $A = \emptyset$ the transition is *reverse*.

In this paper, we do not discuss mixed transitions in great depth; they are treated fully in [32]. Most of our examples concern transitions where A or B (in $\underline{B}$) are singleton sets and where the other set is empty. As a result, the transitions denote either performing an event or undoing an event.

We also define reachable configurations. Let $C = (E, F, \mathsf{C}, \rightarrow)$ be a configuration system. We say that configuration X is a *reachable* configuration if $\emptyset \xrightarrow{A_1 \cup \underline{B}_1} \cdots \xrightarrow{A_n \cup \underline{B}_n} X$. We recall that we have $A_i \subseteq E$ and $B_i \subseteq F$ for each $i = 1, \ldots, n$ by Definition 2.

Reversible Event Structures with Causality and Precedence

In this section, we discuss event structures where the causation, concurrency and precedence relations on events determine how they compute.

To explore different forms of relations between events and how this impacts on performing and undoing of events, we shall mostly consider small event structures. The basis for our main examples is a setting with only three events a, b and c. Even in such a simple setting we will be able to represent the most important styles of executing events forwards and in reverse. The events a, b, c are depicted by the three dimensions of the cube in Fig. 1. Any of the four edges of any dimension denotes an *occurrence* of an appropriate event. The bottom-left vertex represents the empty configuration $\emptyset$ (the origin of computation). The top-right vertex represents the configuration where all the events have taken place, namely $\{a, b, c\}$. If there are no constraints on the events they can happen in any order, which is represented by taking the edges, starting from the origin of the cube, in any order. If events happen simultaneously then we take the appropriate diagonals. We do not display transitions of simultaneous events in our figures.

Causality is a binary irreflexive relation on events. It specifies which events (directly) cause other events. We write $a \prec b$ to mean that the event a causes the

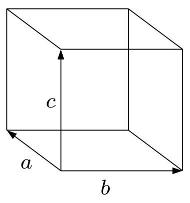

Fig. 1 A cube of three events a, b and c

event b, so b cannot happen before a has taken place. If $a \prec b$ and $b \prec c$ and $a \prec c$, so that $\prec$ is transitive, then we know that an execution of that event structure contains a if it contains b, and both a and b have occurred if c has happened. The maximal execution is $\emptyset \rightarrow \{a\} \rightarrow \{a,b\} \rightarrow \{a,b,c\}$. This is depicted in the left cube in Fig. 2 by the sequence of thick arrows. An alternative way to represent an execution is with a sequence of events that take place in the execution. For example, the sequence abc is the execution of the system with $a \prec b$, $b \prec c$ and $a \prec c$. We can also write $\emptyset \xrightarrow{a} \{a\} \xrightarrow{b} \{a,b\} \xrightarrow{c} \{a,b,c\}$ where we label transitions with the events that happen.

The cube on the right in Fig. 2 shows all possible executions when a, b and c are *independent*, except for those executions that involve steps (sets of simultaneous events) which we do not display for clarity. If events are not related by causality or other relations, then they are independent. Independent events can take place in any order; hence, six complete executions abc, acb, bac, bca, cab and cba are depicted in Fig. 2. Each square of thick arrows represents graphically the independence of the events. When the events can happen in any order we call them *concurrent* events. We also have step transitions here, for example, $\emptyset \rightarrow \{a,b,c\}$ and $\{a\} \rightarrow \{a,b,c\}$ but they are not displayed in Fig. 2. Finally, there are several mixed transitions here, for example, performing b and *undoing* a from $\{a\}$ is represented by $\{a\} \rightarrow \{b\}$, or $\{a\} \xrightarrow{b,\underline{a}} \{b\}$.

If $a \prec b$ and $a \prec c$, meaning that a causes both b and c, and if b, c are independent, then there are only two complete executions: abc and acb. Correspondingly, $a \prec c$ and $b \prec c$ (namely, c is caused by both a and b) produce executions abc and bac.

We have illustrated so far how causality and independence (concurrency) affects the execution. Another important relation on events is the *conflict* relation [38]: if a and b are in conflict, written as $a \sharp b$, then once either a or b occurs in a computation the other event cannot occur afterwards in the same computation. We shall consider carefully *precedence*, a form of *asymmetric conflict* discussed in detail in [2]. The notation $a \lhd b$, read as event a *precedes* event b, means that if both a and b have occurred then a has occurred first. Consequently, if only b occurred then a can no longer happen in the same computation; if it could it would not precede b. The precedence relation has a dual interpretation. We shall use $\rhd$, the symmetric version of the symbol $\lhd$, and write $b \rhd a$, which is read as b *prevents* a and means that if b is present in a configuration and a is not, then a can no longer occur. We have $a \lhd b$ if and only if $b \rhd a$. Moreover, $a \lhd b$ and $b \lhd a$ if and only if $a \sharp b$.

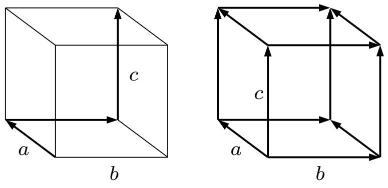

Fig. 2 Left cube: event a causes event b, which causes in turn event c. Right cube: events a, b and c are independent, so can happen in any order

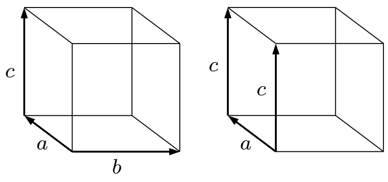

Fig. 3 Left cube: events a and b are in conflict, with a causing c. Right cube: a precedes c

If, additionally, $a \prec c$, then we have two complete executions ac and b depicted by the left cube in Fig. 3. If we only have events a and c, and we have $a \lhd c$ then ac and c are the only complete executions, which can be seen in the right cube in Fig. 3.

There are other forms of execution of the three events a, b, c which cannot be achieved by any combination of the causation, concurrency and precedence relations: we discuss this in the next section.

We are now ready to describe three forms of undoing events. We shall use our simple event structures to illustrate these forms. *Backtracking* is when events are undone in the inverse order they occurred. The event structure $a \prec b \prec c$ which has reached the configuration $\{a, b, c\}$ can backtrack by undoing c first, then undoing b and, lastly, undoing a. The left cube in Fig. 4 shows the system backtracking c and then b (dashed arrows pointing in the opposite direction) from $\{a, b, c\}$, which is written as $\{a, b, c\} \dashrightarrow \{a, b\} \dashrightarrow \{a\}$.

Consider the event structure $a \prec b$ and $a \prec c$. Here a occurs first and then b, c can occur independently. Once we have reached the configuration $\{a, b, c\}$ we cannot deduce which of b and c occurred last by just inspecting the configuration. Since the events are independent, the order of performing or undoing them does not matter. This form of reversing is *causal reversing* [10, 24, 29], also called causal-consistent reversing. Causal reversing is undoing of computation where

- events that cause other events can only be undone after the caused events are undone first, and
- independent events can be undone in any order irrespective of the order they have actually occurred.

We note that backtracking is a form of causal reversing. There are, however, numerous examples of undoing events apparently *out-of-causal order*. For example, this form of doing and undoing events plays a vital rôle in the mechanisms driving biochemical reactions as is argued in [22, 27].

Example 2 We now consider the behaviour shown in the right cube of Fig. 4. Events a, b and c can be interpreted as *chemical reactions* (or bonds) between catalysts

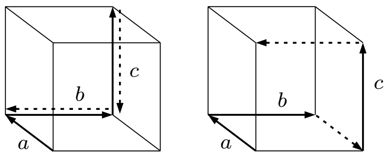

Fig. 4 Left cube: backtracking of events a, b and c. Right cube: a catalytic reaction is an example of the out-of-causal order reversibility

and other molecules. Initially, event a causes event b. Once we have b, a is not needed so it is undone. Finally, c occurs due to b being present and then c cannot be undone. Once c has occurred then b is undone. One can think of the event a as the *catalyst* of reaction b, and b as the catalyst of reaction c. The computation is $\emptyset \rightarrow \{a\} \rightarrow \{a,b\} \dashrightarrow \{b\} \rightarrow \{b,c\} \dashrightarrow \{c\}$. Note that we undo a, the cause of b, before we undo b. Overall, we can reach $\{c\}$ from $\emptyset$ via a combination of forwards and reverse moves but we cannot reach $\{c\}$ by executing forwards only.

To model appropriately different styles of undoing events we extended the causation and precedence relations in [31, 32] to specify additionally how undoing of events is related to performing events. Recall that $\underline{a}, \underline{b}, \underline{c}$ denote undoing of a, b, c. We have now the causation relation $\prec$ with pairs $x \prec \underline{y}$, meaning that the event y can be undone if y has occurred and x has occurred and has not been undone yet. For example, $a \prec \underline{a}$ means that a can be undone if it has occurred. Correspondingly, our precedence relation $\lhd$ can now have also pairs $\underline{x} \lhd y$ to represent that x cannot be undone if y is present.

Definition 3 A *reversible asymmetric event structure* (RAES) is a quadruple $\mathcal{E} = (E, F, \prec, \lhd)$ where E is a set of events and $F \subseteq E$ are those events of E which are reversible, and for any $a, b, c, e \in E$ and $\alpha \in E \cup \underline{F}$:

1. $\lhd \subseteq (E \cup \underline{F}) \times E$ is the *precedence* relation (with $a \lhd b$ if and only if $b \rhd a$), which is irreflexive;
2. $\prec \subseteq E \times (E \cup \underline{F})$ is the *direct causation* relation, which is irreflexive and well-founded, and such that$\{e \in E : e \prec \alpha\}$ is finite and $\lhd$ is acyclic on $\{e \in E : e \prec \alpha\}$;
3. $a \prec \underline{a}$ for all $a \in F$;
4. if $a \prec \alpha$ then not $a \rhd \alpha$;
5. $a \lll b$ implies $a \lhd b$, where *sustained direct causation* $a \lll b$ means that $a \prec b$ and if $a \in F$ then $b \rhd \underline{a}$;
6. $\lll$ is transitive;
7. if $a \sharp c$ and $a \lll b$ then $b \sharp c$, where $\sharp$ is defined to be $\lhd \cap \rhd$.

Well-foundedness of $\prec$ means that there are no infinite descending sequences with respect to $\prec$. We say that $\lhd$ is acyclic on a set S if the transitive closure of $\lhd$ on S is irreflexive.

We now look more closely at the notions of *causation* and sustained direct causation, and at *conflict inheritance* introduced in Definition 3.

Causation can be explained in two different ways. Event a causes event b, written as $a \prec b$, means either

1. in any execution (computation), if b occurs then a occurs earlier, or
2. if b is enabled at configuration X then we must have $a \in X$.

The two views are equivalent for the forwards-only computation when there is no reversing. Consider the events a, b and c with $a \prec b \prec c$. Taking view (1) above we deduce that $a \prec c$. View (2) also allows us to work out that $a \prec c$: this is because in the forwards-only computation configurations X are *left-closed*, namely for each $e \in X$ the set of events that cause e is a subset of X. Thus, causation is transitive when only computing forwards no matter which view is adopted.

In the setting where computation can be reversed view (2) of causation is simpler and we adopt it here. If all reversing is causal, meaning that we reverse events only after all the events they caused were reversed first, then all configurations are left-closed. So it is natural to require $\prec$ to be transitive. If, however, reversing is out-of-causal order, which leads to non-left-closed configurations (such as $\{b\}$, $\{b, c\}$ and $\{c\}$ in Examples 2 and 7), it is no longer reasonable to insist on $\prec$ being transitive. If $a \prec b \prec c$, as in Examples 2 and 7, then a may be reversed after b occurs, and before c occurs. As a result, it is appropriate to use a weaker notion of non-transitive direct causation in RAESs (see condition 2 in Definition 3). Moreover, we have introduced the concept of sustained direct causation (condition 5 in Definition 3), where $a \lll b$ means that a causes b and a cannot reverse until b reverses. This is the analogue of standard causation for forwards computation and we, therefore, take sustained causation to be transitive (condition 6 in Definition 3).

Next we comment on *conflict inheritance* in the reversible setting. Originally, conflict inheritance was defined in [38] as if $a \prec b$ and $a \sharp c$ then $b \sharp c$. If $a \prec b$ and $a \sharp c$ and a is reversible, then we can undo a in $\{a, b\}$ to reach $\{b\}$. Since a is not present in $\{b\}$ there is nothing to prevent c from occurring. This gives us the configuration $\{b, c\}$, which means that b and c are not in conflict. Hence, we do not have conflict inheritance with respect to $\prec$ in RAESs. However, we still have conflict inheritance with respect to sustained causation $a \lll b$ as required by condition 7 in Definition 3.

We now give a definition of the mixed transition relations for RAESs.

Definition 4 Let $\mathcal{E} = (E, F, \prec, \lhd)$ be an RAES. We define the associated configuration system $C(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ as follows. Let C consist of those $X \subseteq E$ such that $\lhd$ is well-founded on X. For $X \in \mathsf{C}$ and $A \subseteq E$, $B \subseteq F$, we define $X \xrightarrow{A \cup \underline{B}} Y$ if and only if $X, Y \in \mathsf{C}$ and $Y = (X \setminus B) \cup A$ and $A \cup \underline{B}$ is *enabled* at X, which is

- $A \cap X = \emptyset$, $B \subseteq X$;
- for every $a \in A$, if $c \prec a$ then $c \in X \setminus B$;
- for every $a \in A$, if $c \rhd a$ then $c \notin X \cup A$;
- for every $b \in B$, if $d \prec \underline{b}$ then $d \in X \setminus (B \setminus \{b\})$;
- for every $b \in B$, if $d \rhd \underline{b}$ then $d \notin X \cup A$.

We have introduced informally backtracking, causal reversing and out-of-causal order reversing. Next we discuss these styles of undoing of events in more detail and show how to model them in RAESs.

Example 3 Consider the behaviour shown in the left cube of Fig. 4. The forwards computation is specified by $a \prec b \prec c$ and $a \prec c$. To ensure that the events can be undone we insist that $x \prec \underline{x}$ for all $x \in \{a, b, c\}$. Note that the lack of $y \prec \underline{x}$ for all $y \neq x$ ensures that the events can be undone freely after they have taken place because there are no further prerequisites for undoing them. Finally, if we add $\underline{a} \lhd b$, and $\underline{b} \lhd c$, then we can achieve undoing of events in the backtracking order represented in Fig. 4. Note that only c can be undone initially in $\{a, b, c\}$ because $\underline{a} \lhd b$ and $\underline{b} \lhd c$ and the presence of b, c prevents undoing of a, b respectively.

In many computing situations, for example, in reversible debugging of concurrent programs, one does not need to undo computation in the backtracking order. We can undo independent events in any order provided that when an event is undone we have first undone all the events caused by the event. To achieve such a style of reversing we need to impose certain conditions on RAESs. Consider first the following condition:

1. causes can be only undone if their effects are not present, namely $x \prec y$ implies $\underline{x} \lhd y$ for all $x \in F, y \in E$.

The RAESs that satisfy condition (1) are called *cause-respecting*.

Example 4 Consider the system $a \prec b$, $a \prec c$. To obtain cause-respecting reversibility we extend $\prec$ and $\lhd$ as follows: $a \prec \underline{a}$, $b \prec \underline{b}$, $c \prec \underline{c}$ and $\underline{a} \lhd b$, $\underline{a} \lhd c$. Once $\{a, b, c\}$ is reached, b, c can be undone in any order. The conditions $\underline{a} \lhd b$, $\underline{a} \lhd c$ ensure that a can only be undone when b, c are not present, meaning after b, c have been reversed. Hence, the reverse executions are $\{a, b, c\} \dashrightarrow \{a, b\} \dashrightarrow \{a\}$ and $\{a, b, c\} \dashrightarrow \{a, c\} \dashrightarrow \{a\}$, and clearly $\{a\} \dashrightarrow \emptyset$.

A consequence of condition (1) is that reversing is cause-respecting in the sense that when we can undo an event e in a configuration X, it means that the events caused by e (the effects of e), if any, are not present in X:

Proposition 1 *Let $\mathcal{E}$ be a cause-respecting RAES, and let X be a configuration of $\mathcal{E}$. Then if $X \xrightarrow{\underline{B}} Y$ and $e \in B$, $x \in X$ then $e \not\prec x$.*

Proof Let $X \xrightarrow{\underline{B}} Y$, $e \in B$, and assume for contradiction $e \prec x$ for some $x \in X$. The condition for cause-respecting RAESs requires $x \rhd \underline{e}$ when $e \prec x$. Since $X \xrightarrow{\underline{B}} Y$ and $x \rhd \underline{e}$ we get $x \notin X$ by Definition 4: contradiction. □

It is shown in [32] that a configuration in a cause-respecting RAES is forwards reachable if it is reachable.

When we can undo a and we can also undo b in a configuration, then one may ask if the two events can be undone in sequence, and if the order in which they are undone matters. The corresponding questions are answered in positive for processes in reversible process calculi such as CCSK [28, 29]. If we can reverse actions a and b in a process P, written as $P \xrightarrow{\underline{a}} P'$ and $P \xrightarrow{\underline{b}} P''$, then we can reverse the actions in sequence and the order of reversing the action does not matter: $P'' \xrightarrow{\underline{a}} Q$ and $P' \xrightarrow{\underline{b}} Q$

for some process Q. This is called the *Reverse Diamond* (RD) property [29]. RD, however, is not valid for cause-respecting RAESs.

Example 5 Consider an RAES with two events e and f, with $e \prec f$ and with empty prevention. The events are reversible, so we have $e \prec \underline{e}$ and $f \prec \underline{f}$. Clearly the RAES is cause-respecting. We have $\{e,f\} \xrightarrow{\underline{e}} \{f\}$ and $\{e,f\} \xrightarrow{\underline{f}} \{e\}$. We can undo e in $\{e\}$ but we are unable to undo f in $\{f\}$ because e, one of its causes, is not present.

For RD to hold in an RAES we need to strengthen the cause-respecting condition further. Consider the following two global conditions:

2. events can only be undone once they occur, and the only cause to undo an event is the event itself, namely $x \prec \underline{y}$ if and only if $x = y$ for all $x \in E, y \in F$,
3. causes can be undone precisely when their effects are not present, namely $x \prec y$ if and only if $\underline{x} \lhd y$ for all $x \in F, y \in E$.

Note that (3) is a strengthened version of (1). The RAESs that satisfy these two conditions are called *causal* and are studied further in [32]. We have RD for causal RAESs:

Proposition 2 *Let $\mathcal{E}$ be a causal RAES and X be a configuration of $\mathcal{E}$. If $X \xrightarrow{\underline{e}} X'$ and $X \xrightarrow{\underline{f}} X''$, then $X'' \xrightarrow{\underline{e}} Y$ and $X' \xrightarrow{\underline{f}} Y$ for some Y.*

Proof Assume $X \xrightarrow{\underline{e}} X'$ and $X \xrightarrow{\underline{f}} X''$. Hence, $e,f \in X$, and $X' = X \backslash \{e\}$ and $X'' = X \backslash \{f\}$. We show $X' \xrightarrow{\underline{f}} Y$ for some Y. First, by condition (2) the only cause of $\underline{f}$ is f itself, and $f \in X'$. Second, $X \xrightarrow{\underline{f}} X''$ implies that no event in X prevents $\underline{f}$, hence no event in X' prevents $\underline{f}$. So, by Definition 4, $X' \xrightarrow{\underline{f}} Y$ where $Y = X' \backslash \{f\} = X \backslash \{e,f\}$. Similarly, we show $X'' \xrightarrow{\underline{e}} Y$. □

The work on reversing asymmetric event structures in [31, 32] led to several interesting results concerning reachable configurations. For example, we have given conditions under which finite and reachable configurations are guaranteed to be reachable without intermediate infinite configurations. Our work has been continued in [17] where a causal version of reversible bundle event structures is used to give denotational semantics to CCSK [29]. Interestingly, it is shown there that a *non-causal* form of such reversible bundle event structures is needed to model appropriately an extension of CCSK with a *rollback* operator.

Example 6 Consider two small programs $\texttt{X} = 5; \texttt{Y} = 7$ and $\texttt{X} = \texttt{X} + 1$ that are executed in parallel on shared memory, where initially the values of the variables are $X = Y = 0$. There is a *race* between $\texttt{X} = 5$ and $\texttt{X} = \texttt{X} + 1$ to update the value of X.

To reverse an execution of this parallel program correctly (to return to the initial state of memory) we need to store all the intermediate values of X, Y that are overwritten during the execution, and we need to record the order in which the racing assignments $\mathtt{X} = 5$ and $\mathtt{X} = \mathtt{X} + 1$ execute [19, 20]. There are two sets of execution sequences that produce two distinct final states of memory:

- $\mathtt{X} = 5, \mathtt{Y} = 7, \mathtt{X} = \mathtt{X} + 1$ and $\mathtt{X} = 5, \mathtt{X} = \mathtt{X} + 1, \mathtt{Y} = 7$ produce $X = 6$ and $Y = 7$, and
- $\mathtt{X} = \mathtt{X} + 1, \mathtt{X} = 5, \mathtt{Y} = 7$ results in $X = 5$ and $Y = 7$.

We model executions of this parallel program with an RAES as follows. Our events are the assignments annotated with the values they overwrite in the shared memory, for example $X = 5(0)$ represents overwriting 0 with 5 for X and saving 0. Hence, we have five events

$$X = 5(0),\ X = 5(1),\ X = X + 1(0),\ X = X + 1(5), \quad \text{and} \quad Y = 7(0).$$

The causation represented by the *program order* in $\mathtt{X} = 5;\mathtt{Y} = 7$ gives us $X = 5(0) \prec Y = 7(0)$ and $X = 5(1) \prec Y = 7(0)$. If $X = 5(0)$ happens first then $X = X + 1(5)$ will eventually follow. So we add $X = 5(0) \prec X = X + 1(5)$. Correspondingly, we add $X = X + 1(0) \prec X = 5(1)$. Clearly the racing events $X = 5(0)$ and $X = X + 1(0)$ cannot happen in the same execution, so we insist that they are in conflict: $X = 5(0) \ \sharp\ X = X + 1(0)$.

Next we define how events are undone. We require that $e \prec \underline{e}$ for each of our five events. It is reasonable to expect that the inverse program order is preserved when reversing, so we have $\underline{X = 5(0)} \lhd Y = 7(0)$ and $\underline{X = 5(1)} \lhd Y = 7(0)$. This means that we cannot undo $X = 5(0)$ or $X = 5(1)$ if the event $Y = 7(0)$ is still present. Also, we insist that $\underline{X = 5(0)} \lhd X = X + 1(5)$ and $\underline{X = X + 1(0)} \lhd X = 5(1)$ to ensure that the racing events are backtracked. Note that, with this definition of $\lhd$, our direct causation is a sustained causation (condition 5 of Definition 4). Hence, the reversing is cause-respecting and there is conflict inheritance with respect to $\prec$, implying, for example, $X = X + 1(0) \ \sharp\ X = X + 1(5)$. We easily check that there are two maximal configurations

$$\{X = 5(0),\ X = X + 1(5),\ Y = 7(0)\} \quad \text{and} \quad \{X = X + 1(0),\ X = 5(1),\ Y = 7(0)\}.$$

We can undo $X = X + 1(5)$ and $Y = 7(0)$ in any order in the first configuration before we undo $X = 5(0)$. Thus, we revert at the end to the initial values of X, Y. However, we must backtrack the events in the second configuration to arrive at $X, Y = 0$ and to preserve program order: we undo $Y = 7(0)$ first, then $X = 5(1)$ and finally $X = X + 1(0)$.

Finally, we model the out-of-causal order reversing in Example 2.

Example 7 We have $a \prec b \prec c$ but no $a \prec c$ (so $\prec$ is not transitive) and $a \prec \underline{a}$, $b \prec \underline{b}$ (there is no $c \prec \underline{c}$ since c is irreversible). That a, b are undone only when b, c are present is ensured by $b \prec \underline{a}$, $c \prec \underline{b}$, respectively. To stop reversing b immediately after it occurs we add $\underline{b} \lhd a$. And, $a \lhd b$, $a \lhd c$ prevent a from re-occurring when b or c are present, and $c \lhd a$ prevents c from occurring when a is present. Hence, $a \sharp c$. As a result, there is a single execution $\emptyset \rightarrow \{a\} \rightarrow \{a,b\} \dashrightarrow \{b\} \rightarrow \{b,c\} \dashrightarrow \{c\}$.

Reversible Event Structures with Enablings

There are executions of events a, b, c which cannot be modelled appropriately by any combination of causation, precedence and concurrency. Consider an event that is caused by a disjunction of events: for example, c is caused by a or b. This form of causation is called *disjunctive causation*. If no other relation holds of a, b, c, then there is an execution where only a occurs before c, there is another execution where only b occurs prior to c, and there are two further executions where both a and b precede c. The complete executions are *acb*, *bca*, *abc* and *bac* and they are depicted in the left cube in Fig. 5. This event structure can be defined using the *enabling* relation from [26, 38] or with our own *enabling with prevention* relation, as we shall see below.

Another example of a relation on events that cannot be expressed in terms of causality, precedence and concurrency is a *resolvable conflict*. Consider a temporary conflict between a and b which becomes resolved once another event c occurs. This is represented in the right cube in Fig. 5 by *acb*, *bca*, *cab* and *cba*. Such event structure cannot be expressed with the traditional enabling relation; instead a more general enabling relation from [15] or our enabling with prevention relation (as in Definition 8) is necessary.

Definition 5 An *event structure* is a triple $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ where E is the set of events, $\mathsf{Con} \subseteq \mathcal{P}_{\text{fin}}(E)$ is the *consistency* relation, which is non-empty and satisfies the property $Y \subseteq X \in \mathsf{Con}$ implies $Y \in \mathsf{Con}$ (downwards closure), and $\vdash \subseteq \mathsf{Con} \times E$ is the *enabling* relation which satisfies the *weakening* condition $X \vdash e$ and $X \subseteq Y \in \mathsf{Con}$ implies $Y \vdash e$ for all $e \in E$.

We omit brackets for singleton sets X in expressions $X \vdash e$ where convenient. Informally, configurations are the sets of events that have occurred (in accordance with Con and $\vdash$). More formally,

Definition 6 Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ be an event structure. The set $S(\mathcal{E})$ of *configurations* of $\mathcal{E}$ consists of $X \subseteq E$ which are

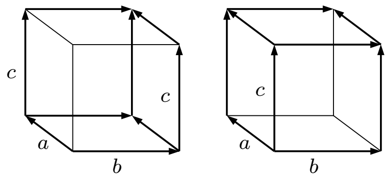

Fig. 5 Left cube (disjunctive causation): either a or b causes c. Right cube (resolvable conflict): an initial conflict between a and b is resolved by occurrence of event c

- *consistent*: every finite subset of X is in Con;
- *secured*: for all $e \in X$ there is a sequence of events $e_0, \dots, e_n \in X$ such that $e_n = e$ and for all $i \leq n$, $\{e_0, \dots, e_{i-1}\} \vdash e_i$.

We shall call the sequence (or the set) $e_0, \dots, e_n$ the *securing* sequence (or the set) for e.

We now present several examples of event structures with enablings and their corresponding configurations.

Consider the events a, b with all subsets of $\{a, b\}$ in Con, and the enabling relation $\emptyset \vdash a$, $a \vdash b$. We notice that $\{a\}$ is a configuration because $\{a\} \in$ Con and a is enabled without any preconditions: $\emptyset \vdash a$. Once a takes place, b can happen because $\{a, b\} \in$ Con and b is enabled by the already performed a: $a \vdash b$. We can say here that a causes b and b cannot take place before a happens first.

Some events can be in conflict and as a result they cannot happen in the same computation. Consider the events a and b, with $\emptyset \vdash a$ and $a \vdash b$, and the event c which is in conflict with a. This is represented by $\{a, c\} \notin$ Con and, by the downwards closure property, $\{a, b, c\} \notin$ Con. The enabling relation is $\emptyset \vdash a$, $a \vdash b$ and $\emptyset \vdash c$. We obtain the following configurations: $\emptyset$, $\{a\}, \{a, b\}$ and $\{c\}$ representing that either a or c can happen first, but once one of a and c has taken place the other cannot happen; see the left cube in Fig. 6.

Some events are independent of each other. Consider the events a, b and c, which are not in conflict, and the enablings $\emptyset \vdash a$, $a \vdash b$ and $\emptyset \vdash c$. Since a and c are not in conflict, $\emptyset \vdash a$, $\emptyset \vdash c$ imply that a, c can happen independently (in any order). Moreover, b and c are independent and can happen in any order (provided that b always follows a). The resulting configurations $\emptyset$, $\{a\}, \{a, b\}$, $\{c\}$, $\{a, c\}$, $\{a, b, c\}$ are given in the right cube in Fig. 6.

Example 8 We now show how to define the disjunctive causation event structure from Fig. 5. Let all subsets of $\{a, b, c\}$ be in Con. If we set the enabling relation to be $\emptyset \vdash a$, $\emptyset \vdash b$, and $a \vdash c$ with $b \vdash c$, then we can deduce that $\{c\}$ is not a configuration since we do not have $\emptyset \vdash c$. All other subsets of $\{a, b, c\}$ are configurations.

The next definition is equivalent to Definition 6; it will be easier to generalise to the reversible setting. It is partly inspired by the step-wise securings of [15, Definition 3.5].

Definition 7 Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ be an event structure. A set $X \subseteq E$ is a *configuration* of $\mathcal{E}$ if there is an infinite sequence $X_0, \dots$ with $X = \bigcup_{n=0}^{\infty} X_n$, $X_0 = \emptyset$, $X_n \subseteq X_{n+1}$

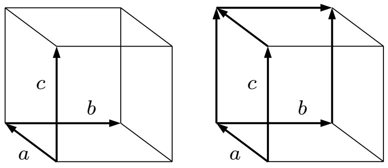

Fig. 6 Left cube: events a and c are in conflict. Right cube: events a and c are concurrent. In both cubes a causes b

and X_n consistent (all $n \in \mathbb{N}$), where for every $n \in \mathbb{N}$, and every $e \in X_{n+1} \backslash X_n$, there is $X' \subseteq_{\text{fin}} X_n$ such that $X' \vdash e$.

Proposition 3 *Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ be an event structure and let $X \subseteq E$. Then X is a configuration according to Definition 6 if and only if X is a configuration according to Definition 7.*

Proof Assume X is a configuration according to Definition 6. For each $n \in \mathbb{N}$, let $X_n = \{e \in X : e \text{ has a securing sequence of length } \leq n\}$. Then clearly $X = \bigcup_{n=0}^{\infty} X_n$, $X_0 = \emptyset$ and $X_n \subseteq X_{n+1}$ (all $n \in \mathbb{N}$). Also each X_n is consistent, since X is consistent. Suppose $e \in X_{n+1} \backslash X_n$. Then e has a securing sequence $e_0, \ldots, e_n = e$ of length $n + 1$. So for all $i \leq n$, $\{e_0, \ldots, e_{i-1}\} \vdash e_i$. It follows that e_i has a securing sequence $e_0, \ldots, e_i$ of length $i + 1$ for $i < n$. Hence, $X' = \{e_0, \ldots, e_{n-1}\} \subseteq_{\text{fin}} X_n$. Also $X' \vdash e$. Hence, X is a configuration according to Definition 7.

Assume $X = \bigcup_{n=0}^{\infty} X_n$ is a configuration according to Definition 7. Then X is consistent, since each X_n is consistent. We show by induction on n that each event in X_n has a securing sequence. This is clearly true for $n = 0$. Suppose that each event in X_n has a securing sequence. Let $e \in X_{n+1} \backslash X_n$. Then we have $X' \vdash e$ with $X' \subseteq_{\text{fin}} X_n$. Let $X' = \{e^1, \ldots, e^k\}$. By induction hypothesis there are securing sequences $e_0^i, \ldots, e_{n_i}^i = e^i$ for $i = 1, \ldots, k$. Then $e_0^1, \ldots, e_{n_1}^1, \ldots, e_0^k, \ldots, e_{n_k}^k, e$ is a securing sequence for e. Hence, each event in X_{n+1} has a securing sequence. We have shown that X is a configuration according to Definition 6. $\square$

We are now ready to introduce reversing of events via the enabling relation. As before assume E is a set of events. We define the corresponding set of *undone* events (strictly speaking, events that are to be undone) to be $\underline{E} = \{\underline{e} : e \in E\}$. To abbreviate presentation of enablings we let e^* be either e or $\underline{e}$ for $e \in E$. We often use the notation $X + e^*$ to denote either $X \cup \{e\}$ or $X \backslash \{e\}$, respectively. We shall use expressions of the form $X \oslash Y$ in $X \oslash Y \vdash e^*$ below, where X and Y are sets of events, to specify that the events in X enable e^* and, at the same time, the events in Y prevent e^*. We require that $X \cap Y = \emptyset$ and $e \notin Y$. Also, if $e^* = \underline{e}$, then we require that $e \in X$. *Reversible event structures* were first introduced in [35]:

Definition 8 A reversible event structure (RES for short) is a triple $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ where E and Con are as before and $\vdash \subseteq \mathsf{Con} \times \mathcal{P}(E) \times (E \cup \underline{E})$ is the enabling with prevention relation. We write a typical element of $\vdash$ as $X \oslash Y \vdash e^*$ and insist that it satisfies

1. if $X \oslash Y \vdash e^*$ then $X \cap Y = \emptyset$ and $e \notin Y$;
2. if $X \oslash Y \vdash \underline{e}$ then $e \in X$;
3. *weakening:* if $X \oslash Y \vdash e^*$ and $X \subseteq X' \in \mathsf{Con}$ then $X' \oslash Y \vdash e^*$, provided $X' \cap Y = \emptyset$.

When $Y = \emptyset$ we shall write $X \oslash Y \vdash e^*$ as $X \vdash e^*$, and we call the relation $\vdash$ simply enabling. We omit brackets for singleton sets in expressions $X \oslash Y \vdash e^*$ where convenient.

Our enabling with prevention relation $\vdash$ extends the enabling relation in [38] in two directions. First, we permit reversing of events as e^* in $X \oslash Y \vdash e^*$ can be $\underline{e}$. Second, it allows us to specify some of the events that *prevent* e^* (here those in Y) in addition to the events that enable e^* (those in X). For example, $\{a\} \oslash \{b\} \vdash \underline{a}$ says that a can be undone in a configuration which contains a and provided that b is not a member of that configuration.

To illustrate the usefulness of the new enabling with prevention relation, we define an RES for resolvable conflict in Fig. 5.

Example 9 We let Con be $\mathcal{P}(\{a, b, c\})$. The enabling with prevention relation is given by $\emptyset \vdash c$, $\emptyset \oslash b \vdash a$ and $\emptyset \oslash a \vdash b$. This means that initially either a or b can occur if the other event has not occurred yet. We also have $c \vdash a$ and $c \vdash b$, which implies that both a and b can happen after c.

Next, we represent the out-of-causal order RAES in Example 7 as an RES.

Example 10 The sets in Con of the RES are the members of $\mathcal{P}(\{a, b, c\})$ except for $\{a, c\}$ and $\{a, b, c\}$ since $a \sharp c$. Initially, the event a occurs. Since we do not want a to reoccur later in computation when b or c are present we use $\emptyset \oslash \{b, c\} \vdash a$ to achieve this. The enabling $a \vdash b$ ensures that a causes b, and $b \oslash a \vdash c$ ensures that b causes c but only when a is not present. Finally, $\{a, b\} \vdash \underline{a}$ and $\{b, c\} \vdash \underline{b}$ ensure that the catalysts a and b are reversed at the appropriate time.

Examples 7 and 10 suggest that it should be possible to represent an arbitrary RAES as a special form of RES as we stated in [36] (explored in the categorical setting in [16]). Given an RAES if $X_a = \{e \mid e \prec a\}$ and $Y_a = \{f \mid a \lhd f\}$, then the enabling with prevention rule $X_a \oslash Y_a \vdash a$ captures the idea that a can occur if all events in X_a have occurred (and are present) and if no events from Y_a are present. A corresponding conversion can be defined for $\underline{e}$ in terms of events that cause it and prevent it, giving us the following mapping from RAESs to RESs.

Definition 9 Let $\mathcal{E} = (E, F, \prec, \lhd)$ be an RAES. We define Con as the set of subsets X in E on which $\lhd$ is well-founded, and $\vdash$ as follows, where $e, f \in E$:

- $X \oslash Y \vdash a$ if $a \in E$, $\{e \mid e \prec a\} = X \in \mathsf{Con}$ and $Y = \{f \mid a \lhd f\}$;
- $X \oslash Y \vdash \underline{a}$ if $a \in F$, $\{e \mid e \prec \underline{a}\} = X \in \mathsf{Con}$ and $Y = \{f \mid \underline{a} \lhd f\}$.
- if $X \oslash Y \vdash e^*$ and $X \subseteq X' \in \mathsf{Con}$ then $X' \oslash Y \vdash e^*$, provided $X' \cap Y = \emptyset$.

Proposition 4 *Let* $\mathcal{E} = (E, F, \prec, \lhd)$ *be an RAES. Then* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$, *where* Con *and* $\vdash$ *are as in Definition 9, is an RES.*

Proof If $\lhd$ is well founded on X then $\lhd$ is well founded on $Y \subseteq X$. Hence, the set of all $X \subseteq E$ on which $\lhd$ is well founded satisfies the property of the consistency relation. Next, we show that the conditions for $\vdash$ in Definition 8 hold too. Condition 4 in Definition 3 implies that $X \cap Y = \emptyset$. Also, not $e^* \lhd e$, for all e, since $\lhd$ is irreflexive, hence $e^* \notin Y$ and $(X \cup \{e\}) \cap Y = \emptyset$. We have $a \prec \underline{a}$, for all $a \in F$ by condition 3 in Definition 8, and so $a \in X$ for each $X \oslash Y \vdash \underline{a}$. Finally, the weakening condition holds. $\square$

If we apply the mapping from Definition 9 to the RAES from Example 7 we obtain the RES in Example 10 except that the enabling with prevention $\{b, c\} \oslash a \vdash \underline{b}$ replaces $\{b, c\} \vdash \underline{b}$, and the other enablings are the same. Since $a \sharp c$ the effect of the two enablings is the same.

Since disjunctive causation in Fig. 5 cannot be expressed in RAESs but, as we have seen in Example 8, can be defined by an RES, RESs are strictly more expressive than RAESs.

Example 11 Consider a simple RES with one event e and the enabling $\emptyset \vdash e$. The sets $\emptyset$ and $\{e\}$ are configurations. Next we add the second enabling $e \vdash \underline{e}$ as this allows us to reverse from $\{e\}$ to $\emptyset$. The sets $\emptyset$ and $\{e\}$ are reachable from $\emptyset$ by performing and reversing e any number of times. We easily check that $\emptyset$ and $\{e\}$ are configurations according to Definition 11 below. Interestingly, there is an infinite computation sequence $\emptyset, \{e\}, \emptyset, \{e\}, \ldots$

The example shows that configurations can grow and shrink as reversible computation progresses. Also, sets of events may grow non-monotonically as, for example, in the computation represented by this sequence:

$$a_0, b, a_1, \underline{b}, a_2, b, a_3, \underline{b}, a_4, \ldots$$

We note that after the initial three events are done we reach the configuration $\{a_0, b, a_1\}$. When b is undone the resulting configuration is smaller: $\{a_0, a_1\}$. Then, when we do a_2, b and a_3, the resulting configuration grows again before becoming smaller again (by undoing b), and so on. Hence, we shall use limits of infinite sequences of subsets of E to define configurations as in [35]. Recall that $S \subseteq \mathbb{N}$ is *cofinite* provided $\mathbb{N} \backslash S$ is finite.

Definition 10 Let $X_0, \ldots$ be an infinite sequence of subsets of E. We say that $X = \lim_{n \to \infty} X_n$ if for every $e \in E$:

1. $\{n \in \mathbb{N} : e \in X_n\}$ is either finite or cofinite;
2. $e \in X$ if and only if $\{n : e \in X_n\}$ is cofinite.

We note that a sequence of sets does not necessarily have a limit. The sequence $\emptyset, \{e\}, \emptyset, \{e\}, \ldots$ in Example 11 has no limit, since e belongs to infinitely many sets and does not belong to infinitely many sets.

Next we state the definition of a configuration for an RES [35]. As a notational convention we write $\underline{e} \in A \backslash B$ to mean $e \in B \backslash A$.

Definition 11 Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ be an RES. A set $X \subseteq E$ is a *configuration* of $\mathcal{E}$ if there is an infinite sequence $X_0, \dots$ with $X = \lim_{n \to \infty} X_n$, $X_0 = \emptyset$ and $X_n \cup X_{n+1}$ consistent (all $n \in \mathbb{N}$), where for every $n \in \mathbb{N}$, and every $e^* \in X_{n+1} \backslash X_n$, there is an enabling with prevention rule $X' \oslash Y' \vdash e^*$ such that:

1. $X' \subseteq_{\text{fin}} X_n$ and $X' + e^* \subseteq X_{n+1}$;
2. $Y' \cap (X_n \cup X_{n+1}) = \emptyset$.

We require $X_n \cup X_{n+1}$ to be consistent, as configurations can only be extended safely in a consistent way. However, there is no requirement that $X_i \cup X_j$ is consistent if $j > i + 1$ because events in X_i which are inconsistent with X_j can be reversed in constructing $X_{i+1}, \dots, X_{j-1}$. Also, we note that the X_is in the above definition can grow smaller as well as bigger as computation progresses. Moreover, a finite sequence $X_0, \dots, X_n$, where $X_n = X$, that satisfies the conditions of Definition 11 is sufficient for X to be a configuration. The sequence $\emptyset, \{e\}$ in Example 11 can be extended to an infinite sequence and, since the conditions of Definition 11 are satisfied, its limit $\{e\}$ is a configuration.

We give an example where we get an infinite configuration as a limit of a non-monotonically increasing sequence.

Example 12 Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ where $E = \{a_i : i \in \mathbb{N}\} \cup \{b_j : j \in \mathbb{N}\}$ and Con consists of the sets $\{a_i, b_0, \dots, b_j\}$ for any $i, j \in \mathbb{N}$, as well all the deducible subsets. The enablings are as follows, for all $i \in \mathbb{N}$:

$$\emptyset \vdash a_0 \quad a_i \vdash b_i \quad \{a_i, b_i\} \vdash \underline{a}_i \quad b_i \vdash a_{i+1}$$

Informally, a_i is the catalyst of b_i, for all i, so once b_i occurs a_i can be undone.

The only possible computation of $\mathcal{E}$ is $a_0, b_0, \underline{a}_0, a_1, b_1, \underline{a}_1, \dots$ It produces the following sequence of sets of events, where the sets grow non-monotonically:

$$\begin{array}{l}
\emptyset, \\
\{a_0\},\ \{a_0, b_0\},\ \{b_0\}, \\
\{b_0, a_1\},\ \{b_0, a_1, b_1\},\ \{b_0, b_1\}, \\
\{b_0, b_1, a_2\},\ \{b_0, b_1, a_2, b_2\},\ \{b_0, b_1, b_2\}, \\
\dots
\end{array}$$

Each of the sets is a configuration and this sequence has as limit the infinite set $\{b_j : j \in \mathbb{N}\}$, so $\{b_j : j \in \mathbb{N}\}$ is also a configuration. Note that each a_i appears finitely often in the sequence, namely twice, while each b_j appears cofinitely often.

Having defined our configurations we go back to Example 9. The set $\{a, b\}$ is consistent (according to Con) but it is not a configuration according to Definition 11. Consider $\emptyset$, $\{a\}$, $\{a, b\}$ and b: there is no enabling with prevention $X' \oslash Y' \vdash b$

such that $X' \subseteq_{\text{fin}} \{a\}$ and $Y' \cap \{a,b\} = \emptyset$. Correspondingly for the sequence $\emptyset$, $\{b\}$, $\{a,b\}$ and a. Hence, $\{a,b\}$ is not a configuration.

We can now show that RESs are a generalisation of traditional event structures in [38] by taking enablings with prevention $X \oslash \emptyset \vdash e$. Then our configurations are just the traditional configurations. This is a consequence of the following observation. Consider consistent sets of events X_n for all $n \in \mathbb{N}$. If $X_n \subseteq X_{n+1}$ for all $n \in \mathbb{N}$, then $\lim_{n\to\infty} X_n$ exists and is $\bigcup_{n=0}^{\infty} X_n$. A finite sequence $X_0, \dots, X_n$ can be extended to an infinite sequence by letting $X_m = X_n$ for all $m > n$; the extended sequence has the limit X_n. In Example 11 the sequence $\emptyset, \{e\}$ can be extended to an infinite sequence $\emptyset, \{e\}, \{e\}, \dots$ and has the limit $\{e\}$.

Proposition 5 *Suppose* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ *is an event structure. Then* $\mathcal{E}' = (E, \mathsf{Con}, \vdash')$ *is a reversible event structure, where we define* $X \oslash \emptyset \vdash' e$ if and only if $X \vdash e$, *and there are no reverse enablings with prevention* $X \oslash Y \vdash' \underline{e}$. *Moreover,* X *is a configuration of* $\mathcal{E}$ *according to Definition 7 if and only if* X *is a configuration of* $\mathcal{E}'$ *according to Definition 11.*

Proof Assume X is a configuration of $\mathcal{E}$ according to Definition 7. There is an infinite sequence $X_0, \dots$ with $X_n \subseteq X_{n+1}$ and X_n consistent for all $n \in \mathbb{N}$. Hence, $\lim_{n\to\infty} X_n$ exists and is $\bigcup_{n=0}^{\infty} X_n$. Since $X_n \subseteq X_{n+1}$ we have that $X_n \cup X_{n+1}$ are consistent for all n. Clearly $X' \vdash e$ if and only if $X' \vdash' e$, with their conditions satisfied. Hence, X is a configuration of $\mathcal{E}'$ according to Definition 11.

Assume X is a configuration of $\mathcal{E}'$ according to Definition 11. Since we do not reverse events in $\mathcal{E}'$ the members of the infinite sequence are not decreasing in size. So $X_n \subseteq X_{n+1}$ and since all $X_n \cup X_{n+1}$ are consistent each X_n is consistent. Since $\lim_{n\to\infty} X_n$ exists it equals $\bigcup_{n=0}^{\infty} X_n$. Also, $X' \vdash e$ if and only if $X' \vdash' e$ with their side conditions. Hence, X is a configuration of $\mathcal{E}$ according to Definition 7. $\square$

We can also show that our enabling with prevention rules are powerful enough that we no longer need the consistency relation.

Proposition 6 *Let* $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ *be an RES. Define* $\mathsf{Con}' = \mathcal{P}_{\text{fin}}(E)$ *and define* $\vdash'$ *by* $X \oslash (Y \cup (E \backslash Z)) \vdash' e^*$ *whenever* X, Y, Z *are such that* $X \oslash Y \vdash e^*$, Z *is consistent with respect to* Con *and* $X + e^* \subseteq Z$. *Then* $\mathcal{E}' = (E, \mathsf{Con}', \vdash')$ *is an RES, and* X *is a configuration of* $\mathcal{E}$ *if and only if* X *is a configuration of* $\mathcal{E}'$.

Proof Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ and $\mathcal{E}' = (E, \mathsf{Con}', \vdash')$ be as stated. It is straightforward to check that $\mathcal{E}'$ is an RES.

Assume that X is a configuration of $\mathcal{E}$ with $X = \lim_{n\to\infty} X_n$. We show that X is a configuration of $\mathcal{E}'$, also with $X = \lim_{n\to\infty} X_n$. It is clear that each $X_n \cup X_{n+1}$ is Con'-consistent. Take $e^* \in X_{n+1} \backslash X_n$. Then there is a rule $X' \oslash Y' \vdash e^*$ such that $X' \subseteq_{\text{fin}} X_n$ and $X' + e^* \subseteq X_{n+1}$ with $Y' \cap X_n = Y' \cap X_{n+1} = \emptyset$. Let $Z = X_n \cup X_{n+1}$. Then Z is Con-consistent. Also $X' + e^* \subseteq Z$. Hence, $X' \oslash (Y' \cup (E \backslash Z)) \vdash' e^*$. Clearly $(E \backslash Z) \cap (X_n \cup X_{n+1}) = \emptyset$. It follows that $(Y' \cup (E \backslash Z)) \cap (X_n \cup X_{n+1}) = \emptyset$, and we have completed checking that X is a configuration of $\mathcal{E}'$.

Assume that X is a configuration of $\mathcal{E}'$ with $X = \lim_{n\to\infty} X_n$. We show that X is a configuration of $\mathcal{E}$, also with $X = \lim_{n\to\infty} X_n$. First, we show by induction on n that $X_n \cup X_{n+1}$ is Con-consistent. For the base case we note that $X_0 = \emptyset$ is Con -consistent. We can assume inductively that X_n is Con-consistent. If $X_{n+1} = X_n$ then plainly $X_n \cup X_{n+1}$ is Con-consistent. So assume that $X_{n+1} \neq X_n$. Then there is $e^* \in X_{n+1} \backslash X_n$, with $X' \oslash Y' \vdash' e^*$. Then $Y' = Y \cup (E \backslash Z)$ where Z is Con-consistent. But $Y' \cap (X_{n+1} \cup X_n) = \emptyset$. Hence, $(E \backslash Z) \cap (X_{n+1} \cup X_n) = \emptyset$. This means that $X_{n+1} \cup X_n \subseteq Z$. So $X_{n+1} \cup X_n$ is Con-consistent.

Now take any $e^* \in X_{n+1} \backslash X_n$ with $X' \oslash Y' \vdash' e^*$. Then there is $Y \subseteq Y'$ such that $X' \oslash Y \vdash e^*$. Since $Y' \cap (X_{n+1} \cup X_n) = \emptyset$, also $Y \cap (X_{n+1} \cup X_n) = \emptyset$. It follows that X is a configuration of $\mathcal{E}$. □

In the light of Proposition 6, we could dispense with Con altogether in the setting of RESs. However, we allow Con as sometimes it may be natural or convenient to identify certain configurations as being consistent or inconsistent, before defining enabling rules in detail.

Example 13 Let $E = \{a, b, c\}$, $\mathsf{Con} = \{\{a, c\}, \{b, c\}\}$ plus deducible subsets, and $\emptyset \vdash a$, $\emptyset \vdash b$, $a \vdash c$, $b \vdash c$. Then $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ is a (reversible) event structure where either a or b causes c, and $\{a, b\}$ is inconsistent. We can use the procedure of Proposition 6 to convert $\mathcal{E}$ into $\mathcal{E}' = (E, \mathsf{Con}', \vdash')$ where $\mathsf{Con}' = \mathcal{P}_{\mathrm{fin}}(E)$ and $\emptyset \oslash b \vdash' a$, $\emptyset \oslash \{b, c\} \vdash' a$, $\emptyset \oslash a \vdash' b$, $\emptyset \oslash \{a, c\} \vdash' b$, $a \oslash b \vdash' c$, $b \oslash a \vdash' c$. The configurations are $\emptyset$, $\{a\}$, $\{b\}$, $\{a, c\}$, $\{b, c\}$ for both $\mathcal{E}$ and $\mathcal{E}'$. However, in $\mathcal{E}'$ there are two extra consistent sets $\{a, b\}$ and $\{a, b, c\}$.

The converted RES in Example 13 can be optimised by removing $\emptyset \oslash \{b, c\} \vdash' a$ and $\emptyset \oslash \{a, c\} \vdash' b$, since they are implied in some sense by $\emptyset \oslash b \vdash' a$ and $\emptyset \oslash a \vdash' b$, respectively.

Definition 12 Let $\mathcal{E} = (E, \mathcal{P}_{\mathrm{fin}}(E), \vdash)$ be an RES. We say that enabling with prevention rule $X \oslash Y \vdash e^*$ is *removable* if,

1. there is no $X' \subsetneq X$ such that $X' \oslash Y \vdash e^*$ (X is minimal);
2. there is $Y' \subsetneq Y$ such that $X \oslash Y' \vdash e^*$ (Y is not minimal).

Proposition 7 *Let $\mathcal{E} = (E, \mathcal{P}_{\mathrm{fin}}(E), \vdash)$ be an RES and let $X' \oslash Y \vdash e^*$ be removable. Let $\mathcal{E}' = (E, \mathcal{P}_{\mathrm{fin}}(E), \vdash')$ be obtained from $\mathcal{E}$ by removing $X' \oslash Y \vdash e^*$. Then $\mathcal{E}'$ is an RES, and X is a configuration of $\mathcal{E}$ if and only if X is a configuration of $\mathcal{E}'$.*

Proof To see that $\mathcal{E}'$ is an RES, it is enough to note that the weakening condition is preserved by removal, since we removed $X' \oslash Y \vdash e^*$ where X' is minimal.

If X is a configuration of $\mathcal{E}'$ then it is clearly a configuration of $\mathcal{E}$.

Assume that X is a configuration of $\mathcal{E}$ with $X = \lim_{n\to\infty} X_n$. We show that if the sequence $X_0, \ldots, X_n, \ldots$ can be derived using $\vdash$ then it can be derived with $\vdash'$. Consider X_n, X_{n+1}, and let $e^* \in X_{n+1} \backslash X_n$ be derived by the rule $X' \oslash Y \vdash e^*$. If

$X' \oslash Y \vdash e^*$ was removed, then there is $X' \oslash Z \vdash e^*$ a rule of $\vdash$ such that $Z \subsetneq Y$ (this implies $X' \oslash Z \vdash e^* \in \vdash'$). We have $Y \cap (X_n \cup X_{n+1}) = \emptyset$ by Definition 11. Hence, $Z \cap (X_n \cup X_{n+1}) = \emptyset$, and so we can use $X' \oslash Z \vdash e^* \in \vdash'$ to compute e^*. If $X' \oslash Y \vdash e^*$ was not removed, then $X' \oslash Y \vdash e^*$ is a rule of $\vdash'$ and so can be used to derive e^*. Hence, X is a configuration of $\mathcal{E}'$. $\square$

In the case of finite RESs we can iterate the removal procedure until no further rules are removable.

We are now ready to define a transition relation between configurations of an RES. A simple transition relation for RESs, where we move between configurations by performing a single event or by reversing a single event, was given in [36]. It is now generalised to a multiple-event mixed transition relation, building on our Definitions 1 and 3 and in [16], as follows:

Definition 13 Let $\mathcal{E} = (E, \mathsf{Con}, \vdash)$ be an RES. Given configurations X, Y of $\mathcal{E}$ and the sets of events A, B of E we let $X \xrightarrow{A \cup \underline{B}} Y$ if

- $Y = (X \backslash B) \cup A$, $X \cap A = \emptyset$, $B \subseteq X$, and $X \cup A$ is a configuration;
- for all $e \in A$ we have $X' \oslash Z \vdash e$ for some X', Z such that $X' \subseteq_{\text{fin}} X \backslash B$ and $Z \cap (X \cup A) = \emptyset$;
- for all $e \in B$ we have $X' \oslash Z \vdash \underline{e}$ for some X', Z such that $X' \subseteq_{\text{fin}} X \backslash (B \backslash \{e\})$ and $Z \cap (X \cup A) = \emptyset$.

Having given the transition relation, we can now define a configuration system for an RES. Assuming an RES $\mathcal{E} = (E, \mathsf{Con}, \vdash)$, the associated configuration system $C(\mathcal{E})$ is $(E, E, \mathsf{C}, \rightarrow)$ where C is the set of configurations for $\mathcal{E}$ as in Definition 11.

Example 14 Consider two RESs P and M which have two events a, b each where all subsets of $\{a, b\}$ are in Con. The enabling rules for P are simply $\emptyset \vdash a$ and $\emptyset \vdash b$, meaning that a and b are concurrent. The configurations of P are $\emptyset$, $\{a\}$, $\{b\}$ and $\{a, b\}$ with all obvious transitions between them, including $\emptyset \xrightarrow{\{a,b\}} \{a, b\}$, which could be verified by Definition 13.

The RES M is exactly as P except that the enablings with prevention rules are $\emptyset \oslash b \vdash a$, $\emptyset \oslash a \vdash b$, and $a \vdash \underline{b}$ and $b \vdash \underline{a}$. We deduce that its configurations are the same as those of P: $\emptyset$, $\{a\}$, $\{b\}$ and $\{a, b\}$. Moreover, the transitions of M are the same as those of P except that we do not have $\emptyset \xrightarrow{\{a,b\}} \{a, b\}$! According to Definition 13 although $\emptyset \oslash Z \vdash a$, with $Z = \{b\}$, we fail to have $Z \cap \{a, b\} = \emptyset$. We conclude that a and b can occur independently of each other but only in *mutual exclusion*. We recall that mutual exclusion cannot be modelled by traditional event structures [38]; it requires more general enabling rules of the form $X \vdash Y$, where $X \cap Y \neq \emptyset$, as can be seen in [15]. This shows the usefulness of enablings with prevention in representing mutual exclusion.

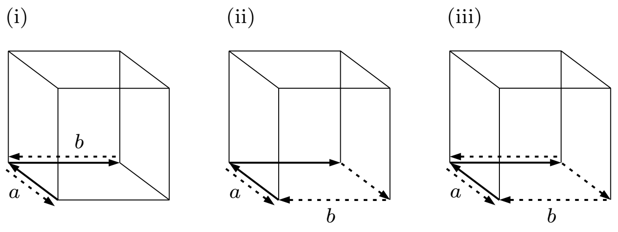

Fig. 7 Left cube: causal reversibility. Middle cube: out-of-causal order reversibility. Right cube: any order reversibility

We now show how to represent different forms of undoing of events in RESs. Consider events a and b with $\emptyset \vdash a$ and $a \vdash b$. We have that a causes b so if we wish to achieve causal reversing we need to add the following to the definition of $\vdash$: $b \vdash \underline{b}$ and $a \oslash b \vdash \underline{a}$. The configuration $\{a, b\}$ can regress to $\{a\}$ by undoing b as allowed by $b \vdash \underline{b}$. But it cannot regress to $\{b\}$ because $a \oslash b \vdash \underline{a}$ can only be applied in a configuration that contains a and does not contain b. This is represented in Fig. 7(i). If undoing events in the same order as they occurred is required, which is an example of the out-of-causal order reversibility, we instead add to the definition of $\vdash$ the following: $a \vdash \underline{a}$ and $b \oslash a \vdash \underline{b}$. This means that a can be reversed in any configuration that contains a (with or without b), and b can be reversed only when a is not present. Since a causes b, this means that b can be reversed only when a is reversed. This is represented in Fig. 7(ii) where reverse transitions are indicated by dashed lines. Finally, if we would like instead that a and b are reversed in any order, then we would extend the enabling relation simply with $b \vdash \underline{b}$ and $a \vdash \underline{a}$. This is shown in Fig. 7(iii).

Example 15 We now show how to control reversing with a trigger event trig. A process performs events $a_1, \ldots, a_n$ successively as in, for example, a long running transaction. The trigger event trig can happen at any stage and when it occurs the process reverses to the start, at which point trig is also reversed, and the process can start afresh. The enabling with prevention relation is given below.

$$\begin{aligned}
\emptyset \oslash \mathsf{trig} &\vdash a_1 \\
a_i \oslash \mathsf{trig} &\vdash a_{i+1} \qquad (i = 1, \ldots, n-1) \\
\emptyset &\vdash \mathsf{trig} \\
a_n, \mathsf{trig} &\vdash \underline{a}_n \\
a_i, \mathsf{trig} \oslash a_{i+1} &\vdash \underline{a}_i \qquad (i = 1, \ldots, n-1) \\
\mathsf{trig} \oslash a_1 &\vdash \underline{\mathsf{trig}}
\end{aligned}$$

$$\begin{tikzcd}
\emptyset \arrow[rr] \arrow[dr] & & \{a_1\} \arrow[rr] \arrow[dr] & & \{a_1, a_2\} \arrow[dr] & \\
& \{\mathsf{trig}\} \arrow[ul, dashed] & & \{a_1, \mathsf{trig}\} \arrow[ll, dashed] & & \{a_1, a_2, \mathsf{trig}\} \arrow[ll, dashed]
\end{tikzcd}$$

Fig. 8 Configurations and transitions in Example 15

See Fig. 8 for the case where $n = 2$.

The next example is inspired by the mechanism of protein bonding in the ERK signalling pathway [34].

Example 16 We describe bonding and unbonding that takes place along a section of the ERK signalling pathway. The molecule A receives a signal P at the top of the pathway by bonding to it. This we represent by $P \cdot A$. The molecule $P \cdot A$ travels then towards the middle of the pathway where it combines with B, producing $P \cdot A \cdot B$ where A is bonded to both P and B but where P and B are not directly bonded. A bond between P and B is then created, which is denoted by the over-bracket joining P and B in $\overline{P \cdot A \cdot B}$. Next, the bond between P and A is dissolved thus, in a sense, passing the signal P to B. Once the bond between A and B is broken B is able to pass P towards the bottom of the ERK pathway. The creation of bonds is represented informally by the following chemical equations

$$P,\ A \rightarrow P \cdot A \qquad P \cdot A, B \rightarrow P \cdot A \cdot B \qquad P \cdot A \cdot B \rightarrow \overline{P \cdot A \cdot B}$$

and the breaking of the bonds is given, again informally, by these equations

$$\overline{P \cdot A \cdot B} \rightsquigarrow \overline{P,\ A \cdot B}\ (\equiv A \cdot B \cdot P) \quad A \cdot B \cdot P \rightsquigarrow A,\ B \cdot P \quad B \cdot P\ \rightsquigarrow B,\ P.$$

The events are pa, ab and bp and they represent the bonds $P \cdot A$, $A \cdot B$ and $B \cdot P$, respectively. The set Con is $\mathcal{P}(\{pa, ab, bp\})$. We use the following enabling rules to model the creation of bonds:

$$\emptyset \vdash pa \qquad pa \vdash ab \qquad \{pa, ab\} \vdash bp$$

and the enabling with prevention rules to define when bonds are broken

$$\{pa, ab, bp\} \vdash \underline{pa} \qquad \{ab, bp\} \oslash pa \vdash \underline{ab} \qquad bp \oslash \{pa, ab\} \vdash \underline{bp}.$$

Note how the operator $\oslash$ is used in the last two rules to enforce the order of undoing of pa, ab and bp.

The configurations are $\emptyset, \{pa\}, \{pa, ab\}, \{pa, ab, bp\}, \{ab, bp\}, \{bp\}$, and the creation and dissolving of the bonds happens in the following order: pa, ab, bp, $\underline{pa}, \underline{ab}, \underline{bp}$. We observe that pa causes ab which causes bp, and that the bonds are reversed out-of-causal order. We note that pa can happen again in $\{ab, bp\}$ and in $\{bp\}$ due to $\emptyset \vdash pa$. However, ab cannot happen in $\{bp\}$ as it requires pa to be present (due to the enabling $pa \vdash ab$).

Table 1 Relations on events, their notation and place of definition

Relation	Notation	Place of definition
Causation	$a \prec b$	[38]
Direct causation	$a \prec b$	Definition 3
Sustained causation	$a \lll b$	Definition 3
Conflict	$a \sharp b$	[38]
Precedence	$a \lhd b$	Definition 3
Prevention	$a \rhd b$	Definition 3
Enabling	$X \vdash e$	[38], Definition 5
Enabling with prevention	$X \oslash Y \vdash e$	Definition 8

Conclusions

We have shown how to model reversible concurrent computation by means of two different reversible event structures, namely event structures defined by the causation and prevention relations and event structures given by an enabling with prevention relation. Table 1 lists all the relations on events that we have used, their notation and the place of definition. We have discussed in this paper both causal reversibility as well as out-of-causal order reversibility.

It would be interesting to investigate the expressiveness of event structures defined by our enabling relation with prevention for forwards-only computation, and to compare such event structures with other forms of event structures. The event structures of van Glabbeek and Plotkin [15], which are more general than those in [38, 39] and which correspond to safe Petri nets and propositional theories, are of particular interest. They are defined by enabling rules of the form $X \vdash Y$ where X, Y are sets of events. However, there is an incompatibility between our definitions and those in [15], especially regarding the rôle of the consistency relation and the notion of configuration, meaning that a common setting needs to be developed first before potential encodings are explored, which goes beyond the scope of this paper.

Concluding, we have shown how to model reversibility in concurrent computation as realised by two different forms of event structures, namely event structures defined in terms of the causation and precedence relations and event structures defined by the enabling relation with prevention. We have discussed and given examples of causal reversibility as well as of out-of-causal order reversibility.

Acknowledgements The first author thanks the University of Leicester for granting Academic Study Leave during which he researched at Graduate School of Information Science, Nagoya University, Japan. The third author is supported by JSPS KAKENHI Grant-in-Aid for Scientific Research(B) number 17H01722. The authors also acknowledge partial support from the COST Action IC1405 on Reversible Computation.

References

1. Aubert, C., Cristescu, I.: Contextual equivalences in configuration structures and reversibility. J. Log. Algebraic Methods Program. **86**(1), 77–106 (2017)
2. Baldan, P., Corradini, A., Montanari, U.: Contextual Petri nets, asymmetric event structures, and processes. Inf. Comput. **171**(1), 1–49 (2001)
3. Barylska, K., Gogolinska, A., Mikulski, L., Philippou, A., Piatkowski, M., Psara, K.: Reversing computations modelled by coloured Petri nets. In: Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data 2018, vol. 2115 of CEUR Workshop Proceedings, pp. 91–111 (2018)
4. Barylska, K., Koutny, M., Mikulski, L., Piatkowski, M.: Reversible computation vs. reversibility in Petri nets. Sci. Comput. Program. **151**, 48–60 (2018)
5. Bednarczyk, M.A.: Hereditary history preserving bisimulations or what is the power of the future perfect in program logics. Technical Report ICS PAS. Polish Academy of Sciences (1991)
6. Berry, G., Boudol, G.: The chemical abstract machine. Theor. Comput. Sci. **96**(1), 217–248 (1992)
7. Cardelli, L., Laneve, C.: Reversible structures. In: 9th International Conference on Computational Methods in Systems Biology. ACM, New York, pp. 131–140 (2011)
8. Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible pi-calculus. In: Proceedings of LICS 2013. IEEE Computer Society, pp. 388–397 (2013)
9. Cristescu, I., Krivine, J., Varacca, D.: Rigid families for the reversible pi-calculus. In: Proceedings of Reversible Computation 2016, vol. 9720, LNCS. Springer, New York, pp. 3–19 (2016)
10. Danos, V., Krivine, J.: Reversible communicating systems. In: Proceedings of CONCUR 2004, vol. 3170 of LNCS. Springer, New York, pp. 292–307 (2004)
11. Danos, V., Krivine, J.: Transactions in RCCS. In: Proceedings of CONCUR 2005, vol. 3653 of LNCS. Springer, New York, pp. 398–412 (2005)
12. Danos, V., Krivine, J.: Formal molecular biology done in CCS-R. In: Proceedings of BioConcur 2003, vol. 180 of ENTCS, pp. 31–49 (2007)
13. Giachino, E., Lanese, I., Mezzina, C.A.: Causal-consistent reversible debugging. In: Proceedings of FASE 2014, vol. 8411 of LNCS. Springer, New York, pp. 370–384 (2014)
14. van Glabbeek, R.J., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Inform. **37**, 229–327 (2001)
15. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures, event structures and Petri nets. Theor. Comput. Sci. **410**(41), 4111–4159 (2009)
16. Graversen, E., Phillips, I.C.C., Yoshida, N.: Towards a categorical representation of reversible event structures. In: Proceedings of PLACES 2017, vol. 246 of EPTCS, pp. 49–60 (2017)
17. Graversen, E., Phillips, I.C.C., Yoshida, N.: Event structure semantics of (controlled) reversible CCS. In: Proceedings of Reversible Computation (2018) (**to appear**)
18. Hennessy, M., Milner, R.: Algebraic laws for non-determinism and concurrency. J. ACM **32**, 137–161 (1985)
19. Hoey, J., Ulidowski, I., Yuen, S.: Reversing imperative parallel programs. In: Proceedings of Express/SOS 2017, vol. 255 of EPTCS, pp. 51–66 (2017)
20. Hoey, J., Ulidowski, I., Yuen, S.: Reversing parallel programs with blocks and procedures. In: Proceedings of Express/SOS (2018) (**to appear**)
21. Kaarsgaard, R., Axelsen, H.B., Glück, R.: Join inverse categories and reversible recursion. J. Log. Algebraic Methods Program. **87**, 35–50 (2017)
22. Kuhn, S., Ulidowski, I.: Local reversibility in a calculus of covalent bonding. Sci. Comput. Program. **151**, 18–47 (2017)
23. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.-B.: Controlling reversibility in higher-order pi. In: Proceedings of CONCUR 2011, vol. 6901 of LNCS. Springer, New York, pp. 297–311 (2011)
24. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Reversing higher-order pi. In: Proceedings of CONCUR 2010, vol. 6269 of LNCS. Springer, New York, pp. 478–493 (2010)
25. Lanese, I., Mezzina, C.A., Stefani, J.-B.: Controlled reversibility and compensations. In: Proceedings of Reversible Computation 2012, vol. 7581 of LNCS. Springer, New York, pp. 240–246 (2012)
26. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theor. Comput. Sci. **13**, 85–108 (1981)
27. Philippou, A., Psara, K.: Reversible computation in Petri nets. In: Proceedings of Reversible Computation (2018) (**to appear**)

28. Phillips, I.C.C., Ulidowski, I.: Reversing algebraic process calculi. In: Proceedings of FOSSACS 2006, vol. 3921 of LNCS. Springer, New York, pp. 246–260 (2006)
29. Phillips, I.C.C., Ulidowski, I.: Reversing algebraic process calculi. J. Logic Algebraic Program. **73**, 70–96 (2007)
30. Phillips, I.C.C., Ulidowski, I.: A hierarchy of reverse bisimulations on stable configuration structures. Math. Struct. Comput. Sci. **22**, 333–372 (2012)
31. Phillips, I.C.C., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. In: Proceedings of CONCUR 2013, vol. 8052 of LNCS. Springer, New York, pp. 303–318 (2013)
32. Phillips, I.C.C., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. J. Logic Algebraic Methods Program. **84**(6), 781–805 (2015)
33. Phillips, I.C.C., Ulidowski, I.: Event identifier logic. Math. Struct. Comput. Sci. **24**, 1–51 (2014)
34. Phillips, I.C.C., Ulidowski, I., Yuen, S.: A reversible process calculus and the modelling of the ERK signalling pathway. In: Proceedings of Reversible Computation 2012, vol. 7581 of LNCS. Springer, New York, pp. 218–232 (2013)
35. Phillips, I.C.C., Ulidowski, I., Yuen, S.: Modelling of bonding with processes and events. In: Proceedings of Reversible Computation 2013, vol. 7948 of LNCS. Springer, New York, pp. 141–154 (2013)
36. Ulidowski, I., Phillips, I.C.C., Yuen, S.: Concurrency and reversibility. In: Proceedings of Reversible Computation 2014, vol. 8507 of LNCS. Springer, New York, pp. 1–14 (2014)
37. Varacca, D., Yoshida, N.: Typed event structures and the linear π-calculus. Theor. Comput. Sci. **411**(19), 1949–1973 (2010)
38. Winskel, G.: Event structures. In: Advances in Petri Nets 1986, vol. 255 of LNCS. Springer, New York, pp. 325–392 (1987)
39. Winskel, G.: Events, causality and symmetry. Comput. J. **54**(1), 42–57 (2011)
40. Yokoyama, T., Axelsen, H.B., Glück, R.: Reversible flowchart languages and the structured reversible program theorem. In: Proceedings of ICALP 2008, vol. 5126 of LNCS. Springer, New York, pp. 258–270 (2008)