Event structure semantics of (controlled) reversible CCS

Eva Graversen, Iain Phillips, and Nobuko Yoshida

Imperial College London

Abstract. CCSK is a reversible form of CCS which is causal, meaning that actions can be reversed if and only if each action caused by them has already been reversed; there is no control on whether or when a computation reverses. We propose an event structure semantics for CCSK. For this purpose we define a category of reversible bundle event structures, and use the causal subcategory to model CCSK. We then modify CCSK to control the reversibility with a rollback primitive, which reverses a specific action and all actions caused by it. To define the event structure semantics of rollback, we change our reversible bundle event structures by making the conflict relation asymmetric rather than symmetric, and we exploit their capacity for non-causal reversibility.

1 Introduction

Reversible process calculi have been studied in works such as [5, 7, 10, 19]. One feature of such reversible processes is their ability to distinguish true concurrency in a way forward-only processes cannot [15]. For instance, using CCS notation, the processes $a|b$ and $a.b + b.a$ are equivalent under interleaving semantics; however in a reversible setting we can distinguish them by noting that $a|b$ allows us to perform a followed by b and then to reverse a, which is impossible for $a.b + b.a$. This motivates us to use event structures [14] to describe truly concurrent semantics of a reversible process calculus.

Two reversible forms of CCS have been proposed: RCCS [7] and CCSK [19]. RCCS creates separate memories to store past (executed) actions, while CCSK annotates past actions with keys within the processes themselves. We formulate an event structure semantics for CCSK rather than RCCS, since the semantics for past and future actions can be defined in a similar manner, rather than having to encompass both processes and memories. We note that Medić and Mezzina [12] showed that RCCS and CCSK can be encoded in each other, meaning one can use their encoding in conjunction with our event structure semantics to obtain an event structure semantics for RCCS.

Event structures have been used for modelling forward-only process calculi [2,4,21]. Cristescu et al. [6] used rigid families [3], related to event structures, to describe the semantics of Rπ [5]. However, their semantics requires a process to first reverse all actions to find the original process, map this process to a rigid family, and then apply each of the reversed memories in order to reach the current state of the process. Aubert and Cristescu [1] used a similar approach to describe the semantics of RCCS processes without auto-concurrency, auto-conflict, or recursion as configuration structures. By contrast, we map a CCSK process (with auto-concurrency, auto-conflict, and recursion) with past actions directly to a (reversible) event structure in a strictly denotational fashion.

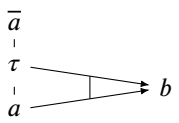
Reversible forms of prime [16], asymmetric [16], and general [18] event structures have already been defined, but the usual way of doing parallel composition of forward-only prime (PES) and asymmetric event structures (AES) [20] does not translate into a reversible setting, and general event structures are more expressive that is necessary for modelling reversible CCSK. We therefore base our semantics on a reversible form of bundle event structures (BESs) [11].

BESs were created with the specific purpose of allowing the same event to have multiple conflicting causes, thereby making it possible to model parallel composition without creating multiple copies of events. They do this by associating events with bundles of conflicting events, $X \mapsto e$, where in order for event e to happen one of the events of X must have already happened.

This approach can be used for modelling cases such as Example 1.1 below, where an action a has multiple options for synchronisation, either of which would allow the process to continue with the action b. If we model each synchronisation or lack thereof as a separate event then we clearly need to let b have multiple possible causes, which we can accomplish using BESs, but not using PESs.

Example 1.1 (Process easily representable by a bundle event structure).

The CCS process $a.b \mid \overline{a}$ can be described by a BES with the events $a, \tau, \overline{a}, b$, the bundle $\{a, \tau\} \mapsto b$, and the conflicts $a \sharp \tau$ and $\overline{a} \sharp \tau$. The process cannot be represented by a PES or AES without splitting some events into multiple events, due to b having multiple possible causes.



We therefore define a category of reversible BESs (RBESs). Since the reversibility allowed in CCSK (as in RCCS) is *causal*, meaning that actions can be reversed if and only if every action caused by them has already been reversed, we use the causal subcategory of RBESs for defining a denotational semantics of CCSK.

Causal reversibility has the drawback of allowing a process to get into a loop doing and undoing the same action indefinitely; there is no control on whether or when a computation reverses. We modify CCSK to control reversibility by adding the *rollback* introduced for Roll-π in [9]. In Roll-CCSK every action receives a tag γ, and the process only reverses when reaching a $\mathsf{roll}\ \gamma$ primitive, upon which the action tagged with γ, together with all actions caused by it, are reversed. As in Roll-π, the rollback in Roll-CCSK is *maximally permissive*, meaning that any subset of reached rollbacks may be executed, even if one of them rolls back the actions leading to another. The operational semantics of rollback work somewhat differently in Roll-CCSK from Roll-π, since Roll-π has a set of memories describing past actions in addition to a π-calculus process, while CCSK has the past actions incorporated into the structure of the process, meaning that it is harder to know whether one has found all the actions necessary to reverse. Roll-CCSK allows recursion using binding on tags. Mezzina and Koutava [13] added rollback to a variant of CCS, though they use a set of memories to store their past actions, making their semantics closer to Roll-π.

Once a $\mathsf{roll}\ \gamma$ event has happened, we need to ensure that not only are the events caused by the γ-tagged action a_γ able to reverse, but they cannot re-occur until the rollback is complete, at which point the $\mathsf{roll}\ \gamma$ event is reversed. This requires us to model

asymmetric conflict between $\mathsf{roll}\ \gamma$ and events caused by a_γ (apart from $\mathsf{roll}\ \gamma$ itself). Asymmetric conflict is allowed in *extended* BESs (EBESs) [11]. We define a category of reversible EBESs (REBESs) and use them to give an event structure semantics of rollback. Note that we do not restrict ourselves to the causal subcategory of REBESs, since reversibility in Roll-CCSK is not necessarily causal. An action a_γ tagged with γ is a cause of $\mathsf{roll}\ \gamma$, but we want a_γ to reverse before $\mathsf{roll}\ \gamma$ does.

Contributions We formulate reversible forms of bundle, and extended bundle event structures. We show that these form categories equipped with products and coproducts. We extend CCSK with recursion and use the category of RBESs to define its event structure semantics. We define the operational semantics of Roll-CCSK, which uses rollback to control the reversibility in CCSK, showing that our rollbacks are both sound (Theorem 6.6) and complete (Theorem 6.9) with respect to CCSK. We use the category of REBESs to define the event structure semantics of Roll-CCSK. We prove operational correspondence between the operational semantics and event structure semantics of both CCSK and Roll-CCSK (Theorems 4.10, 7.5 and 7.7).

Outline Section 2 recalls the semantics of CCSK. Section 3 describes RBESs and their category. Section 4 defines the event structure semantics of CCSK. Section 5 describes REBESs and their category. Section 6 introduces Roll-CCSK and its operational semantics and Section 7 uses REBESs to describe the event structure semantics of Roll-CCSK.

2 CCSK

CCSK was defined in [19], and distinguishes itself from most reversible process calculi by retaining the structure of the process when actions are performed, and annotating past actions with keys instead of generating memories. For instance we have $a.P \mid \overline{a}.Q \xrightarrow{\tau[n]} a[n].P \mid \overline{a}[n].Q$, with the key n denoting that a and $\overline{a}$ have previously communicated, and we therefore cannot reverse one without reversing the other.

We call the set of actions of CCSK $\mathcal{A}$ and let a, b, c range over $\mathcal{A}$, α, β range over $\mathcal{A} \cup \overline{\mathcal{A}}$, and μ range over $\mathcal{A} \cup \overline{\mathcal{A}} \cup \{\tau\}$. We let $\mathcal{K}$ be an infinite set of communication keys and let m, n range over $\mathcal{K}$.

CCSK then has the following syntax, very similar to CCS:

$$P ::= \alpha.P \mid \alpha[n].P \mid P_0 + P_1 \mid P_0 | P_1 \mid P \setminus A \mid P[f] \mid 0$$

Here $P \setminus A$ restricts communication on actions in $A \cup \overline{A}$ and $P[f]$ applies a function $f : \mathcal{A} \to \mathcal{A}$ to actions done by P.

Table 1 shows the forwards rules of the operational semantics of CCSK. As CCSK is causal, the reverse rules can be derived from these. We use $\rightsquigarrow$ to denote a reverse action, $\mathsf{std}(P)$ to denote that P is a standard process, meaning it contains no past actions, and $\mathsf{fsh}[n](P)$ to denote that the key n is fresh for P. The rules are slightly reformulated compared to [19] in that we use structural congruence $\equiv$. The rules for structural congruence are:

$P \mid 0 \equiv P$	$P_0 \mid P_1 \equiv P_1 \mid P_0$	$P_0 \mid (P_1 \mid P_2) \equiv (P_0 \mid P_1) \mid P_2$
$P + 0 \equiv P$	$P_0 + P_1 \equiv P_1 + P_0$	$P_0 + (P_1 + P_2) \equiv (P_0 + P_1) + P_2$

Table 1. Forwards semantics of CCSK [17]

$$\frac{\mathsf{std}(P)}{\alpha.P \xrightarrow{\alpha[n]} \alpha[n].P} \qquad \frac{P \xrightarrow{\mu[m]} P' \quad m \neq n}{\alpha[n].P \xrightarrow{\mu[m]} \alpha[n].P'} \qquad \frac{P \equiv Q \xrightarrow{\mu[n]} Q' \equiv P'}{P \xrightarrow{\mu[n]} P'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{fsh}[n](P_1)}{P_0 \mid P_1 \xrightarrow{\mu[n]} P_0' \mid P_1} \qquad \frac{P_0 \xrightarrow{\alpha[n]} P_0' \quad P_1 \xrightarrow{\overline{\alpha}[n]} P_1'}{P_0 \mid P_1 \xrightarrow{\tau[n]} P_0' \mid P_1'}$$

$$\frac{P_0 \xrightarrow{\mu[n]} P_0' \quad \mathsf{std}(P_1)}{P_0 + P_1 \xrightarrow{\mu[n]} P_0' + P_1} \qquad \frac{P \xrightarrow{\mu[n]} P' \quad \mu, \overline{\mu} \notin A}{P \setminus A \xrightarrow{\mu[n]} P' \setminus A} \qquad \frac{P \xrightarrow{\mu[n]} P'}{P[f] \xrightarrow{f(\mu)[n]} P'[f]}$$

We extend CCSK with recursion as follows. We add process constants $A\left\langle \tilde{b} \right\rangle$, together with definitions $A(\tilde{a}) = P_A$, where P_A is a standard process and $\tilde{a}$ is a tuple containing the actions of P_A. This leads us to expand our definition of structural congruence with $A\left\langle \tilde{b} \right\rangle \equiv P_A\{^{\tilde{b}}/_{\tilde{a}}\}$.

Definition 2.1. *A process P is* reachable *if there exists a standard process Q such that $Q(\rightarrow \cup \rightsquigarrow)^* P$, and* forwards-reachable *if there exists a standard process Q such that $Q \rightarrow^* P$.*

Since CCSK is causal all reachable processes are forwards-reachable ([19], Proposition 5.15; the proof still applies with recursion added).

3 Reversible Bundle Event Structures

Bundle event structures (BES) [11] extend prime event structures by allowing multiple possible causes for the same event. They do this by replacing the causal relation with a bundle set, so that if $X \mapsto e$ then exactly one of the events in X must have happened before e can happen, and all the events in X must be in conflict.

We define reversible bundle event structures (RBES) by extending the bundle relation to map to reverse events, denoted $\underline{e}$, and adding a prevention relation, such that if $e \rhd \underline{e'}$ then e' cannot be reversed from configurations containing e. We use e^* to denote either e or $\underline{e}$.

Definition 3.1 (Reversible Bundle Event Structure). *A reversible bundle event structure is a 5-tuple $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ where:*

1. *E is the set of events;*
2. *$F \subseteq E$ is the set of reversible events;*
3. *the bundle set, $\mapsto \subseteq 2^E \times (E \cup \underline{F})$, satisfies $X \mapsto e^* \Rightarrow \forall e_1, e_2 \in X. e_1 \neq e_2 \Rightarrow e_1 \sharp e_2$ and for all $e \in F$, $\{e\} \mapsto \underline{e}$;*
4. *the conflict relation, $\sharp \subseteq E \times E$, is symmetric and irreflexive;*
5. *$\rhd \subseteq E \times \underline{F}$ is the prevention relation.*

In order to obtain a category of RBESs, we define a morphism in Definition 3.2.

Definition 3.2 (RBES-morphism). *Given RBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$, an RBES-morphism from $\mathcal{E}_0$ to $\mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that $f(F_0) \subseteq F_1$ and for all $e, e' \in E_0$:*

1. *if $f(e) \sharp_1 f(e')$ then $e \sharp_0 e'$;*
2. *if $f(e) = f(e')$ and $e \neq e'$ then $e \sharp_0 e'$;*
3. *for $X_1 \subseteq E_1$ if $X_1 \mapsto_1 f(e)^*$ then there exists $X_0 \subseteq E_0$ such that $X_0 \mapsto_0 e^*$, $f(X_0) \subseteq X_1$, and if $e' \in X_0$ then $f(e') \neq \bot$;*
4. *if $f(e) \rhd_1 \underline{f(e')}$ then $e \rhd_0 \underline{e'}$.*

It can be checked that RBESs with this notion of morphism form a category **RBES**. We define a product of RBESs in Definition 3.3. A coproduct can also be defined similarly to other coproducts of event structures.

Definition 3.3 (Product of RBESs). *Let $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$ be reversible bundle event structures. Their product $\mathcal{E}_0 \times \mathcal{E}_1$ is the RBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ with projections π_0 and π_1 where:*

1. *$E = E_0 \times_* E_1 = \{(e, *) \mid e \in E_0\} \cup \{(*, e) \mid e \in E_1\} \cup \{(e, e') \mid e \in E_0 \text{ and } e' \in E_1\}$;*
2. *$F = F_0 \times_* F_1 = \{(e, *) \mid e \in F_0\} \cup \{(*, e) \mid e \in F_1\} \cup \{(e, e') \mid e \in F_0 \text{ and } e' \in F_1\}$;*
3. *for $i \in \{0, 1\}$ we have $(e_0, e_1) \in E$, $\pi_i((e_0, e_1)) = e_i$;*
4. *for any $e^* \in E \cup \underline{F}$, $X \subseteq E$, $X \mapsto e^*$ iff there exists $i \in \{0, 1\}$ and $X_i \subseteq E_i$ such that $X_i \mapsto \pi_i(e)^*$ and $X = \{e' \in E \mid \pi_i(e') \in X_i\}$;*
5. *for any $e, e' \in E$, $e \sharp e'$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \sharp_i \pi_i(e')$, or $\pi_i(e) = \pi_i(e') \neq \bot$ and $\pi_{1-i}(e) \neq \pi_{1-i}(e')$;*
6. *for any $e \in E$, $e' \in F$, $e \rhd \underline{e'}$ iff there exists $i \in \{0, 1\}$ such that $\pi_i(e) \rhd_i \underline{\pi_i(e')}$.*

We wish to model RBESs as configuration systems (CSs), and therefore define a functor from one category to the other in Definition 3.5. A CS consists of a set of events, some of which are reversible, configurations of these events, and labelled transitions between them, as described in Definition 3.4. We will later use the CSs corresponding to our event structure semantics to describe the operational correspondence between our event structure semantics and the operational semantics of CCSK.

Definition 3.4 (Configuration system [16]). *A* configuration system *(CS) is a quadruple $\mathcal{C} = (E, F, \mathsf{C}, \to)$ where E is a set of events, $F \subseteq E$ is a set of reversible events, $\mathsf{C} \subseteq 2^E$ is the set of configurations, and $\to \subseteq \mathsf{C} \times 2^{E \cup \underline{F}} \times \mathsf{C}$ is a labelled transition relation such that if $X \xrightarrow{A \cup \underline{B}} Y$ then:*

- *$X, Y \in \mathsf{C}$, $A \cap X = \emptyset$; $B \subseteq X \cap F$; and $Y = (X \setminus B) \cup A$;*
- *for all $A' \subseteq A$ and $B' \subseteq B$, we have $X \xrightarrow{A' \cup \underline{B'}} Z \xrightarrow{(A \setminus A') \cup \underline{(B \setminus B')}} Y$, meaning $Z = (X \setminus B') \cup A' \in \mathsf{C}$.*

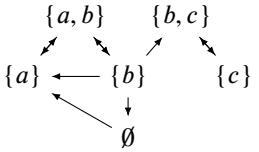
Definition 3.5 (From RBES to CS). *The functor C_{br} : **RBES** $\to$ **CS** is defined as:*

1. *$C_{br}((E, F, \mapsto, \sharp, \rhd)) = (E, F, \mathsf{C}, \to)$ where:*
   (a) $X \in \mathsf{C}$ if X is conflict-free;

(b) *For $X, Y \in \mathsf{C}$, $A \subseteq E$, and $B \subseteq F$, there exists a transition $X \xrightarrow{A \cup \underline{B}} Y$ if:*
   i. *$Y = (X \setminus B) \cup A$; $X \cap A = \emptyset$; $B \subseteq X$; and $X \cup A$ conflict-free;*
   ii. *for all $e \in B$, if $e' \rhd \underline{e}$ then $e' \notin X \cup A$;*
   iii. *for all $e \in A$ and $X' \subseteq E$, if $X' \mapsto e$ then $X' \cap (X \setminus B) \neq \emptyset$;*
   iv. *for all $e \in B$ and $X' \subseteq E$, if $X' \mapsto \underline{e}$ then $X' \cap (X \setminus (B \setminus \{e\})) \neq \emptyset$.*

2. *$C_{br}(f) = f$.*

Example 3.6 shows an RBES mapped to a CS. The configuration $\{b, c\}$ is reachable despite b being required for c to happen and c being a possible cause of b.



Example 3.6 (RBES). An RBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ where $E = \{a, b, c\}$, $F = \{a, b\}$, $a \sharp c$, $\{a, c\} \mapsto b$, $\{b\} \mapsto c$ $\{a\} \mapsto \underline{a}$, $\{b\} \mapsto \underline{a}$, and $\{b\} \mapsto \underline{b}$, gives the CS $C_{br}(\mathcal{E})$.

We define a causal variant of RBESs in Definition 3.7. The subcategory **CRBES** consists of CRBESs and the RBES-morphisms between them.

Definition 3.7 (Causal RBES). *$\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a causal RBES (CRBES) if (1) if $e \rhd \underline{e'}$ then either $e \sharp e'$ or there exists an $X \subseteq E$ such that $X \mapsto e$ and $e' \in X$, (2) if $X \mapsto e$ and $e' \in X \cap F$, then $e \rhd \underline{e'}$, and (3) if $X \mapsto \underline{e}$ then $e \in X$.*

Proposition 3.8.

1. *Given a CRBES, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ and corresponding CS $C_{rb}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, any reachable $X \in \mathsf{C}$ is forwards-reachable.*
2. *If $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd)$ is a CRBES and $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$ then whenever $X \in \mathsf{C}$, $X \xrightarrow{A \cup \underline{B}} Y$ and $A \cup B \subseteq F$, we get a transition $Y \xrightarrow{B \cup \underline{A}} X$.*

Since our motivation for defining RBESs was modelling reversible processes, we need to be able to label our events with a corresponding action from a process. For this we use a *labelled RBES* (LRBES).

Definition 3.9 (Labelled Reversible Bundle Event Structure). *An LRBES $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ consists of an RBES $(E, F, \mapsto, \sharp, \rhd)$, a set of labels Act, and a surjective labelling function $\lambda : E \to \mathsf{Act}$.*

Definition 3.10 (LRBES-morphism). *Given LRBESs $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ and $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, an* LRBES-morphism *$f : \mathcal{E}_0 \to \mathcal{E}_1$ is a partial function $f : E_0 \to E_1$ such that $f : (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) \to (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$ is an RBES-morphism and for all $e \in E_0$, either $f(e) = \bot$ or $\lambda_0(e) = \lambda_1(f(e))$.*

4 Event Structure Semantics of CCSK

Having defined RBESs, we will now use them to describe the semantics of CCSK [19]. Unlike the event structure semantics of CCS [2, 21], our semantics will generate both an event structure and an *initial configuration* containing all the events corresponding to past actions. This means that if $P \to P'$ then P and P' will be described by the same event structure with different initial states.

First we define the operators we will use in the semantics, particularly restriction, parallel composition, choice, and action prefixes. Restriction is achieved by simply removing any events associated with the restricted action.

Definition 4.1 (Restriction). *Given an LRBES,* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$*, restricting* $\mathcal{E}$ *to* $E' \subseteq E$ *creates* $\mathcal{E} \upharpoonright E' = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ *where:*

1. $F' = F \cap E'$;
2. $\mapsto' = \mapsto \cap (\mathcal{P}(E') \times (E' \cup \underline{F'}))$;
3. $\sharp' = \sharp \cap (E' \times E')$;
4. $\rhd' = \rhd \cap (E' \times \underline{F'})$;
5. $\lambda' = \lambda \upharpoonright_{E'}$;
6. Act *is the range of* λ.

Parallel composition uses the product of RBESs, labels as τ any event corresponding to a synchronisation, and removes any invalid events describing an impossible synchronisation.

Definition 4.2 (Parallel). *Given LRBESs* $\mathcal{E}_0$ *and* $\mathcal{E}_1$, $\mathcal{E}_0 || \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act}) \upharpoonright \{e \mid \lambda(e) \neq 0\}$ *where:* $(E, F, \mapsto, \sharp, \rhd) = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) \times (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$;

$$\lambda(e) = \begin{cases} \lambda_0(e_0) & \text{if } e = (e_0, *) \\ \lambda_1(e_1) & \text{if } e = (*, e_1) \\ \tau & \text{if } e = (e_0, e_1) \text{ and } \lambda_0(e_0) = \overline{\lambda_1(e_1)} \\ 0 & \text{if } e = (e_0, e_1) \text{ and } \lambda_0(e_0) \neq \overline{\lambda_1(e_1)}; \end{cases}$$

and $\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1 \cup \{0, \tau\}$.

Choice, which acts as a coproduct of LRBESs, simply uses the coproduct of RBESs, and defines the labels as expected.

Definition 4.3 (Choice). *Given LRBESs* $\mathcal{E}_0$ *and* $\mathcal{E}_1$, $\mathcal{E}_0 \& \mathcal{E}_1 = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$ *where:* $(E, F, \mapsto, \sharp, \rhd) = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0) + (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1)$; $\lambda(i_j(e)) = \lambda_j(e)$; *and* $\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1$.

Causally prefixing an action onto an event structure means the new event causes all other events and is prevented from reversing by all other events.

Definition 4.4 (Causal Prefix). *Given an LRBES* $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$*, an event* $e \notin E$*, and a label* α, $\alpha(e).\mathcal{E} = (E', F', \mapsto', \sharp', \rhd', \lambda', \mathsf{Act}')$ *where:*

1. $E' = E \cup \{e\}$;
2. $F' = F \cup \{e\}$;
3. $\mapsto' = \mapsto \cup (\{\{e\}\} \times (E \cup \{\underline{e}\}))$;
4. $\sharp' = \sharp$;
5. $\rhd' = \rhd \cup (E \times \{\underline{e}\})$;
6. $\lambda' = \lambda[e \mapsto \alpha]$;
7. $\mathsf{Act}' = \mathsf{Act} \cup \{\alpha\}$.

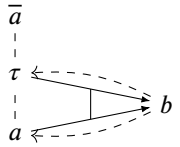
Now that we have defined the main operations of the process calculus, we define the event structure semantics in Table 2. We do this using rules of the form $[\![P]\!]_l = \langle \mathcal{E}, \mathsf{Init}, k \rangle$ wherein l is the level of unfolding, which we use to model recursion, $\mathcal{E}$ is an LRBES, Init is the initial configuration, and $k : \mathsf{Init} \to \mathcal{K}$ is a function assigning communication keys to the past actions, which we use in parallel composition to determine which synchronisations of past actions to put in Init.

Table 2. RBES-semantics of CCSK

$$
\begin{array}{ll}
\{\!|0|\!\}_l = & \langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle \\
\{\!|P_0+P_1|\!\}_l = & \langle \mathcal{E}_0 \& \mathcal{E}_1, \mathsf{Init}, k\rangle \text{ where} \\
& \text{For } i\in\{0,1\},\ \{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle \\
& \mathsf{Init} = \{(j,e) \mid j\in\{0,1\} \text{ and } e\in \mathsf{Init}_j\} \\
& k((j,e)) = k_j(e) \text{ if } e\in \mathsf{Init}_j \\
\{\!|\alpha.P|\!\}_l = & \left\langle \alpha(e).(E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}, k\right\rangle \text{ for } e \text{ fresh for } E \text{ where} \\
& \{\!|P|\!\}_l = \left\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}, k\right\rangle \\
\{\!|\alpha[m].P|\!\}_l = & \left\langle \alpha(e).(E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}\cup\{e\}, k[e\mapsto m]\right\rangle \text{ for } e \text{ fresh for } E \text{ where} \\
& \{\!|P|\!\}_l = \left\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}, k\right\rangle \\
\{\!|P_0 \mid P_1|\!\}_l = & \left\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}, k\right\rangle \text{ where} \\
& \text{For } i\in\{0,1\},\ \{\!|P_i|\!\}_l = \langle \mathcal{E}_i, \mathsf{Init}_i, k_i\rangle \\
& (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}) = \mathcal{E}_0 || \mathcal{E}_1 \\
& \mathsf{Init} = \{(e_0,e_1) \mid e_0\in\mathsf{Init}_0, e_1\in\mathsf{Init}_1, k_0(e_0) = \overline{k_1(e_1)}\}\cup \\
& \{(*,e_1) \mid e_1\in\mathsf{Init}_1 \text{ and } \nexists e_0\in\mathsf{Init}_0.\lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\}\cup \\
& \{(e_0,*) \mid e_0\in\mathsf{Init}_0 \text{ and } \nexists e_1\in\mathsf{Init}_1.\lambda_0(e_0) = \overline{\lambda_1(e_1)} \text{ and } k_0(e_0) = k_1(e_1)\} \\
& k(e) = \begin{cases} k_0(e_0) & \text{if } e=(e_0,*) \\ k_1(e_1) & \text{if } e=(*,e_1) \\ k_0(e_0) & \text{if } e=(e_0,e_1) \quad -\textit{ note that } k_0(e_0)=k_1(e_1) \end{cases} \\
\{\!|P\setminus A|\!\}_l = & \langle \mathcal{E}\upharpoonright\{e\mid\lambda(e)\notin\mathcal{A}\}, \mathsf{Init}\cap\{e\mid\lambda(e)\notin\mathcal{A}\}, k\upharpoonright\{e\mid\lambda(e)\notin\mathcal{A}\}\rangle \text{ where} \\
& \{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k\rangle \\
& \mathcal{A} = A\cup\overline{A} \\
\{\!|P[f]|\!\}_l = & \left\langle (E,F,\mapsto,\sharp,\rhd,\lambda,\mathsf{Act}), \mathsf{Init}, k\right\rangle \text{ where} \\
& \{\!|P|\!\}_l = \left\langle (E,F,\mapsto,\sharp,\rhd,\lambda',\mathsf{Act}'), \mathsf{Init}, k\right\rangle \\
& \mathsf{Act} = f(\mathsf{Act}') \\
& \lambda = f\circ\lambda' \\
\left\{\!\left|A\left\langle\tilde{b}\right\rangle\right|\!\right\}_0 = & \langle(\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset,\emptyset),\emptyset,\emptyset\rangle \\
\left\{\!\left|A\left\langle\tilde{b}\right\rangle\right|\!\right\}_l = & \left\{\!\left|P_A\{^{\tilde{b}}/_{\tilde{a}}\}\right|\!\right\}_{l-1} \text{ where } A(\tilde{a}) = P_A
\end{array}
$$

Note that the only difference between a future and a past action is that the event corresponding to a past action is put in the initial state and given a communication key.

Example 4.5. The CCSK process $a.b \mid \overline{a}$ (cf. Example 1.1) can be represented by the RBES with events labelled a, $\overline{a}$, τ, and b, the bundle $\{a,\tau\} \mapsto b$, the conflicts $a \sharp \tau$ and $\overline{a} \sharp \tau$, and the preventions $b \rhd \underline{a}$ and $b \rhd \underline{\tau}$.



We say that $\{\!|P|\!\} = \sup_{l\in\mathcal{N}} \{\!|P|\!\}_l$. This means we need to show that there exists such a least upper bound of the levels of unfolding. As shown in [8], ordering closed BESs by restriction produces a complete partial order. Since our LRBESs do not have overlapping bundles ($X \mapsto e^*$ and $X' \mapsto e^*$ implies $X \neq X'$ or $X \cap X' = \emptyset$) they are closed, and we can use a similar ordering.

Definition 4.6 (Ordering of LRBESs). *Given LRBESs* $\mathcal{E}_0 = (E_0, F_0, \mapsto_0, \sharp_0, \rhd_0, \lambda_0, \mathsf{Act}_0)$ *and* $\mathcal{E}_1 = (E_1, F_1, \mapsto_1, \sharp_1, \rhd_1, \lambda_1, \mathsf{Act}_1)$, $\mathcal{E}_0 \leq \mathcal{E}_1$ *if* $\mathcal{E}_0 = \mathcal{E}_1 \upharpoonright E_0$.

Proposition 4.7 (Unfolding). *Given a reachable process* P *and a level of unfolding* l*, if* $\{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ *and* $\{\!|P|\!\}_{l-1} = \left\langle \mathcal{E}', \mathsf{Init}', k'\right\rangle$*, then* $\mathcal{E}' \leq \mathcal{E}$*,* $\mathsf{Init} = \mathsf{Init}'$*, and* $k = k'$.

In order to prove that our event structure semantics correspond with the operational semantics for CCSK defined in [17] we first show that our event structures are causal.

Proposition 4.8. *Given a process P such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, $\mathcal{E}$ is causal.*

Structurally congruent processes will generate isomorphic event structures:

Proposition 4.9 (Structural Congruence). *Given processes P and P', if $P \equiv P'$, $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, and $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(\mathsf{Init}) = \mathsf{Init}'$ and for all $e \in \mathsf{Init}$, $k(e) = k'(f(e))$.*

Finally we show in Theorem 4.10 that given a process P with a conflict-free initial state, including any reachable process, there exists a transition $P \xrightarrow{\mu} P'$ if and only if the event structure corresponding to P is isomorphic to the event structure corresponding to P' and an event e labelled μ exists such that e is available in P's initial state, and P''s initial state is P's initial state with e added.

Theorem 4.10. *Let P be a process with $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$, $\mathcal{E} = (E, F, \mapsto, \sharp, \rhd, \lambda, \mathsf{Act})$, $C_{br}(\mathcal{E}) = (E, F, \mathsf{C}, \to)$, and Init conflict-free. Then*

1. *if there exists a P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \xrightarrow{\mu[m]} P'$ then there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$;*
2. *and if there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ then there exists a P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \xrightarrow{\mu[m]} P'$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

Corollary 4.11. *Let P be a process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k \rangle$. Then Init is forwards-reachable in $\mathcal{E}$ if and only if there exists a standard process Q such that $Q \to^* P$.*

Since we showed in Proposition 4.8 that any event structures generated by processes are causal, it follows that we get a similar correspondence between the reverse transitions of processes and event structures.

5 Reversible Extended Bundle Event Structures

In CCSK a process can reverse actions at any time. Suppose that we wish to control this reversibility by having a 'rollback' action that causes all actions, or all actions since the last safe state, to be reversed before the process can continue, similar to the roll command of [9]. RBESs can easily ensure that this rollback event roll is required for other events to reverse; we simply say that $\{\mathsf{roll}\} \mapsto \underline{e}$ for all e. However, preventing events from happening during the roll in RBESs requires symmetric conflict, which would mean the other events also prevent roll from occurring. To solve a similar problem, Phillips and Ulidowski [16] use reversible asymmetric event structures, which replace symmetric conflict with asymmetric. But since these use the same notion of causality

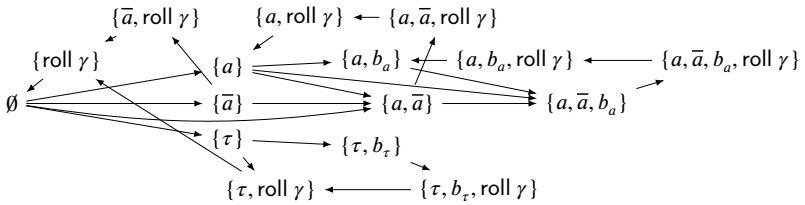


Fig. 1. The reachable configurations of the REBES described in Example 5.1

as reversible prime event structures, they have trouble modelling concurrent processes with synchronisation, as shown in Example 1.1.

Extended bundle event structures (EBES) [11] add asymmetric conflict; so defining a reversible variant of these will allow us to model the above scenario.

Example 5.1 (The necessity of REBESs for modelling rollback). Consider $a.b \mid \overline{a}_\gamma.\mathsf{roll}\ \gamma$, where $\mathsf{roll}\ \gamma$ means undo the action labelled γ, that is $\overline{a}$, and everything caused by it before continuing. To model this we would need to expand the RBES from Example 4.5 with a new event $\mathsf{roll}\ \gamma$, and split b into two different events depending on whether it needs to be reversed during the rollback or not. This would give us an RBES $(\{a, \tau, \overline{a}, b_a, b_\tau, \mathsf{roll}\ \gamma\}, \{a, \tau, \overline{a}, b_a, b_\tau, \mathsf{roll}\ \gamma\}, \mapsto, \sharp, \rhd)$ where $\{a\} \mapsto b_a$, $\{\tau\} \mapsto b_\tau$, $\{\overline{a}, \tau\} \mapsto \mathsf{roll}\ \gamma$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{\tau}$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{\overline{a}}$, $\{\mathsf{roll}\ \gamma\} \mapsto \underline{b_\tau}$, $a \sharp \tau$, $\overline{a} \sharp \tau$, $b_a \rhd \underline{a}$, $b_\tau \rhd \underline{\tau}$, $\overline{a} \rhd \underline{\mathsf{roll}\ \gamma}$, and $\tau \rhd \underline{\mathsf{roll}\ \gamma}$. This would indeed ensure that $\overline{a}$ and the events caused by it could only reverse if one of the roll events had occurred, but it would not force them to do so before doing anything else. For this we use asymmetric conflict: $\mathsf{roll}\ \gamma \rhd \overline{a}$, $\mathsf{roll}\ \gamma \rhd \tau$, $\mathsf{roll}\ \gamma \rhd b_\tau$, giving us a CS with the *reachable configurations* shown in Figure 1.

We define a reversible version of EBESs in Definition 5.2, treating the asymmetric conflict similarly to RAESs in [16].

Definition 5.2 (Reversible Extended Bundle Event Structure). *An REBES is a 4-tuple $\mathcal{E} = (E, F, \mapsto, \rhd)$ where:*

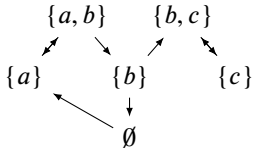
1. *E is the set of events;*
2. *$F \subseteq E$ is the set of reversible events;*
3. *$\mapsto \subseteq 2^E \times (E \cup \underline{F})$ is the bundle set, satisfying $X \mapsto e \Rightarrow \forall e_1, e_2 \in X.(e_1 \neq e_2 \Rightarrow e_1 \rhd e_2)$, and for all $e \in F$, $\{e\} \mapsto \underline{e}$;*
4. *$\rhd \subseteq E \times (E \cup \underline{F})$ is the asymmetric conflict relation, which is irreflexive.*

In order to define REBES-morphisms, we extend the RBES morphism in the obvious way, letting the condition on preventions also apply to prevention on forwards events. This gives us a category **REBES**, in which we can define products and coproducts much like we did for RBESs, treating asymmetric conflict the same as we did symmetric.

We again model REBESs as CSs, defining configurations as sets of events on which $\rhd$ is well-founded, and extending the requirements of prevention in transitions to forwards events.

Example 5.3 shows an REBES, which cannot be represented by an RBES, since we get a transition $\emptyset \to \{a\}$, but no $\{b\} \to \{a, b\}$, despite $\{a, b\}$ being a configuration.

Example 5.3 (REBES). An REBES $\mathcal{E} = (E, F, \mapsto, \rhd)$ where $E = \{a, b, c\}$, $F = \{a, b\}, \{a, c\} \mapsto b$, $\{b\} \mapsto c$ $\{a\} \mapsto \underline{a}$, $\{b\} \mapsto \underline{a}$, $\{b\} \mapsto \underline{b}$, $a \rhd c$, $c \rhd a$, and $b \rhd a$ gives the CS $C_{er}(\mathcal{E})$ in the diagram.



Since we are using our REBESs for modelling the semantics of rollback in CCSK, we need a labelled variant, which we can define much as we did labelled RBESs.

6 Roll-CCSK

The operational semantics for roll-π [9] are not translatable directly to CCSK, as they make use of the fact that one can know, when looking at a memory, whether the communication it was associated with was with another process or not, and therefore, for a given subprocess P and a memory m, one knows whether all the memories and subprocesses caused by m are part of P. In CCSK, this is not as easy, as the roll in a subprocess $\alpha_\gamma[n] \ldots \mathsf{roll}\ \gamma$, where γ is a *tag* denoting which rollback rolls back which action, may or may not require rolling back the other end of the α communication, and all actions caused by it. We therefore need to check at every instance of parallel composition whether any communication has taken place, and if so roll back those actions and all actions caused by them. This may include rolling back additional actions from the subprocess containing the roll as in $a[n_1].\overline{b}[n_2] \mid c[n_3].(\overline{a}_\gamma[n_1].\mathsf{roll}\ \gamma \mid b[n_2])$, where it does not become clear that $b[n_2]$ needs to be reversed during the roll until the outer parallel composition. Unlike [9], we therefore do not provide low-level operational semantics for Roll-CCSK, only providing high-level operational semantics in this section, and low-level denotational event structure semantics in Section 7.

The syntax of Roll-CCSK is as follows:

$$P ::= \alpha_\gamma.P \mid \alpha_\gamma[n].P \mid P_0 + P_1 \mid P_0 | P_1 \mid P\backslash A \mid P[f] \mid 0 \mid \mathsf{roll}\ \gamma \mid \mathsf{rolling}\ \gamma \mid (\nu\, \gamma)P$$

Most of the syntax is the same as CCSK and CCS, but adding tags and rolls as described above, and $\mathsf{rolling}\ \gamma$, which denotes a roll in progress, the necessity of which is justified later. From now on we will use $\alpha.P$ to denote $\alpha_\gamma.P$ where no $\mathsf{roll}\ \gamma$ exists in P. Before presenting the operational semantics of rollback, we define causal dependence and projection similarly to [9], on which we base our own semantics.

Definition 6.1 (Causal dependence). *Let P be a process and Γ be the set of tags in P. Then the binary relation $\leq_P$ is the smallest relation satisfying*

- *if there exists a process P' and past actions $\alpha_\gamma[n]$ and $\beta_{\gamma'}[m]$ such that $\alpha_\gamma[n].P'$ is a subprocess of P and $\beta_{\gamma'}[m]$ occurs in P' then $\gamma \leq_P \gamma'$;*
- *if there exist past actions $\alpha_\gamma[n]$ and $\beta_{\gamma'}[n]$ in P with the same keys then $\gamma \leq_P \gamma'$;*
- *$\leq_P$ is reflexively and transitively closed.*

Table 3. The main rules for rollback in the operational semantics of Roll-CCSK

(start ROLL)	$\mathsf{roll}\ \gamma \overset{\text{start roll } \gamma}{\rightsquigarrow} \mathsf{rolling}\ \gamma$	(par ROLL)	$\dfrac{P_0 \overset{\text{roll } \gamma}{\rightsquigarrow} P_0' \quad C = \{\gamma' \mid \gamma \leq_{P_0 \mid P_1} \gamma'\}}{P_0 \mid P_1 \overset{\text{roll } \gamma}{\rightsquigarrow} (P_0 \mid P_1)_{\natural C}}$
(ROLL)	$\mathsf{rolling}\ \gamma \overset{\text{roll } \gamma}{\rightsquigarrow} \mathsf{roll}\ \gamma$	(act ROLL)	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P' \quad C = \{\gamma' \mid \gamma \leq_{\alpha_\gamma[n].P} \gamma'\}}{\alpha_\gamma[n].P \overset{\text{roll } \gamma}{\rightsquigarrow} \alpha_\gamma.P_{\natural C}}$
(bind ROLL)	$\dfrac{P \overset{\text{roll } \gamma}{\rightsquigarrow} P'}{(\nu\ \gamma)P \overset{\text{roll bound}}{\rightsquigarrow} (\nu\ \gamma)P'}$	(bind ROLL struct)	$\dfrac{P \equiv Q \overset{\text{roll bound}}{\rightsquigarrow} Q' \equiv P'}{P \overset{\text{roll bound}}{\rightsquigarrow} P'}$

Definition 6.2 (Projection). *Given a process P and a set of tags C, $P_{\natural C}$ is defined as:*

$(\alpha_\gamma[n].P)_{\natural C} = \alpha_\gamma[n].(P_{\natural C})$ *if* $\gamma \notin C$ $\quad 0_{\natural C} = 0$ $\quad (P \setminus A)_{\natural C} = (P_{\natural C}) \setminus A$

$(\alpha_\gamma[n].P)_{\natural C} = \alpha_\gamma.(P_{\natural C})$ *if* $\gamma \in C$ $\quad \mathsf{roll}\ \gamma_{\natural C} = \mathsf{roll}\ \gamma$ $\quad (P_0 \mid P_1)_{\natural C} = P_{0_{\natural C}} \mid P_{1_{\natural C}}$

$\mathsf{rolling}\ \gamma_{\natural C} = \mathsf{rolling}\ \gamma$ *if* $\gamma \notin C$ $\quad A\left\langle \tilde{b}, \tilde{\gamma} \right\rangle_{\natural C} = A\left\langle \tilde{b}, \tilde{\gamma} \right\rangle$ $\quad (\nu\ \gamma)P_{\natural C} = (\nu\ \gamma)(P_{\natural C})$

$\mathsf{rolling}\ \gamma_{\natural C} = \mathsf{roll}\ \gamma$ *if* $\gamma \in C$ $\quad (\alpha_\gamma.P)_{\natural C} = \alpha_\gamma.(P_{\natural C})$ $\quad (P[f])_{\natural C} = P_{\natural C}[f]$

$(P_0 + P_1)_{\natural C} = P_{0_{\natural C}} + P_{1_{\natural C}}$

Much as in [9] we perform our rollback in two steps, the first triggering the rollback, and the second actually performing the rollback, in order to ensure that we can start multiple rollbacks at the same time. For instance, in the process $(a_\gamma.(\overline{d}.0 \mid c.\mathsf{roll}\ \gamma) \mid b_{\gamma'}.(\overline{c} \mid d.\mathsf{roll}\ \gamma') \mid \overline{a} \mid \overline{b}) \setminus \{a, b, c, d\}$ we will otherwise never be able to roll all the way back to the beginning, as rolling back a_γ will stop us from reaching $\mathsf{roll}\ \gamma'$ and vice versa.

Table 3 shows the most important rules for reversing actions in Roll-CCSK. The remaining rules permit the $\mathsf{roll\ start}\ \gamma$ and $\mathsf{roll}\ \gamma$ to propagate in the same way as actions in CCSK (and past tag bindings), with the exception that in the rule for choice, if one path has already triggered a roll, the other cannot trigger or perform a roll or a forwards action. The semantics of forwards actions are otherwise identical to CCSK, except again propagating past the tag bindings. By contrast, $\mathsf{roll\ bound}\ \gamma$ does not propagate. We extend our process definitions $A(\tilde{a}) = P_A$ to also include a tuple of tags in P_A, giving us $A(\tilde{a}, \tilde{\gamma}) = P_A$, where P_A is a standard process containing no instances of $\mathsf{rolling}\ \gamma$.

Since we want to be able to handle recursion without confusing instances of multiple actions or rollbacks being associated with the same tags, we introduce binding of tags $(\nu\ \gamma)$, which allows us to avoid clashes. We use $\mathsf{ft}(P)$ to denote the free tags of P. To ensure that we cannot perform $\mathsf{roll}\ \gamma$ in $Q \mid (\nu\ \gamma)P$ without rolling back all actions in Q caused by γ, we only have rule (bind ROLL struct) for bound tags, meaning that to roll back a bound tag we must use structural congruence to move it to the outermost layer of the process. This is also why we have the two rules allowing us to move $(\nu\ \gamma)$ from one side of an action with a different tag to the other.

We also change the rule for applying definitions to ensure all tags are fresh for the unfolded process. This is again to prevent the process from unfolding more rollbacks for

a previous action, such as in $a_\gamma.A\langle a,\gamma\rangle$ with $A(b,\delta) = b_\delta.(A\,\langle b,\delta\rangle \mid \mathsf{roll}\ \delta)$, where there would otherwise be confusion about how far back one should roll each time.

Structural congruence for bound tags:

$\alpha_\gamma(\nu\gamma')P \equiv (\nu\gamma')\alpha_\gamma P$ if $\gamma \neq \gamma'$ $\quad$ $\alpha_\gamma[n](\nu\gamma')P \equiv (\nu\gamma')\alpha_\gamma[n]P$ if $\gamma \neq \gamma'$

$((\nu\gamma')P) \mid Q \equiv (\nu\gamma')(P \mid Q)$ if $\gamma \notin \mathsf{ft}(Q)$ $\quad$ $((\nu\gamma')P) + Q \equiv (\nu\gamma')(P + Q)$ if $\gamma \notin \mathsf{ft}(Q)$

$(\nu\gamma')(P \setminus A) \equiv ((\nu\gamma')P) \setminus A$ $\quad$ $(\nu\gamma)(P[f]) \equiv ((\nu\gamma)P)[f]$

$A\left\langle \tilde{b},\tilde{\delta}\right\rangle \equiv (\nu\ \tilde{\delta})P_A\{^{\tilde{b},\tilde{\delta}}/_{\tilde{a},\tilde{\gamma}}\}$ if $A(\tilde{a},\tilde{\gamma}) = P_A$ $\quad$ $(\nu\ \gamma)(\nu\gamma')P \equiv (\nu\ \gamma')(\nu\ \gamma)P$

Example 6.3 (Bound Tags). Consider the process $P = a_\gamma[n].(\nu\ \gamma)b_\gamma.\mathsf{roll}\ \gamma$. This can clearly do the actions $P \xrightarrow{b[m]} a_\gamma[n].(\nu\ \gamma)b_\gamma[m].\mathsf{roll}\ \gamma \xrightarrow{\text{start roll } \gamma} a_\gamma[n](\nu\ \gamma).b_\gamma[m].\mathsf{rolling}\ \gamma$. However, when actually performing the rollback, we need to use the structural congruence rule to α-convert the bound γ into δ and move the binding to before $a_\gamma[n]$ because roll bound does not propagate through $a_\gamma[n]$. Then we can do $a_\gamma[n].(\nu\ \gamma)b_\gamma[m].\mathsf{rolling}\ \gamma \equiv (\nu\ \delta)a_\gamma[n].b_\delta[m].\mathsf{rolling}\ \delta \overset{\text{roll bound}}{\rightsquigarrow} (\nu\ \delta)a_\gamma[n].b_\delta.\mathsf{roll}\ \delta$.

In addition, to ensure every rollback is associated with exactly one action, we define a *consistent process*.

Definition 6.4 (Consistent process). *A Roll-CCSK process P is* consistent *if*

1. *there exists a standard process Q with no subprocess* rolling γ *such that $Q \to^* P$;*
2. *there exists $P' \equiv_\alpha P$, such that*
   (a) *for any tag γ, P' has at most one subprocess* roll γ *or* rolling γ*;*
   (b) *for any tag γ, there exists exactly one α and at most one n such that α_γ or $\alpha_\gamma[n]$ occur in P';*
   (c) *if* roll γ *is a subprocess of P' then there exists an action α and process P'' such that* roll γ *is a subprocess of P'' and either $\alpha_\gamma.P''$ is a subprocess of P' or there exists a key n such that $\alpha_\gamma[n].P''$ is a subprocess of P';*
3. *if $A\left\langle \tilde{b},\tilde{\delta}\right\rangle$ is a subprocess of P defined as $A(\tilde{a},\tilde{\gamma}) = P_A$, then P_A is consistent.*

Proposition 6.5. *Let P be a consistent process, P' be a process, and either $P \equiv P'$, $P \to P'$, or $P \rightsquigarrow P'$. Then P' is consistent.*

We are then ready to prove Theorem 6.6, stating that for consistent subprocesses, any rollback can be undone by a sequence of forwards actions.

Theorem 6.6 (Loop (Soundness)). *Let P_0 and P_1 be consistent processes containing no subprocesses* rolling γ*, and such that $P_0 \overset{\text{start roll } \gamma}{\rightsquigarrow} P_0' \overset{\text{roll bound}}{\rightsquigarrow} P_1$. Then $P_1 \to^* P_0$.*

We will from now on use $\to_{CCSK}$ and $\rightsquigarrow_{CCSK}$ to distinguish CCSK-transitions from Roll-CCSK transitions, which will continue to be denoted by arrows without subscripts. The last thing we need to prove about our rollback operational semantics before moving on to event structure semantics is Theorem 6.9, stating that (1) our rollbacks only reverse the actions caused by the action we are rolling back according to CCSK, and (2) our rollbacks are *maximally permissive*, meaning that any subset of reached rollbacks may be successfully executed.

Definition 6.7 (Transforming Roll-CCSK to CCSK). *We define a function, ϕ, which translates a Roll-CCSK process into CCSK:*

$\phi(\mathsf{roll}\ \gamma) = 0 \quad \phi(\alpha_\gamma[n].P) = \alpha[n].\phi(P) \quad \phi(\alpha_\gamma.P) = \alpha.\phi(P) \quad \phi((\nu\ \gamma)P) = \phi(P)$

ϕ is otherwise homomorphic on the remainder.

Definition 6.8. *Let P be a CCSK process and $T = \{m_0, m_1, \ldots m_n\}$ be a set of keys. We say that $P \rightsquigarrow_T P'$ if there exist actions μ, ν and a tag m such that $P \overset{\mu[m]}{\rightsquigarrow}_{CCSK} P'$ and $\nu[m_i] \leq_P \mu[m]$ for some $m_i \in T$.*

Theorem 6.9 (Completeness). *Let P be a consistent Roll-CCSK process with sub-processes $\alpha_{0_{\gamma_0}}[m_0] \ldots \mathsf{roll}\ \gamma_0$, $\alpha_{1_{\gamma_1}}[m_1] \ldots \mathsf{roll}\ \gamma_1$, $\ldots$, $\alpha_{n_{\gamma_n}}[m_n] \ldots \mathsf{roll}\ \gamma_n$. Then for all $T \subseteq \{m_0, m_1, \ldots m_n\}$, if $\phi(P) \rightsquigarrow_T^* P' \not\rightsquigarrow_T$ then there exists a Roll-CCSK process P'' such that $\phi(P'') = P'$ and $P \rightsquigarrow^* P''$.*

7 Event Structure semantics of Roll-CCSK

Having proved that our rollback semantics behave as intended, we are ready to translate them into event structure semantics in Table 4. We use labelled REBESs.

To model $\mathsf{roll}\ \gamma$ as an event structure, we have two events, one which triggers the roll, labelled $\mathsf{start\ roll}\ \gamma$, and another, $\mathsf{roll}\ \gamma$, which denotes that the roll is in progress, allowing the events caused by the associated action to begin reversing. When prefixing a process P with an action α_γ, we now need to ensure that any action in P, and any $\mathsf{start\ roll}$ associated with such an action, will be reversed by any $\mathsf{roll}\ \gamma$ in P, and that the rollback does not stop, signified by the event labelled $\mathsf{roll}\ \gamma$ being reversed, until those actions have all been reversed.

When composing the LREBESs of two processes, we also create a separate event for each set of causes it might have (Definition 7.1). This allows us to say that an event can be rolled back if it was caused by a communication with one of the events being rolled back, but not if the communication went differently. Consider the process $a_\gamma.\mathsf{roll}\ \gamma \mid \overline{a}.b \mid a_{\gamma'}.\mathsf{roll}\ \gamma'$. In this case we will want b to roll back if (a_γ, a) and $\mathsf{roll}\ \gamma$ have happened, or if $(a_{\gamma'}, a)$ and $\mathsf{roll}\ \gamma'$ have happened, but not if any other combination of the four events has happened, something which bundles cannot express unless b is split into multiple events. In addition, we use the sets of causes to ensure that if e is in e''s set of causes and e_{roll} can cause e to reverse, then e_{roll} can cause e' to reverse.

Definition 7.1. *Given an LREBES, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, the set of possible causes for an event $e \in E$, $\mathsf{cause}(e) = X$, contains minimal sets of events such that if $x \in X$ then:*

1. *if $x' \mapsto e$ then there exists e' such that $x' \cap x = \{e'\}$;*
2. *if $e' \in x$ then there exists $x' \in \mathsf{cause}(e')$ such that $x' \subseteq x$;*
3. *if $e_1, e_0 \in X$ then we cannot have both $e_0 \rhd e_1$ and $e_1 \rhd e_0$.*

When giving the semantics of restriction, we remove not only the actions associated with the restricted labels, but also the actions caused by them. This is because we want the event structures generated by P and $0 \mid P$ always to be isomorphic; if $P = (a.b)\backslash\{a\}$,

we will otherwise get an event b, which, having no possible causes, disappears once we put P in parallel with any other process, since this involves generating a b event for each set of possible causes.

Definition 7.2 (Removing labels and their dependants). *Given an event structure $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$ and a set of labels $A \subseteq \mathsf{Act}$, we define $\rho(A) = X$ as the maximum subset of E such that if $e \in X$ then $\lambda(e) \notin A$, and if $e \in X$ then there exists $x \in \mathsf{cause}(e)$ such that $x \subseteq X$.*

We give the REBES-semantics of Roll-CCSK in Table 4.

Much as we did in Proposition 4.7, we need to show that there exists a least upper bound of the event structures resulting from unfolding recursion. For this we first show that our action prefix, parallel composition, and tag binding are monotonic.

Proposition 7.3 (Unfolding). *Given a consistent process P and a level of unfolding l, if $\{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k\rangle$ and $\{\!|P|\!\}_{l-1} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, then $\mathcal{E}' \leq \mathcal{E}$, $\mathsf{Init} = \mathsf{Init}'$, and $k = k'$.*

Structurally congruent processes result in isomorphic event structures:

Proposition 7.4 (Structural Congruence). *Given consistent Roll-CCSK-processes P and P', if $P \equiv P'$, $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, and $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, then there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $f(\mathsf{Init}) = \mathsf{Init}'$ and for all $e \in \mathsf{Init}$, $k(e) = k'(f(e))$.*

Table 4: LREBES-semantics of Roll-CCSK

$\{\!|\mathsf{roll}\ \gamma|\!\}_l = \langle(\{e_r, e_t\}, \{e_r, e_t\}, \mapsto, \rhd, \lambda, \mathsf{Act}), \emptyset, \emptyset\rangle$ where:
$\{e_r\} \mapsto \underline{e_r}$, $\{e_t\} \mapsto \underline{e_t}$ $\quad \{e_t\} \mapsto e_r$, and $\{e_r\} \mapsto \underline{e_t}$ $\quad e_t \rhd \underline{e_r}$ and $e_r \rhd e_t$

$$\lambda(e) = \begin{cases} \mathsf{roll}\ \gamma & \text{if } e = e_r \\ \mathsf{start\ roll}\ \gamma & \text{if } e = e_t \end{cases} \qquad \mathsf{Act} = \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$$

$\{\!|\mathsf{rolling}\ \gamma|\!\}_l = \langle(\{e_r, e_t\}, \{e_r, e_t\}, \mapsto, \rhd, \lambda, \mathsf{Act}), \{e_t\}, \emptyset\rangle$ where:
$\{e_r\} \mapsto \underline{e_r}$, $\{e_t\} \mapsto \underline{e_t}$ $\quad \{e_t\} \mapsto e_r$, and $\{e_r\} \mapsto \underline{e_t}$ $\quad e_t \rhd \underline{e_r}$ and $e_r \rhd e_t$

$$\lambda(e) = \begin{cases} \mathsf{roll}\ \gamma & \text{if } e = e_r \\ \mathsf{start\ roll}\ \gamma & \text{if } e = e_t \end{cases} \qquad \mathsf{Act} = \{\mathsf{roll}\ \gamma, \mathsf{start\ roll}\ \gamma\}$$

$\{\!|\alpha_\gamma.P|\!\}_l = \langle(E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where:
$\{\!|P|\!\} = \langle(E_P, F_P, \mapsto_P, \rhd_P, \lambda_P, \mathsf{Act}_P), \mathsf{Init}, k\rangle$
$E = E_P \cup \{e_\alpha\}$ where e_α fresh

$$E_{Roll} = \left\{ e \,\middle|\, \begin{array}{l} \lambda_P(e) \in \{\mathsf{roll}\ \gamma', \mathsf{roll\ bound}\} \text{ or} \\ \lambda_P(e) \in \{\mathsf{start\ roll}\ \gamma' \mid \nexists \beta, n.\beta_{\gamma'} \text{ or } \beta_{\gamma'}[n] \text{ occurs in } \alpha_\gamma.P\} \end{array} \right\}$$

$F = F_P \cup \{e_\alpha\}$
$X \mapsto e$ if $X \mapsto_P e$ or $X = \{e_\alpha\}$, $e \in E_P$, and $\lambda_P(e) \neq \mathsf{roll}\ \gamma'$
$X \mapsto \underline{e}$ if $X = \{e\}$, or $e = e_\alpha$ and $X = \{e' \mid \lambda_P(e') = \mathsf{roll}\ \gamma\}$, or
$e \in E_{Roll}$ and $X \mapsto_P \underline{e}$, or $e \notin E_{Roll}$, $\{e\} \neq X' \mapsto_P \underline{e}$, and $X = X' \cup \{e' \mid \lambda_P(e') = \mathsf{roll}\ \gamma\}$
$\rhd = \rhd_P \cup (E_{Roll} \times \{\underline{e_\alpha}\}) \cup (\{e_\alpha\} \times \{\underline{e_r} \mid \lambda_P(e_r) = \mathsf{roll}\ \gamma\}) \cup$
$(\{e_r \mid \lambda_P(e_r) = \mathsf{roll}\ \gamma\} \times (E_{Roll} \cup \{e_\alpha\}))$
$\mathsf{Act} = \mathsf{Act}_P \cup \{\alpha\}$

$$\text{For all } e \in E,\ \lambda(e) = \begin{cases} \lambda_P(e) & \text{if } e \in E_P \\ \alpha & \text{if } e = e_\alpha \end{cases}$$

Table 4: LREBES-semantics of Roll-CCSK (continued)

$\{\!|\alpha_\gamma[m].P|\!\}_l = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where:

$\{\!|\alpha_\gamma.P|\!\} = \left\langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}', k' \right\rangle \quad \{e_\alpha\} = \{e \in E \mid \lambda(a) = \alpha \text{ and } \nexists X \subseteq E. X \mapsto e_\alpha\}$

$\mathsf{Init} = \mathsf{Init}' \cup \{e_\alpha\} \qquad k(e) = \begin{cases} k'(e) & \text{if } e \in \mathsf{Init}_P \\ m & \text{if } e = e_\alpha \end{cases}$

$\left\{\!\left| A\left\langle \tilde{b}, \tilde{\delta}\right\rangle \right|\!\right\}_0 = \langle (\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset), \emptyset, \emptyset\rangle$

$\left\{\!\left| A\left\langle \tilde{b}, \tilde{\delta}\right\rangle \right|\!\right\}_l = \left\{\!\left| (\nu\ \tilde{\delta}) P_A\{^{\tilde{b},\tilde{\delta}}/_{\tilde{a},\tilde{\gamma}}\} \right|\!\right\}_{l-1}$ where $A\langle \tilde{a}, \tilde{\gamma}\rangle = P_A$ and $l \geq 1$

$\{\!|P_0 \mid P_1|\!\}_l = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where:

$\{\!|P_i|\!\} = \mathcal{E}_i = (E_i, F_i, \mapsto_i, \rhd_i, \lambda_i, \mathsf{Act}_i)$ for $i \in \{0, 1\}$; $\mathcal{E}_0 || \mathcal{E}_1 = (E_\times, F_\times, \mapsto_\times, \rhd_\times, \lambda_\times, \mathsf{Act}_\times)$

$\mathsf{Init}_\times = \{(e_0, e_1) \mid e_0 \in \mathsf{Init}_0, e_1 \in \mathsf{Init}_1, k_0(e_0) = k_1(e_1)\} \cup$

$\{(*, e_1) \mid e_1 \in \mathsf{Init}_1, \nexists e_0 \in \mathsf{Init}_0. \lambda_0(e_0) = \overline{\lambda_1(e_1)}, \text{ and } k_0(e_0) = k_1(e_1)\} \cup$

$\{(e_0, *) \mid e_0 \in \mathsf{Init}_0, \nexists e_1 \in \mathsf{Init}_1. \lambda_0(e_0) = \overline{\lambda_1(e_1)}, \text{ and } k_0(e_0) = k_1(e_1)\}$

$E_{action} = \left\{ (X, e) \middle| \begin{array}{l} e \in E_\times, \lambda_\times(e) \notin \{\mathsf{roll}\ \gamma, \mathsf{roll\ bound}\}, \\ X \in \mathsf{cause}(e), \text{ and } \forall e' \in X. \exists X' \in \mathsf{cause}(e'). X' \subseteq X \end{array} \right\}$

$E_{roll} = \{e \mid e \in E_\times \text{ and } \lambda_\times(e) \in \{\mathsf{roll}\ \gamma, \mathsf{roll\ bound}\}\}$

$E = E_{action} \cup E_{roll}; \quad F_{action} = \{(X, e) \in E \mid e \in F_\times\}; \quad F_{roll} = E_{roll} \cap F_\times; \quad F = F_{action} \cup F_{roll}$

We define π_0 and π_1 such that for $(X, (e_0, e_1)) \in E_a$, $\pi_i((X, (e_0, e_1))) = e_i$, and

for $(e_0, e_1) \in E_r$, $\pi_i(e_0, e_1) = e_i$

$\{(X, e') \mid X' \subseteq X\} \mapsto (X', (e_0, e_1))$ if $e' \in X'$

$X \mapsto (e_0, e_1)$ if there exists X' such that $X' \mapsto_\times e$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$

$X \mapsto \underline{e}$ if $X = \{e\}$ or $e = (X', e_\times)$ and $X = \bigcup \left\{ X'' \middle| \begin{array}{l} \exists i \in \{0,1\}, X_i \in E_i. X_i \mapsto \overline{\pi_i(e)} \\ \text{ or } \exists e_\times \in X'. X_i \mapsto \overline{\pi_i(e_\times)} \\ \text{, and } e' \in X'' \text{ iff } \pi_i(e') \in X_i \end{array} \right\}$ or

$e = (e_0, e_1)$ and there exists X' such that $X' \mapsto_\times \underline{e}$ and $X = \{e' \mid (\pi_0(e'), \pi_1(e')) \in X'\}$

$e \rhd e'^*$ if there exists $i \in \{0, 1\}$ such that $\pi_i(e) \rhd_i \pi_i(e')^*$, or

$\pi_i(e) = \pi_i(e') \neq \bot$, and $e \neq e'$, $e'^* = e'$, or $e \neq e'$, and $e \in X \mapsto \underline{e'}$, or $e'^* = e'$ and $e, e' \in E_r$

$\mathsf{Act} = \mathsf{Act}_0 \cup \mathsf{Act}_1 \cup \{\tau\}$

$\lambda(e) = \begin{cases} \tau & \text{if } e = (X, (e_0, e_1)) \\ \lambda_0(e_0) & \text{if } e = (X, (e_0, *)) \text{ or } e = (e_0, *) \\ \lambda_1(e_1) & \text{if } e = (X, (*, e_1)) \text{ or } e = (*, e_1) \end{cases}$

$\mathsf{Init} = \{(X, e) \mid X \cup \{e\} \subseteq \mathsf{Init}_\times\} \cup (E_{roll} \cap \mathsf{Init}_\times)$

$k(e) = \begin{cases} k_0(e_0) & \text{if } e = (X, (e_0, *)) \\ k_1(e_1) & \text{if } e = (X, (*, e_1)) \\ k_0(e_0) & \text{if } e = (X, (e_0, e_1)) \quad \textit{– note that } k_0(e_0) = k_1(e_1) \end{cases}$

$\{\!|(\nu\ \gamma)P|\!\}_l = \langle (E, F, \mapsto, \rhd, \lambda, \mathsf{Act}), \mathsf{Init}, k\rangle$ where:

$\{\!|P|\!\} = \langle (E, F, \mapsto, \rhd, \lambda_P, \mathsf{Act}_P), \mathsf{Init}, k\rangle \qquad \mathsf{Act} = \mathsf{Act}_P \cup \{\mathsf{roll\ bound}\} \setminus \{\mathsf{roll}\ \gamma\}$

For all $e \in E$, $\lambda(e) = \begin{cases} \lambda_P(e) & \text{if } \lambda_P(e) \neq \mathsf{roll}\ \gamma \\ \mathsf{roll\ bound} & \text{if } \lambda_P(e) = \mathsf{roll}\ \gamma \end{cases}$

$\{\!|P \setminus A|\!\}_l = \left\langle \mathcal{E} \upharpoonright \rho(A \cup \overline{A}), \mathsf{Init} \cap \rho(A \cup \overline{A}), k \upharpoonright \rho(A \cup \overline{A}) \right\rangle$ where $\{\!|P|\!\}_l = \langle \mathcal{E}, \mathsf{Init}, k\rangle$

We next show that process P has a transition $P \xrightarrow{\mu} P'$ if and only if P and P' correspond to isomorphic event structures, and there exists a μ-labelled transition from the initial state of P's event structure to the initial state of P''s event structure.

Theorem 7.5. *Let P be a consistent Roll-CCSK process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, Init is conflict-free, and $C_{er}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$. Then*

1. *if there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \xrightarrow{\mu_\gamma[m]} P'$ then there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$;*
2. *and if there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ then there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, a transition $P \xrightarrow{\mu_\gamma[m]} P'$, and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \mu$, $f \circ k' = k[e \mapsto m]$, and $f(X) = \mathsf{Init}'$.*

We then prove the same correspondence for start roll transitions.

Proposition 7.6. *Let P be a consistent Roll-CCSK process such that $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, Init is conflict-free, and $C_{er}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$. Then*

1. *if there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \overset{\text{start roll } \gamma}{\rightsquigarrow} P'$ then there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \text{start roll } \gamma$, $f \circ k' = k$, and $f(X) = \mathsf{Init}'$;*
2. *and if there exists a transition $\mathsf{Init} \xrightarrow{\{e\}} X$ then there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$, a transition $P \overset{\text{start roll } \gamma}{\rightsquigarrow} P'$, and an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e) = \text{start roll } \gamma$, $f \circ k' = k$, and $f(X) = \mathsf{Init}'$.*

We finally show that a process P can make a roll γ transition if and only if the REBES corresponding to P can perform a roll γ event, followed by reversing all the events corresponding to actions and start roll's with tags causally dependent on γ, and then finally reversing the roll γ event.

Theorem 7.7. *Let P be a consistent process with $\{\!|P|\!\} = \langle \mathcal{E}, \mathsf{Init}, k\rangle$, $\mathcal{E} = (E, F, \mapsto, \rhd, \lambda, \mathsf{Act})$, $C_{er}(\mathcal{E}) = (E, F, \mathsf{C}, \rightarrow)$, and Init conflict-free, and let $\rho \in \{\text{roll } \gamma, \text{bound roll}\}$ be a roll label. Then*

1. *if there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \overset{\rho}{\rightsquigarrow} P'$, then there exist events e_r and $e_0, e_1, \ldots e_n$ such that $\mathsf{Init} \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \ldots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{done}$ and there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e_r) = \rho$, $\{e_0, e_1, \ldots e_n\} = \{e \mid \exists \gamma'.\gamma \leq_P \gamma'$ and either $\lambda(e)_{\gamma'}[k(e)]$ occurs in P or $\lambda(e) = \text{start roll } \gamma'$ and $\text{rolling } \gamma'$ occurs in $P\}$, $f \circ k' = k \upharpoonright \{e \mid f(e) \in \mathsf{Init}'\}$, and $f(X_{done}) = \mathsf{Init}'$;*
2. *and if there exist events e_r and $e_0, e_1, \ldots e_n$ such that $\mathsf{Init} \xrightarrow{\{e_r\}} X_0 \xrightarrow{\{\underline{e_0}\}} X_1 \ldots \xrightarrow{\{\underline{e_n}\}} X_{n+1} \xrightarrow{\{\underline{e_r}\}} X_{done}$ then there exists a process P' with $\{\!|P'|\!\} = \left\langle \mathcal{E}', \mathsf{Init}', k' \right\rangle$ and a transition $P \overset{\rho}{\rightsquigarrow} P'$ and there exists an isomorphism $f : \mathcal{E} \to \mathcal{E}'$ such that $\lambda(e_r) = \rho$, $\{e_0, e_1, \ldots e_n\} = \{e \mid \exists \gamma'.\gamma \leq_P \gamma'$ and either $\lambda(e)_{\gamma'}[k(e)]$ occurs in P or $\lambda(e) = \text{start roll } \gamma'$ and $\text{rolling } \gamma'$ occurs in $P\}$, $f \circ k' = k \upharpoonright \{e \mid f(e) \in \mathsf{Init}'\}$, and $f(X_{done}) = \mathsf{Init}'$.*

8 Conclusion

We have defined a category of reversible bundle event structures, and used the causal subcategory to model uncontrolled CCSK. Unlike previous work giving a truly concurrent semantics of a reversible process calculus using rigid families [6] or configuration structures [1], we have used the way CCSK handles past actions to generate both the event structure and the initial state directly from the process, rather than needing to first undo past actions to get the original process and from there the rigid family or configuration structure, and then redo the actions to get the initial state.

We have proposed a variant of CCSK called Roll-CCSK, which uses the rollback described in [9] to control its reversibility. We have defined a category of reversible extended bundle event structures, which use asymmetric rather than symmetric conflict, and used this category to model Roll-CCSK. Unlike in the case of CCSK, when modelling rollbacks in Roll-CCSK we use *non-causal* reversible event structures.

We have proved operational correspondence between the operational and event structure semantics of both CCSK (Theorem 4.10) and Roll-CCSK (Theorems 7.5 and 7.7).

Future work: We would like to provide event structure semantics for other reversible calculi. These mostly handle past actions using separate memories, which may prove challenging, particularly if we wish to avoid basing the semantics on finding the fully reversed process.

We also intend to explore the relationship between equivalences of processes and equivalences of event structures.

Acknowledgements: We thank the referees of RC 2018 for their helpful comments. This work was partially supported by EPSRC DTP award; EPSRC projects EP/K034413/1, EP/K011715/1, EP/L00058X/1, EP/N027833/1 and EP/N028201/1; and EU COST Action IC1405.

References

1. Aubert, C., Cristescu, I.: Contextual equivalences in configuration structures and reversibility. JLAMP **86**(1), 77 – 106 (2017). https://doi.org/10.1016/j.jlamp.2016.08.004
2. Boudol, G., Castellani, I.: Permutation of transitions: An event structure semantics for CCS and SCCS. In: de Bakker, J.W., de Roever, W.P., Rozenberg, G. (eds.) Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency. pp. 411–427. No. 354 in LNCS, Springer, Berlin, Heidelberg (1989). https://doi.org/10.1007/BFb0013028
3. Castellan, S., Hayman, J., Lasson, M., Winskel, G.: Strategies as concurrent processes. Electr. Notes Theor. Comput. Sci. **308**, 87–107 (2014). https://doi.org/10.1016/j.entcs.2014.10.006
4. Crafa, S., Varacca, D., Yoshida, N.: Event Structure Semantics of Parallel Extrusion in the Pi-Calculus. In: Birkedal, L. (ed.) FOSSACS. pp. 225–239. No. 7213 in LNCS, Springer, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28729-9_15
5. Cristescu, I., Krivine, J., Varacca, D.: A compositional semantics for the reversible pi-calculus. In: IEEE Symposium on Logic in Computer Science. pp. 388–397. LICS '13, IEEE Computer Society, Washington, DC, USA (2013). https://doi.org/10.1109/LICS.2013.45

6. Cristescu, I., Krivine, J., Varacca, D.: Rigid families for the reversible π-calculus. In: RC 2016. LNCS, vol. 9720, pp. 3–19. Springer (2016). https://doi.org/10.1007/978-3-319-40578-0_1
7. Danos, V., Krivine, J.: Reversible Communicating Systems. In: Gardner, P., Yoshida, N. (eds.) CONCUR. pp. 292–307. No. 3170 in LNCS, Springer, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-28644-8_19
8. Fecher, H., Majster-Cederbaum, M., Wu, J.: Bundle event structures: A revised cpo approach. Information Processing Letters **83**(1), 7 – 12 (2002). https://doi.org/10.1016/S0020-0190(01)00310-6
9. Lanese, I., Mezzina, C.A., Schmitt, A., Stefani, J.B.: Controlling Reversibility in Higher-Order Pi. In: Katoen, J.P., König, B. (eds.) CONCUR. pp. 297–311. No. 6901 in LNCS, Springer, Berlin, Heidelberg (2011). https://doi.org/10.1007/978-3-642-23217-6_20
10. Lanese, I., Mezzina, C.A., Stefani, J.B.: Reversing Higher-Order Pi. In: Gastin, P., Laroussinie, F. (eds.) CONCUR. pp. 478–493. No. 6269 in LNCS, Springer, Berlin, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15375-4_33
11. Langerak, R.: Transformations and Semantics for LOTOS. Ph.D. thesis, Universiteit Twente (1992), https://books.google.com/books?id=qB4EAgAACAAJ
12. Medić, D., Mezzina, C.A.: Static VS Dynamic Reversibility in CCS. In: Devitt, S., Lanese, I. (eds.) RC 2016. LNCS, vol. 9720, pp. 36–51. Springer International Publishing (2016). https://doi.org/10.1007/978-3-319-40578-0_3
13. Mezzina, C.A., Koutavas, V.: A safety and liveness theory for total reversibility. In: TASE. pp. 1–8 (Sept 2017). https://doi.org/10.1109/TASE.2017.8285635
14. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains. In: Kahn, G. (ed.) Semantics of Concurrent Computation. pp. 266–284. No. 70 in LNCS, Springer, Berlin, Heidelberg (1979). https://doi.org/10.1007/BFb0022474
15. Phillips, I., Ulidowski, I.: Reversibility and models for concurrency. Electr. Notes Theor. Comput. Sci. **192**(1), 93–108 (2007). https://doi.org/10.1016/j.entcs.2007.08.018
16. Phillips, I., Ulidowski, I.: Reversibility and asymmetric conflict in event structures. JLAMP **84**(6), 781 – 805 (2015). https://doi.org/10.1016/j.jlamp.2015.07.004, Special Issue on Open Problems in Concurrency Theory
17. Phillips, I., Ulidowski, I., Yuen, S.: A Reversible Process Calculus and the Modelling of the ERK Signalling Pathway. In: Glück, R., Yokoyama, T. (eds.) RC. pp. 218–232. No. 7581 in LNCS, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36315-3_18
18. Phillips, I., Ulidowski, I., Yuen, S.: Modelling of Bonding with Processes and Events. In: Dueck, G.W., Miller, D.M. (eds.) RC. pp. 141–154. No. 7948 in LNCS, Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38986-3_12
19. Phillips, I., Ulidowski, I.: Reversing algebraic process calculi. Journal of Algebraic and Logic Programming **73**(1-2), 70–96 (2007). https://doi.org/10.1016/j.jlap.2006.11.002
20. Vaandrager, F.W.: A simple definition for parallel composition of prime event structures. CS R 8903, Centre for Mathematics and Computer Science, P. O. box 4079, 1009 AB Amsterdam, The Netherlands (1989)
21. Winskel, G.: Event structure semantics for CCS and related languages. In: Nielsen, M., Schmidt, E.M. (eds.) ICALP. pp. 561–576. No. 140 in LNCS, Springer, Berlin, Heidelberg (1982). https://doi.org/10.1007/BFb0012800