

On the origin of events: branching cells as stubborn sets^{*}

Henri Hansen¹ and Xu Wang²

¹ Department of Software Systems, Tampere University of Technology
PO Box 553, FI-33101 Tampere, FINLAND

`henri.hansen@tut.fi`

² International Institute of Software Technology, United Nations University
Macau, PO Box 3058

`wx@iist.unu.edu`

Abstract. In prime event structures with binary conflicts (pes-bc)³ a *branching cell* [1] is a subset of events closed under *downward causality* and *immediate conflict* relations. This means that no event outside the branching cell can be *in conflict with* or *enable* any event inside the branching cell. It bears a strong resemblance to *stubborn sets*, a partial order reduction method on transition systems. A stubborn set (at a given state) is a subset of actions such that no execution consisting entirely of actions outside the stubborn set can be in conflict with or enable actions that are inside the stubborn set.

A rigorous study of the relationship between the two ideas, however, is not straightforward due to the facts that 1) stubborn sets utilise sophisticated causality and conflict relations that invalidate the *stability* and *coherence* of event structures [18], 2) without stability it becomes very difficult to define concepts like prefixes and branching cells, which pre-require a clear notion of causality, and 3) it is challenging to devise a technique for identifying ‘proper’ subsets of transitions as ‘events’ such that the induced event-based system captures exactly the causality and conflict information needed by stubborn sets.

In this paper we give a solution to the problems by providing an unfolding of labelled transition systems into *configuration structures*, a more general structure supporting or-causality and finite conflict. We show that the branching cell definition can be extended to configuration structures and that each branching cell in the unfolding is a *long-lived* stubborn set, such that no matter how the system evolves, what remains of the branching cell is always a stubborn set.

1 Introduction

Partial Order Reduction (POR) has been successful in alleviating the state explosion problem that manifests in the verification of concurrent systems. Although

^{*} Supported by the PEARL project (041/2007/A3) from FDCT (Macau).

³ Pes-bc was originally called event structures in [8]. Later Winksel introduced, in [18], a more general structure, which was also named event structure. In this paper, we use the name event structure for the latter.

drawing ideas and intuition from non-interleaving models of concurrency (esp. Mazurkiewicz traces [7]), most existing POR methods, e.g. the ample, persistent and stubborn set methods [9, 3, 11] and confluence based methods [4, 6], are built on top of interleaving models, mostly transition systems of various types.

On the face of it, it seems straightforward that notions like stubborn sets can be projected onto non-interleaving models and we can readily obtain their true-concurrency counterparts. In particular, branching cells [1, 17] of pes-bc bear resemblance to stubborn sets. A closer look of the problem, however, shows that the projection is nontrivial and should better be carried out in two steps.

The first step is fairly easy: in a non-interleaving model it is intuitively simple to define a construct that captures the essence of stubborn sets, which, surprisingly though, is not branching cells. The second step is where the difficulty lies. The correspondence between stubborn sets and the construct cannot be established if we limit our notions of conflict and causality to those of pes-bc, when projecting transition systems into event-based models. It seems stubborn sets do implicitly utilise the extra concurrency affordable by or-causality and finite conflicts (details in Section 3.2). Thus our event-based model have to be more general than pes-bc.

In this paper our projection from transition systems to event-based systems is based on similar frameworks as those of [19, 10, 14]. We use local ‘diamonds’ in LTSes and identify an event with the subset of parallel edges (i.e. transitions) in a ‘diamond-ful’ subgraph. In [19, 10], the proposed formalism is *transition systems with independence*. However, given a transition system, the consistency conditions (on the independence relations) rule out a significant amount of diamonds to be interpreted as independence. The derived event-based model will have a simplified structure regarding causality and conflicts, and consequently fails to capture as independence some of the diamonds utilised by stubborn sets. In [14], a formalism called *process graphs* is proposed. They can be projected into very general event-based models with more complicated structure regarding causality and conflicts. Indeed, process graphs try to exploit as many diamonds as possible, to the extent that even *resolvable conflicts* [14] can be included in a single (non-interleaving) run of the system. This contradicts with the intuition of stubborn sets, which treats resolvable conflicts as normal conflicts.

In this paper we propose a new technique, a dynamic procedure to identify events from LTSes and build a corresponding event-based system, utilising the same class of diamonds as POR. We use configuration structures [15] as our model of event-based systems. It is a general structure supporting finer grained causality and conflicts needed by stubborn sets. Moreover, compared to [19, 10, 14], our procedure supports dynamic assignment of independence relations: a pair of transitions can be dependent in one execution while be independent in another.

Our paper is organized as follows. In Section 2 and Section 3, we outline the basic definitions needed, on one hand, for LTSes and stubborn sets and, on the other hand, for configuration structures and branching cells. Then we introduce the notion of *extended knots*, which is defined w.r.t. a given state of the system,

and show that a branching cell corresponds to an extended knot that spans over a given subgraph of a system. Section 4 gives the construction and correctness proof of *unfolding* LTSes into configuration structures and mapping (subsets of) transitions into events. In Section 5 we prove the correspondence result between extended knots and stubborn sets. Section 6 concludes.

2 Definitions and Fundamentals

2.1 Labelled transition systems

Definition 1. A deterministic labelled transition system (DLTS) is a 4-tuple $(S, \Sigma, \Delta, \hat{s})$ where

- S is a set of states,
- Σ is a finite set of actions (ranged over by a, b , etc.)⁴,
- Δ is a partial function from $S \times \Sigma$ to S (i.e. the transition function), and
- $\hat{s} \in S$ is the initial state.

For a given DLTS $L = (S, \Sigma, \Delta, \hat{s})$, we define the following:

- $s \xrightarrow{a} s'$ means $(s, a, s') \in \Delta$,
- $s \xrightarrow{a_1 \cdots a_n} s'$ means there exists s_0, \dots, s_n such that $s_0 = s$, $s_n = s'$ and for each $1 \leq i \leq n$, $s_{i-1} \xrightarrow{a_i} s_i$.
- $s \xrightarrow{a}$ means $\exists s' : s \xrightarrow{a} s'$, and $s \not\xrightarrow{a}$ means that $s \xrightarrow{a}$ does not hold;
- $eb(s) = \{a | s \xrightarrow{a}\}$ denotes the set of actions enabled at s .

When there is any danger of confusion, we use \rightarrow_Δ to say it is derived from a DLTS with Δ as the transition function.

The following definition of stubborn sets can be found in [13], albeit notation may differ slightly.

Definition 2. Let $L = (S, \Sigma, \Delta, \hat{s})$ be a DLTS and let $s_0 \in S$ be a state. A set $T \subseteq \Sigma$ is a stubborn set at s_0 if for every $a \in T$ and $b_1, \dots, b_k \notin T$ the following hold:

1. If $s_0 \xrightarrow{a} s'_0$ and $s_0 \xrightarrow{b_1 \cdots b_k} s_k$, then $s'_0 \xrightarrow{b_1 \cdots b_k} s'_k$ and $s_k \xrightarrow{a} s'_k$
2. if $s_0 \xrightarrow{b_1 \cdots b_k} s_k \xrightarrow{a}$, then $s_0 \xrightarrow{a}$
3. $T \cap eb(s_0) = \{\}$ if and only if $eb(s_0) = \{\}$

Furthermore, if additionally $T \subseteq eb(s_0)$ holds, we say T is a persistent set at s_0 .

Unlike, e.g., for persistent/ample sets [9, 2], the definition of a stubborn set does not exclude actions that are not enabled (at s_0). For application in state space reduction this makes no real difference. If we are interested only in the enabled subset of actions of a stubborn set T , we denote it by $T \cap eb(s_0)$, which is exactly a persistent set.

⁴ We further assume that, for each $a \in \Sigma$, there is a reachable state s such that $s \xrightarrow{a}$.

It is perhaps easier to understand the stubborn set definition in terms of five types of *state-local* binary relations between actions for a given LTS, $(S, \Sigma, \Delta, \hat{s})$. Any local relation between say action a and b w.r.t. a state s_0 can be calculated by checking at most four transitions ‘local’ to s_0 : the a and b transitions at s_0 , the b transition after a , and the a transition after b (c.f. Figure 1).

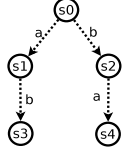


Fig. 1. Local transitions at s_0

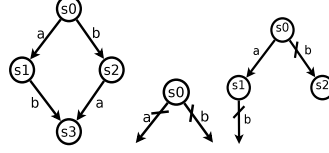


Fig. 2. Conditional independence ($a ||_{s_0} b$)

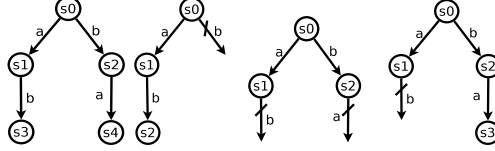


Fig. 3. Shapes for (the four types of) non-independence

With the four potential transitions, we can construct a number of possible ‘shapes’ for local transition systems at s_0 . The set of shapes can be partitioned into five groups; each one characterised by a type of binary relations between a and b :

Definition 3. Let $a, b \in \Sigma$ be actions of a DLTS and $s_0 \in S$ be a state.

- a and b are conditionally independent at s_0 (i.e. $a ||_{s_0} b$) if one of the following hold (c.f. Figure 2):
 1. $s_0 \xrightarrow{ab} s_3$, $s_0 \xrightarrow{ba} s_4$, and $s_3 = s_4$,
 2. $s_0 \not\xrightarrow{a}$ and $s_0 \not\xrightarrow{b}$, or
 3. $s_0 \xrightarrow{a} s_1$, $s_0 \xrightarrow{b}$ and $s_1 \xrightarrow{b}$ (or symmetrically by exchanging a and b).
- a and b are non-commutative if $s_0 \xrightarrow{ab} s_3$, $s_0 \xrightarrow{ba} s_4$, and $s_3 \neq s_4$ (c.f. the first case in Figure 3).
- a enables b if $s_0 \not\xrightarrow{b}$ and $s_0 \xrightarrow{ab}$ (c.f. the second case in Figure 3).
- a and b symmetrically disable each other if $s_0 \xrightarrow{a}$ and $s_0 \xrightarrow{b}$ but $s_0 \not\xrightarrow{ab}$ and $s_0 \not\xrightarrow{ba}$ (c.f. the third case in Figure 3).
- a asymmetrically disables b if $s_0 \xrightarrow{a}$ and $s_0 \xrightarrow{ba}$, but $s_0 \not\xrightarrow{ab}$ (c.f. the fourth case in Figure 3).

Note that conditional independence, non-commutativity, and symmetrical disabling are irreflexive symmetric relations while enabling and asymmetrical disabling are irreflexive antisymmetric relations. Furthermore we say a and b are *in conflict* at s_0 if either a and b are non-commutative, or a and b symmetrically or asymmetrically disable each other. Conflict is an irreflexive symmetric relation.

Using the five binary relations, a non-local property (w.r.t. a state) can sometimes be decomposed into a series of local relations over a subgraph of states⁵. For example, stubborn sets are characterized by these local relations according to the following folk theorem.

Proposition 1. *The non-empty set $T \subseteq \Sigma$ is a stubborn set at $s_0 \in S$ iff, for every $a \in T$ and $b \notin T$, **either a and b are conditionally independent at s_k or a enables b at s_k** (i.e. neither a and b are in conflict at s_k nor b enables a at s_k) for all states s_k reachable from s_0 by using non- T transitions, i.e. $s_0 \xrightarrow{b_1 \cdots b_k} s_k$ with $b_1, \dots, b_k \in \Sigma \setminus T$.*

Proof. From SS (stubborn set) to local definition: Assume that neither a and b are conditionally independent nor a enables b at s_k , and that s_k is the first such state on the path $s_0 \xrightarrow{b_1 \cdots b_n} s_k$. Then there are three possibilities: 1) a is enabled at s_0 and a and b symmetrically or asymmetrically disable each other at s_k , 2) a is not enabled at s_0 and b enables a at s_k , and 3) a is enabled at s_0 and a and b are non-commutative at s_k . The second case contradicts the second requirement of stubborn sets; while the other two cases contradict the first requirement.

From local definition to SS: Let us assume $s_0 \xrightarrow{a} s'$ and $s_0 \xrightarrow{b_1 \cdots b_k} s_k$. It is clear that a and b_i are conditionally independent at each state s_{i-1} (such that $s_0 \xrightarrow{b_1 \cdots b_{i-1}} s_{i-1}$) for every $1 \leq i \leq k$; it cannot be true that a enables b_i at s_{i-1} since b_i is already enabled at s_{i-1} . Therefore we know that $s_i \xrightarrow{a}$. From conditional independence it also follows that $s_{i-1} \xrightarrow{a} s'_{i-1} \xrightarrow{b_i} s'_i$ and $s_i \xrightarrow{a} s'_i$, for each $i \leq k$, and the first requirement of stubborn sets is satisfied.

Assume then, that $s_0 \xrightarrow{b_1 \cdots b_k} s_k \xrightarrow{a}$. Now, the independence/enabling requirement states that none of the b_i for $1 \leq i \leq k$ enables a , so the second requirement readily follows, and T must be stubborn.

3 Configuration structures and Branching cells

In this section we first introduce configuration structures [16, 15]. Then, as our original contribution, we develop the new notion of *knots* and *extended knots*, which are the non-interleaving counterparts of persistent sets and stubborn sets resp. (see proofs in Section 5). Finally, we extend to configuration structures the definition of *prefixes*, *immediate conflicts* and *branching cells*, and show how branching cells are actually long-lived extended knots.

⁵ Decomposability, in practice, implies the existence of efficient algorithms to analyse such non-local properties.

3.1 Well-formed configuration structures

Definition 4. A configuration structure (or simply CS) is a pair (E, C) , where

- E is a potentially infinite set of events, and
- C is a set of configurations over E , where a configuration $c \in C$ is a finite subset of E .

A configuration c can be thought of as representing a state reached after an execution of exactly the set c of events. The empty configuration $\{\}$ represents the initial state and is a required member of C in our model.

Below we fix a configuration structure $cs = (E, C)$ and introduce some basic notions (based on those in [16, 15]) for CSes.

- We say there is a *step-transition* from c to c' , written $c \xrightarrow{c' \setminus c}_C$, if $c \subseteq x \subseteq c' \implies x \in C$ holds for all $x \subseteq E$. (Note that in this paper we do not list explicitly the destination configuration of the transition, i.e. c' .)
- Sometimes we use $c \xrightarrow{e}_C$ to mean $c \xrightarrow{\{e\}}_C$, i.e. e is *enabled* at c . We write $eb(c) = \{e \in E \mid c \xrightarrow{e}_C\}$, to denote the set of enabled events at c .
- We say cs is *finitely branching* if $eb(c)$ is finite for all $c \in C$. In such a cs *infinite configurations* can be derived from finite ones and there is no unbounded concurrency and conflict.
- For a nonsingleton set x , $c \xrightarrow{x}_C$ represents the concurrent execution of multiple events. Obviously, a nonsingleton step-transition is reducible to a sequence of singleton step-transitions.
- We say a nonempty finite set $K \subseteq E$ is a *consistent set* if there is $c \in C$ such that $K \subseteq c$. Otherwise, K is an *inconsistent set*.
- We say that cs is *closed under (nonempty) bounded union* if, for all nonempty finite subsets $D \subseteq C$, $\bigcup D$ is a consistent set implies $\bigcup D \in C$.
- We say that cs is *connected* if all the configurations in C are reachable from $\{\}$ by step-transitions, and that cs is *irredundant* if every event from E is contained inside at least one configuration from C .

Based on the above, we can say cs is *well-formed* if it is irredundant, finitely branching, connected and closed under bounded union. Well-formed CSes have roughly the same expressiveness as event structure from [18]. We prefer to use configuration structures in this paper mainly because of its affinity to LTSes. We establish a few basic properties of well-formed CSes:

Lemma 1. *If there is a step-transition path from c to c' such that $e \in eb(c)$ and $c' \cup \{e\}$ is consistent, then we have either $e \in c'$ or $e \in eb(c')$.*

Proof. If c_i and c_{i+1} are two adjacent configurations (connected by event $e' \neq e$) on the path and e is enabled at c_i , closure under bounded union gives us that e is enabled at c_{i+1} .

Lemma 2. *Let $c, c' \in C$ such that $c \subseteq c'$. Then there is a (step-transition) path from c to c' .*

Proof. Follows from Lemma 5.2.5 in [16].

For the rest of this paper we only consider well-formed CSes and simply call them CSes. Furthermore, to gain a deeper understanding of CSes, we need to use a few more notions:

- We say e is *initially enabled* at $c \in C$ (or c is the *cause* of e) if e is enabled at c but not enabled at any $c' \subset c$.
- We use $IE(e)$ to denote the set of configurations at which e is initially enabled. In other words, $IE(e)$ is the set of minimal configurations on which e is enabled.
- We say c has previously *encountered* an event e in its evolution if $e \in A(c) = \{e \mid \exists c_0 \in IE(e) : c_0 \subseteq c\}$.
- Given a configuration $c \in C$, the *future* of cs after c , is defined to be the configuration structure: $cs/c = (E_{cs/c}, C_{cs/c})$, where $C_{cs/c} = \{c' \setminus c \mid c' \in C \wedge c \subseteq c'\}$ and $E_{cs/c} = \bigcup C_{cs/c}$. Obviously cs/c is well-formed if cs is well-formed.
- We say a step-transition $c \xrightarrow{e}_C$ *disables* an event e' if e' is enabled at c but not at $c \cup \{e\}$.
- We say a step-transition $c \xrightarrow{e}_C$ *eliminates* an event e' from the system if $e' \in E_{cs/c}$ but $e' \notin E_{cs/(c \cup \{e\})}$. According to Lemma 1 and 2, we know that once e' is disabled it is eliminated from the system.

The causes of an event in CSes are not necessarily unique. The loss of uniqueness is either due to *or-causality* or due to *the loss of conflict heredity*. In Figure 4, the 1st and 3rd graphs describe the or-causality case [5, 20], i.e. $e3$ or-causally depends on $e1$ and $e2$; while the 2nd and 4th graphs describe the loss-of-conflict-heredity case, i.e. $e3$ causally depends on a pair of conflicting events, i.e. $e1$ and $e2$.⁶ The 3rd and 4th graphs are the CSes of the examples, from which we can see that $\{e1\}$ and $\{e2\}$ are both causes of $e3$. The 1st and 2nd graphs are the ‘*extended event structures*’ of the examples⁷: head-conjoined arrows (marked by *or*) represent or-causal dependency while dotted polygons represent conflict relations (i.e. the set of events inside a polygon form a conflict). Furthermore, with ‘extended event structure’ notation derivable conflicts or causal dependency do not need to be drawn explicitly.

3.2 Knots and extended knots

A concurrent system consists of a set, say H , of behaviours. In a non-interleaving setting, such a behaviour, say $h \in H$, is a non-interleaving maximal run of

⁶ Note that the second case will be impossible if we have conflict heredity, from which we can derive $e3$ is in conflict with $e3$, contradicting the irreflexivity of conflict relations.

⁷ It might look unlikely that the pathological case of the 2nd graph can produce a well-formed CS. But by checking the 4th graph readers can verify that it is so indeed.

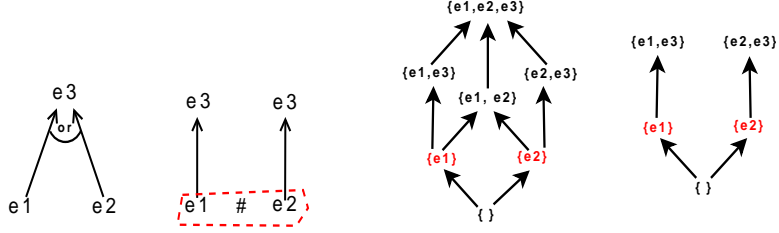


Fig. 4. Two types of multi-causes

the system, which usually is characterised by a maximal configuration. A graph traversal algorithm is supposed to exhaustively explore the set of H . The idea of POR lies in that, given any $h \in H$, it is necessary and only necessary to explore one linearisation of h . To this end the following notion of *weak knots* is useful.

- A non-empty subset $K_0 \subseteq eb(\{\})$ is a *weak knot* if, for all $c \in C$, there exists some $e \in K_0$ such that $c \cup \{e\}$ is consistent.

The definition says that, no matter how the system evolves, the resultant configuration either contains a member of K_0 or is extendable to one containing a member of K_0 . Thus, ignoring events outside K_0 at the current configuration will not cause the exploration missing any behaviour of the system.

Weak knots seem to be the weakest possible notion preserving behaviours. In this sense the notion has some canonicity and is of independent interests. However, weak knots are too weak to capture persistent sets and stubborn sets.

Persistent sets and stubborn sets impose stronger restrictions. For weak knots, we require that, no matter how the system evolves, at least one member of K_0 has survived (i.e. executed) or will survive (i.e. not eliminated yet), since the evolution of the system may eliminate members of K_0 . For persistent sets and stubborn sets, we require that before a members of K_0 has survived no members of K_0 can be eliminated.

- A non-empty subset $K_0 \subseteq eb(\{\})$ is a *knot* if, for all $c \in C$, either $c \cap K_0 \neq \{\}$ or $c \cup \{e\}$ is consistent for all $e \in K_0$.

Thus, knots should be the counterparts to persistent set. The counterpart of stubborn sets is more complicated, which needs to further accommodate the requirement that any member of K_1 that is not enabled yet cannot become enabled without first executing a member of K_1 :

- Given any $K \subseteq E$, we say K can block an event $e \in E$ if $e \notin K$ and $K \cap c \neq \{\}$ for all $c \in IE(e)$. And we say e can block e' if $\{e\}$ can block e' .
- A non-empty subset $K_1 \subseteq E$ is an *extended knot* if, $K_0 = K_1 \cap eb(\{\})$ is a knot and K_0 can block all $e \in K_1 \setminus eb(\{\})$.

To formally establish the correspondence between knots and persistent/stubborn sets, we need to project LTSes into event-based systems that support or-causality and finite conflicts. The reasons are based on the following observations:

- For a confluent LTS, such as the one derived from the 3rd graph in Figure 4, singleton sets suffice as stubborn sets at each state, which implies the corresponding event-based system consists of a single behaviour, i.e. a conflict-free system. However, without or-causality, there is no way to project the LTS into a conflict-free event-based system.
- With finite conflicts the LTS like the one in Figure 9 can be projected into an event-based system that consists of three non-interleaving runs. This matches with the three sequential runs generated by minimal stubborn sets. However, without finite conflicts the projected event-based system will contain at least four non-interleaving runs.

We will formally define the projection to CSes in Section 5 . But before that, let us first extend the definition of branching cells onto CSes.

3.3 Prefixes, immediate conflicts and branching cells

The definition of configuration structures is very ‘low-level’. We have to build up other notions of true concurrency (e.g. conflict and causality) step by step from configurations. We will see that some of the notions are surprisingly subtle to define and there has been few previous works we can rely on.

- We say an inconsistent subset $K \subseteq E$ is a *conflict set* (at $\{\}$) if all its strict subsets are consistent. We say cs is *conflict-free* if cs has no conflict set. Obviously cs is conflict-free iff $E \in C$.

The arity or degree of conflicts is not fixed in configuration structures: a conflict may involve any number of events. Thus we use *conflict sets* (rather than conflict tuples) to define conflicts. A conflict set is a dynamic notion relative to (i.e. indexed by) a configuration: as cs evolves, conflicts in cs evolve too. For instance, an initially consistent subset $K \subseteq E$ may become a conflict later on (say in cs/c). On the other hand, conflict-freedom is preserved by the evolutions of cs . Let us take a closer look at conflict dynamism.

We say cs/c *inherits* conflict K of cs if $K \setminus c$ is a conflict of cs/c . Actually, all the conflict sets in cs/c are inherited from cs :

Lemma 3. *Given $e \in eb(\{\})$, K is a conflict of $cs/\{e\}$ implies K or $K \cup \{e\}$ is a conflict of cs .*

Proof. If K is inconsistent in cs , then obviously K is a conflict set of cs . If K is consistent in cs , then $K \cup \{e\}$ is a conflict set of cs since $K \cup \{e\}$ is inconsistent and $(K \setminus \{e'\}) \cup \{e\}$ is consistent in cs for all $e' \in K$.

It is also possible that, with cs evolution, some conflicts of cs will be *discarded* by cs/c . An example is that, given a conflict K of cs , suppose an enabled event $e \notin K$ is in a ternary conflict with $e', e'' \in K$ ⁸. By executing e , the ternary conflict is reduced to conflict $\{e', e''\}$ in $cs/\{e\}$, making K obsolete in $cs/\{e\}$.

⁸ That is, $(E = \{e, e', e'', e3\}, C = \{c \subseteq E | \{e, e', e''\} \not\subseteq c \wedge K \not\subseteq c\})$, where $K = \{e', e'', e3\}$.

Proposition 2. Given a conflict K of cs and $e \in eb(\{\})$,

1. $K = \{e, e'\}$ implies e' is eliminated in $cs/\{e\}$, i.e. $e' \notin E_{cs/\{e\}}$,
2. $e \in K$ and K is not binary implies that $K \setminus \{e\}$ is a conflict of $cs/\{e\}$ (i.e. indirect inheritance),
3. $e \notin K$ but there exists another conflict K' with $e \in K'$ and $K' \setminus \{e\} \subseteq K$ implies K is not a conflict in $cs/\{e\}$ (i.e. discard), and
4. $e \notin K$ and there does not exist any conflict K' with $e \in K'$ and $K' \setminus \{e\} \subseteq K$ implies K is a conflict in $cs/\{e\}$ (i.e. direct inheritance).

Proof. Case 1 is obvious. For case 2, it comes from the facts that $K \setminus \{e\}$ is inconsistent in $cs/\{e\}$ but $K \setminus \{e'\}$ is consistent in $cs/\{e\}$ for all $e' \in K \setminus \{e\}$. Case 3 is also straightforward after separating binary K' from non-binary K' . For case 4, we can first prove that $K \subseteq E_{cs/\{e\}}$ since otherwise a binary K' satisfying the conditions must exist in order to eliminate a member of K . Then, since K is inconsistent in $cs/\{e\}$ and no subset of K can be a conflict of $cs/\{e\}$ (due to Lemma 3), it concludes that K is a conflict of $cs/\{e\}$.

The notion of causality is even more subtle. An extreme example of such subtlety can be found in Figure 5. At $\{e1\}$, the execution of $e2$ enables $e4$; while at $\{e3\}$, the execution of $e4$ enables $e2$. Thus $e2$ and $e4$ (or-)causally depend on each other. The ‘extended event structure’ representation of the same example can be found in Figure 6 (the left graph).

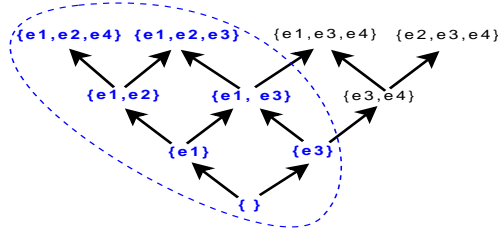


Fig. 5. Cyclic (or-)causality

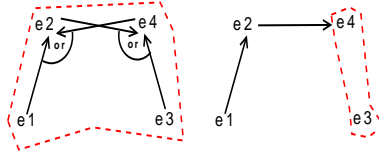


Fig. 6. Two example prefixes

In this paper we will not try to give an explicit definition to causality. Rather, we define notions that involve causality only implicitly.

Given a finite $D \subseteq C$, we say D is *downward-closed* (w.r.t. \subseteq) if $c \subseteq c' \in D \implies c \in D$, and D is *bounded-union closed* if $c, c' \subseteq D \wedge c \cup c' \in C \implies c \cup c' \in D$. We use D^\downarrow to mean the downward closure of D . Then,

- We say a finite subset of configurations $D \subseteq C$ is a *prefix* if there exists $D_0 \subseteq C$ so that D_0 is downward-closed and D is the bounded-union closure of D_0 . Given a prefix D' , another prefix D is a *sub-prefix* of D' if $D \subseteq D'$. Prefixes induce well-formed CSes, i.e. $(\bigcup D, D)$.

Note that prefixes are subsets of configurations rather than subsets of events since the latter is not expressive enough⁹: In Figure 5 both the whole graph and the subgraph carved out by the dotted ellipse are prefixes (corresponding to the left and right ‘extended event structures’ in Figure 6 resp.) of the CS; but their sets of involved events are the same, i.e. $\{e_1, e_2, e_3, e_4\}$.

Note also that prefixes are not necessarily downward-closed. In Figure 7, the bounded-union closure of $\{\{e, d, c\}, \{e, b\}, \{a\}\}^\downarrow$ is not downward closed.

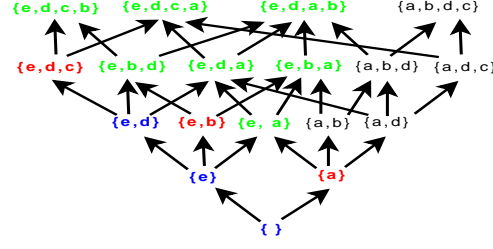


Fig. 7. A more sophisticated prefix

- A prefix D' is *convex* if, for all other prefix D with $\bigcup D = \bigcup D'$, we have $D \subseteq D'$.

Given $B \subseteq E$, the *projection* of cs onto B , $cs \upharpoonright_B = (E_{cs \upharpoonright_B}, C_{cs \upharpoonright_B})$, is defined to be $C_{cs \upharpoonright_B} = \{c \mid c \in C \wedge c \subseteq B\}$ and $E_{cs \upharpoonright_B} = \bigcup C_{cs \upharpoonright_B}$. For a convex prefix D it is completely characterised by $\bigcup D$ since $D = cs \upharpoonright_{\bigcup D}$.

We will use D and $\bigcup D$ interchangeably for convex prefixes, and we have, for any two convex prefixes B and B' , B is a sub-prefix of B' iff $B \subseteq B'$.

An interesting subclass of convex prefixes are the so-called *strong prefixes*.

- A prefix D is a *s-prefix* if, for all $e \in \bigcup D$ and $c \in IE(e)$, we have $c \in D$.

The alternative definition of s-prefix is: $B \subseteq E$ is a *s-prefix* if, for all $e \in B$ and $c \in IE(e)$, $c \subseteq B$. Note the distinctions between prefixes and s-prefixes: for each $e \in \bigcup D$ a prefix requires inclusion of one cause of e while an s-prefix requires inclusion of all causes of e .

Now we are ready to define immediate conflicts and branching cells.

- We say a subset of events $K \subseteq E$ is a *local conflict set* (LCS) at c if $K \subseteq eb(c)$ and K is a conflict of cs/c . Given two LCSes K at c and K' at c' , we say K' at c' is *derivable from* K at c if $K' \subseteq K$ and $c \subset c'$ (c.f. direct and indirect inheritance of conflicts).

⁹ Obviously with event-based prefixes it becomes impossible (due to or-causality) to extend a prefix on an event by event basis, if prefixes are still interpreted as ‘downward-closure of causality’.

- We say a subset of events $K \subseteq E$ is an *immediate conflict set* (ICS) at c if K at c is an LCS that is not derivable from any other LCS.
- We say a prefix D *characterises* an ICS (i.e. $\bigcup D \setminus \bigcap \max(D)$ at $\bigcap \max(D)$), where $\max(D)$ denotes the set of maximal elements of D) if D is a minimal prefix that is not conflict-free¹⁰.

The first definition of ICS captures the localness of immediate conflicts. For instance, in Figure 7, $\{a, b, c\}$ at $\{e, d\}$ is an ICS but $\{a, b\}$ at $\{e, d, c\}$ is not an LCS. The second definition, i.e. characterisation, lifts the original definition on pes-bc. We can see that the two definitions coincide:

Lemma 4. *A prefix D characterises an immediate conflict implies that $\bigcup D \setminus \bigcap \max(D)$ is an ICS at $\bigcap \max(D)$; and K is an ICS at c implies there is a prefix characterising it.*

Proof. From characterisation to ICS: For any $c \in \max(D)$, find some $e \in \bigcup D \setminus c$ and $c_0 \in IE(e)$ such that $c_0 \subseteq c$. Then the bounded-union closure of $\{c, c_0 \cup \{e\}\}^\downarrow$ gives rise to D since there is no prefix that is (strictly) larger than $\{c\}^\downarrow$ but smaller than D . Thus $c \cup \{e\} = \bigcup D$. Furthermore, for any $c' \in D$ with $c' \setminus c = \{e\}$, there exists some $c_1 \in \{c\}^\downarrow$ such that $c_1 \cup (c_0 \cup \{e\}) = c'$. Thus we have $c_0 \cup c_1 = c \cap c' \in C$. Using this rule we can intersect members of $\max(D)$ up one by one. Eventually we get $\bigcap \max(D) \in C$ and $\bigcup D \setminus \bigcap \max(D)$ is an LCS at $\bigcap \max(D)$. Also we can easily see that if the LCS is derivable from another LCS. Then D cannot be minimal.

From ICS to characterisation: Let D be a minimal prefix that contains $M = \{c \cup (K \setminus \{e\}) \mid e \in K\}$. K at c being ICS implies $c \in IE(K)$. The minimality of D implies that, for all $e \in K \cup c$ and $c_0 \in D$, $c_0 \in IE(e)$ implies $c_0 \subseteq c$. Thus there is no $c' \in D$ such that $K \subseteq c'$ and we have $\max(D) = M$. Since adding $K \cap c$ to D will give rise to a confluent CS, say cs' , and any strict sub-prefix of D is also a strict sub-prefixes of $D \cup \{K \cap c\}$, all strict sub-prefix of D are conflict-free.

It seems that immediate conflict and initial enabledness are two fundamental notions in CSes. They together completely characterise CSes.

Lemma 5. *Two CSes $cs = (E, C)$ and $cs' = (E', C')$ are isomorphic iff there is a bijection ϕ from E to E' such that 1) $\phi(IE_{cs}(e)) = IE_{cs'}(\phi(e))$ for all $e \in E$ and 2) K at c is an ICS of cs iff $\phi(K)$ at $\phi(c)$ is an ICS of cs' .*

Proof. Use the fact that $c \in C$ implies there is $(e_1, c_1), (e_2, c_2), \dots, (e_n, c_n)$ such that $c_i \in IE(e_i)$ for all $1 \leq i \leq n$ and $c = \bigcup_{1 \leq i \leq n} c_i \cup \{e_i\}$.

Based on immediate conflicts we can define branching cells:

- A s-prefix B is an (initial) *branching cell*¹¹ if it is a minimal non-empty s-prefix such that, for all $c \in C$, K is an ICS at c and $K \cap B \neq \{\}$ implies $K \subseteq B$.

¹⁰ A prefix D is *conflict-free* if the CS it induces is conflict-free.

¹¹ On pes-bc a subset B of events is called an initial (i.e. at configuration $\{\}$) branching cell, or an initial stopping prefix, if it is a minimal non-empty subset closed under both immediate conflict and downward causality.

Note that we use s-prefix to define branching cells. That is because, if not all causes are included in a branching cell, it might allow the possibility that outside events enable inside events, contradicting the requirements of stubborn sets. Given a branching cell B and the sub-structure $cs \upharpoonright_B$ carved out by B , the evolution of $cs \upharpoonright_B$ is independent to the evolution of the rest of the system:

Lemma 6. *Given a branching cell B of cs , $c \cap B \in C$ and $(cs/c) \upharpoonright_B = (cs/(c \cap B)) \upharpoonright_B$ hold for all $c \in C$.*

Proof. Easy to see $c \cap B \in C$ as, for all $e \in c \cap B$, there is $c_B^e \in IE(e)$ such that $c_B^e \subseteq c$. Obviously $c \cap B = \bigcup_{e \in c \cap B} (c_B^e \cup \{e\}) \in C$ (due to bounded union closure). Assume $(cs/c) \upharpoonright_B = (cs/(c \cap B)) \upharpoonright_B$ is not true. Then there exists $c \in C$, $c_B \subseteq B$ and $e \in E \setminus B$ such that $c_B \in C_{cs/c}$, $c \xrightarrow{e} c'$ and $c_B \notin C_{cs/c'}$. Thus e must be involved in a conflict (of cs/c) with a subset of c_B . Then on a (singleton) path from c to $c \cup c_B$, e will eventually be eliminated by one of its transition $c'' \xrightarrow{e'}$. Obviously, at c'' , $\{e, e'\}$ is an LCS. Contradiction with BC (branching cell) definition!

3.4 Knots and extended knots are branching cells

Branching cells are defined in terms of immediate conflicts, whereas knots are in terms of event elimination. The two ideas, however, are closely linked since any ICS containing events from both inside and outside of a knot can evolve into a binary conflict on a pair of inside and outside events that eliminate each other.

The evolution considered in the branching cell definition is based on events from both inside and outside, whereas that in knot definition is based on outside events only. As a consequence, knots is a local (i.e. dynamic) notion whereas branching cell is a global notion.

Branching cells can contain events that are not yet enabled, whereas knots contain only enabled events. In this sense, knots is less closely linked to branching cells than extended knots is. The downward-causality closure condition (i.e s-prefix) of branching cells appears in extended knots partially as the condition that the enabled subset of inside events can block the rest of inside events (i.e. those not yet enabled).

Theorem 1. *Given a configuration structure $cs = (E, C)$, $B \subseteq E$ is a branching cell of cs iff B is a minimal non-empty s-prefix of cs such that $eb(c) \cap B$, if non-empty, forms a knot of cs/c for all $c \in C$.*

Proof. We first prove that, given any s-prefix B , ICS closure and long-lived knot are equivalent. Then, automatically the minimality of one leads to the minimality of the other.

From ICS closure to long-lived knot: $K_0 = eb(\{\}) \cap B$ is a knot at $\{\}$ since 1) eliminating any member of K_0 requires a binary LCS involving e and a non- B event at some subsequent future reachable by non- B events, 2) mixed binary LCSes implies mixed ICSes and thus contradiction with BC definition! $eb(c) \cap B$

is a knot at $c \in C$ since what remains of B , or the *residues* of B , continues to be branching cells.

From long-lived knot to ICS closure: Assume there is a mixed ICS K at $c \in C$. Select one event e from $K \cap B$ and one event e' from $K \setminus B$. Then there exists $c \subseteq c' \in C'$ such that $\{e, e'\}$ is an LCS at c' . Obviously $eb(c') \cap B$ is neither empty nor a knot of cs/c' . Contradiction!

The above theorem works for knots only when B is assumed to be s-prefix. This restriction, however, can be lifted for the case of extended knots.

Corollary 1. *Given a configuration structure $cs = (E, C)$, $B \subseteq E$ is a branching cell of cs iff B is a minimal non-empty subset of E such that $B \cap E_{cs/c}$ (i.e. the residue of B in cs/c), if non-empty, is an extended knot of cs/c for all $c \in C$.*

Proof. Only need to prove that, given any $B \subseteq E$, long-lived knot are equivalent to BC (i.e ICS closure plus s-prefix).

From BC to long-lived ex-knot: Obvious since $B \cap eb(\{\})$ is a knot and can block $B \setminus eb(\{\})$.

From long-lived ex-knot to BC: If B is not an s-prefix, then we can always find $e \in B$ and $c \in IE(e)$ such that $c \setminus B \neq \{\}$ and $\forall e' \in B \cap c : c_0 \in IE(e') \wedge c_0 \subseteq c \implies c' \subseteq B$. Consequently $c \cap B \in C$ holds and $B \setminus c$ is not an ex-knot of $cs/(c \cap B)$. Contradictions! ICS closure is obvious.

4 Events in LTSes

We now establish a connection between configuration structures and LTSes. Our method follows the tradition of identifying *events* (of a non-interleaving model) with *subsets of transitions* (of a transition system). Given such a subset, the transitions inside it are the occurrences of the same event under different global states. They form a set of parallel edges on a ‘diamond-ful’ subgraph of the transition system. However, unlike previous works which are more interested in the direction from event-based systems to transition systems (e.g. characterising the subclass of transition systems produced by a given type of event-based systems, say one-safe PN), our work focuses more on the direction from transition systems to event-based systems and we use the full class of DLTSes.

Definition 5. *Let $L = (S, \Sigma, \Delta, \hat{s})$ be a DLTS. A labelled CS over L is a tuple (E, C, lb, st) , where*

- (E, C) constitutes a CS
- $lb : E \rightarrow \Sigma$ is a function
- $st : C \rightarrow S$ is a function

A labelled CS $lcs = (E, C, lb, st)$ over L is an *unfolding* of L , if

- (a) $st(\{\}) = \hat{s}$,
- (b) for all $c \in C$, lb maps the set of enabled events at c (i.e. $eb_C(c)$) in an 1-to-1 fashion to the set of enabled actions at $st(c)$ (i.e. $eb_\Delta(st(c))$),

- (c) for all $c \in C$, $c \xrightarrow{e}_C c \cup \{e\}$ implies $st(c) \xrightarrow{lb(e)}_{\Delta} st(c \cup \{e\})$, and
(d) Given any $c' \in C$ and any $e' \in A(c')$ (i.e. one of its previously encountered event), we have $e' \in eb(c')$ iff for all transitions $c \xrightarrow{e}_C c \cup \{e\}$ with $c \cup \{e\} \subseteq c'$, $e \neq e'$ and $lb(e)$ and $lb(e')$ are not in conflict at $st(c)$.

The four conditions in the unfolding definition seem to be natural requirements. The last one essentially says that an enabled event e' at c' should not be in conflict with any $e \in c'$ (i.e. judging from their DLTS mappings). It can be equivalently reformulated as: Given any $c' \in C$ and $e' \in A(c')$, $e' \in eb(c')$ iff, either $c' \in IE(e')$ or, for all $c \xrightarrow{e}_C c'$ with $e' \in A(c)$, event e' is *transferred* on the transition from c to c' , i.e. $e' \in eb(c)$ and $lb(e) \parallel_{st(c)} lb(e')$ (which implies $e \neq e'$). Thus we say c' *inherits* e' if e' is transferred on all $c \xrightarrow{e}_C c'$ with $e' \in A(c)$.

We can now give a procedure $lcs = \mathcal{U}(L)$ to unfold a DLTS (i.e. L) into a labelled CS over L (i.e. lcs).

```

 $\mathcal{U}(L = (S, \Sigma, \Delta, \hat{s}))$ 
1 Set  $i = 0$  and  $lcs = (E, C, lb, st) = (\{\}, \{\{\}\}, \{\}, \{\{\}, \hat{s}\})$ ;
2 Set  $H = \{\{\}, \{\}\}$ ;
   -- function  $H(c)$  gives the set of inherited events at  $c$ 
3 While there exists  $c \in C$  with  $|c| = i$  do
4   Foreach  $c \in C$  with  $|c| = i$  do
5     Foreach transition  $st(c) \xrightarrow{a}_{\Delta} s'$  do
6       If there is  $e \in H(c)$  such that  $lb(e) = a$  Then select  $e$ 
7       Else generate a new event  $e$ , add  $e$  to  $E$  and  $(e, a)$  to  $lb$ , and select  $e$ ;
8       Add  $c \cup \{e\}$  to  $C$  and  $(c \cup \{e\}, s')$  to  $st$ ;
9   Foreach  $c' \in C$  with  $|c'| = i + 1$  do
10    Add  $(c', \{\})$  to  $H$ ;
11    Foreach  $e' \in E \setminus c'$  such that  $e' \in A(c')$  do
12      -- function  $IE(e)$ , similarly  $eb(c)$ , can be derived from  $(E, C)$ 
13      -- even though the current  $(E, C)$  might be just a partial CS
14      If  $\exists c \xrightarrow{e}_C c' : e' \in A(c) \wedge \neg(e' \in eb(c) \wedge lb(e) \parallel_{st(c)} lb(e'))$ 
15        Then continue Else add  $e'$  to  $H(c')$ ;
16    Foreach  $a \in eb(st(c'))$  -- or-causality event merge
17      If  $|A = \{e : H(c') \mid lb(e) = a\}| > 1$  Then substitute a new event  $e''$ 
18      for every member of  $A$  in  $lcs$  and  $H$ ;
19     $i := i + 1$ ;
20 return  $(E, C, lb, st)$ 

```

Fig. 8. An unfolding procedure

We can illustrate the procedure by unfolding a broken cube in Figure 9. In the broken cube parallel edges are labelled by the same action; that is, along the three axes, they are resp. a , b and c .

In the step 0 the initial state is mapped to configuration $\{\}$, on which the set of inherited events $H(\{\})$ is empty. Thus it is labelled as $\{\}/\{\}$ (i.e. $c/H(c)$). Since there is no inherited events to cover (i.e. through sharing of labels) any

of the outgoing transitions of the initial state, three new events $e1$, $e2$ and $e3$ are generated (note the use of symbol !) for the three outgoing transitions (c.f. line 7 of the code). Each transition (say the one labelled by $e1$) leads to a new configuration (i.e. $\{e1\}$), carrying the set of transferred events (i.e. $\{e2, e3\}$). For instance $e2$ is transferred on the transition $\{\} \xrightarrow{e1}_C$ because the a and b transitions at the initial state (corresponding to $e1$ and $e2$ resp.) forms a local diamond. Similarly the transfer of $e3$ by $\{\} \xrightarrow{e1}_C$ is due to the a and c diamond at the initial state.

In the step 1, configuration $\{e1\}$ inherits $\{e2, e3\}$ since it is what is transferred on its only incoming transition. And no new event will be generated at $\{e1\}$ since the labels from $\{e2, e3\}$ are sufficient to cover those of the (two) outgoing transitions at $st(\{e1\})$. At configurations $\{e2\}$ and $\{e3\}$, they similarly inherit two events. However, in further exploration of the graph from $st(\{e2\})$ and $st(\{e3\})$ both encounter non-commutativity for their (pairs of) outgoing transitions. Thus $\{e2\} \xrightarrow{e3}_C$ will not transfer $e1$ and $\{e3\} \xrightarrow{e1}_C$ will not transfer $e2$.

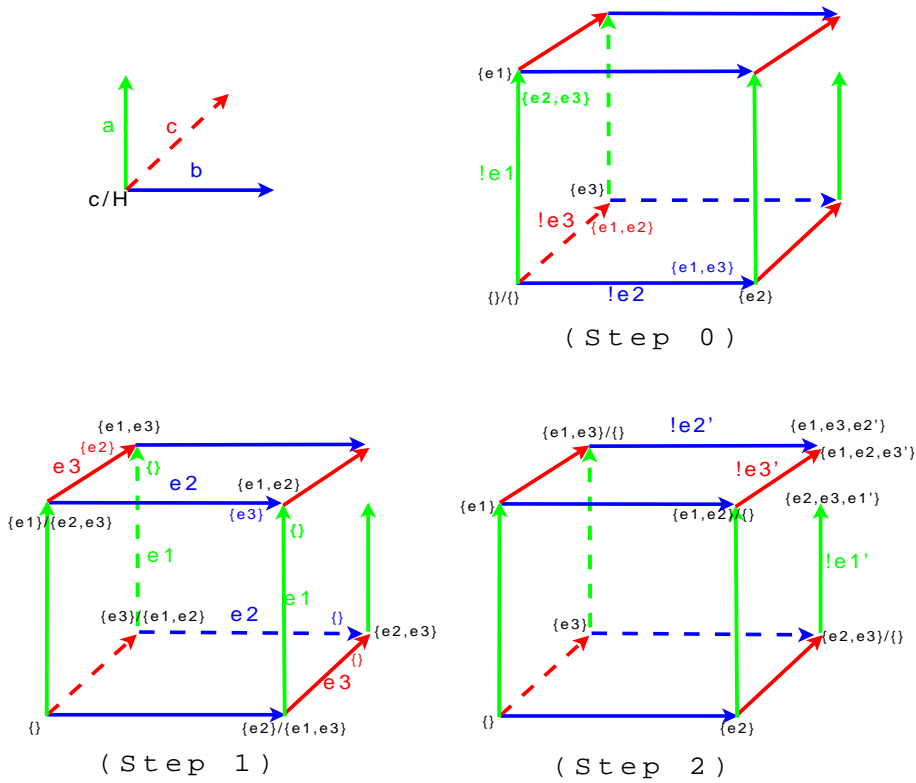


Fig. 9. Unfolding of a broken cube

In the step 2, the set of inherited events at $\{e1, e3\}$ is empty since $e2$ is enabled at $\{e3\}$ but $e2$ is not transferred by $\{e3\} \xrightarrow{e1}$. This vetoes the inheritance of $e2$ at $\{e1, e3\}$, even though $e2$ does get transferred on $\{e1\} \xrightarrow{e3}$. Thus we have to create a second event for action b at $\{e1, e3\}$, i.e. $e2'$. Similarly $e1'$ and $e3'$ are created for a and c again. So the result is a CS with three maximal configurations. Two of them, $\{e1, e3, e2'\}$ and $\{e1, e2, e3'\}$, are mapped to a same deadlock state.

Note that the procedure is non-terminating for cyclic graphs, in which case the output should be an infinite event system.

Proposition 3. $\mathcal{U}(L) = (E, C, lb, st)$ is an unfolding of L .

Proof. Use induction on the size of c' we can prove that st is a function and condition c is true. The induction step needed is that given $c_1 \xrightarrow{e1}_C c'$ and $c_2 \xrightarrow{e2}_C c'$, $st(c_1) \xrightarrow{lb(e1)}_{\Delta} s_1$ implies $st(c_2) \xrightarrow{lb(e2)}_{\Delta} s_2$ and $s_1 = s_2$. To prove the induction step, we need commutativity argument as well as the fact that there exists $c \in C$ such that $c \subseteq c_1 \cap c_2$, $e_1, e_2 \in eb(c)$ and $lb(e_1) \parallel_{st(c)} lb(e_2)$. The fact can be proved because $cs_0 = (E, C) \downarrow_{c'}$ is a conflict-free CS and $e1$ cannot block $e2$.

The above also leads to the fact that any pair of configurations in C with subthood relation must be connected by a path. Based on the fact and using some commutativity argument we can prove closure under bounded union.

Condition b is preserved due to line 6-8 and line 14-16 in the code, while condition d is preserved due to line 11-13 in the code.

Discussion: A major difference of our work from [19, 10, 14] lies in that our procedure dynamically assigns the independence relations to transitions. In the left graph of Figure 10, transitions $s1 \xrightarrow{c} s4$ and $s1 \xrightarrow{b} s5$ are assigned to be independent if $s1$ is reached from $s0$ via d (since the b, c and d transitions form a cube); otherwise (i.e. $s1$ is reached from $s0$ via a) they are assigned to be dependent.¹² The right graph of Figure 10 gives the unfolding of the left graph. Let it be cs . It is obvious that $s1 \xrightarrow{c} s4$ and $s1 \xrightarrow{b} s5$ are assigned to be $e3$ and $e2$ resp. but $\{e2, e3\}$ is consistent in $cs/\{e4\}$ ($s1$ is reached via d) and inconsistent in $cs/\{e1\}$ ($s1$ is reached via a).

5 Correspondence

Based on the unfolding of L into lcs , we can establish the correspondence between extended knots of lcs and stubborn sets of L .

Given $lcs = (E, C, lb, st)$, we say lcs is free of *auto-concurrency* if, for any pair of events $e, e' \in E$ with $lb(e) = lb(e')$, there is no $c \in C$ such that $c \xrightarrow{\{e, e'\}}_C$. We say lcs is free of *auto-conflict* if, for all $c \in C$, there is no ICS K at c so that K contains a pair of events with the same label.

¹² Note that this is exactly the situation in Figure 9.

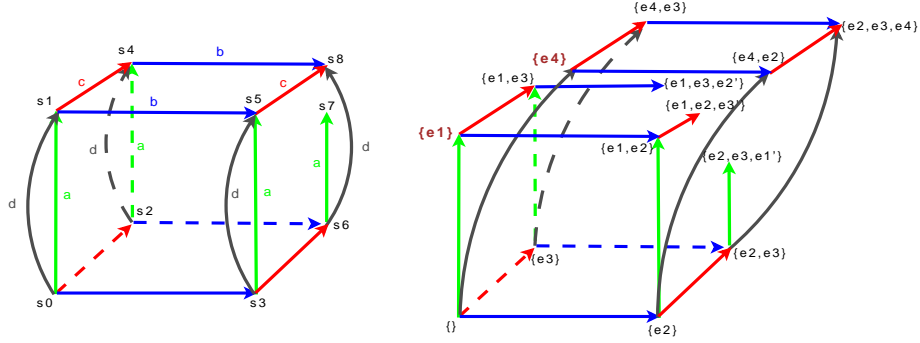


Fig. 10. Dynamic independence on transitions

Proposition 4. $lcs = (E, C, lb, st)$ is an unfolding of L implies lcs is free of auto-concurrency and auto-conflict.

Proof. It is obvious since e and e' are contained in an ICS K at c implies both are enabled at c . Due to requirement b , i.e. bijection, in the unfolding definition, no two elements of K can have the same label. Similarly we can prove freedom of auto-concurrency.

Theorem 2. Given any $L = (S, \Sigma, \Delta, \hat{s})$ and any unfolding $lcs = (E, C, lb, st)$ of L , $T \subseteq \Sigma$ is a persistent set at \hat{s} iff there is a knot $K_0 \subseteq E$ such that $lb(K_0) = T$.

Proof. From Knots to PS (Persistent Set): Let K_0 be a knot of lcs , $T = lb(K_0)$ and $e \in eb(\{\}) \setminus K_0$ be an initial event outside the knot. Then at state $st(\{\}) = \hat{s}$, $lb(e)$ and $lb(e')$ are conditional independent for all $e' \in K_0$ (since $\{e, e'\}$ is consistent due to K_0 being a knot). Since lb is a bijection between $eb(\{\}) \setminus K_0$ and $eb(\hat{s}) \setminus T$ the configurations reachable by executing an event from $eb(\{\}) \setminus K_0$ covers all the states reachable by executing an action from $eb(\hat{s}) \setminus T$. For instance, say $\{e\}$ with $e \in eb(\{\}) \setminus K_0$ covers s with $\hat{s} \xrightarrow{lb(e)} s$. Obviously $K_0 \subseteq eb(\{e\})$. From s on every state reachable by executing one step of $eb(s) \setminus T$ transition is covered by some configuration $\{e, e'\}$ with $e' \in eb(\{e\}) \setminus K_0$ due to the bijection. By induction we can say every state s' reachable from \hat{s} by executing $\Sigma \setminus T$ transitions is covered by a configuration c with $c \cap K_0 = \{\}$ and on all such s' any pair of enabled inside (w.r.t. T) and enabled outside transitions are conditionally independent. This combined with the fact that all T transitions are enabled on such s' implies T is a PS.

From PS to Knots: Let $T (\subseteq eb(\hat{s}))$ be a PS at \hat{s} of L and K_0 be the set of events mapped to T by lb from $eb(\{\})$ of lcs . Let us assume K_0 is not a knot. Then there is an event $e \in K_0$ and a configuration $c \in C$ with $c \cap K_0 = \{\}$ such that $c \cup \{e\}$ is inconsistent. Then on the path from $\{\}$ to c , we can find $e' \in c$ and a configuration $c_0 \subseteq c$ such that $c_0 \xrightarrow{e'}$ disables e . Due to freedom of auto-concurrency and auto-conflict, all the events executed on the path are

events labelled by actions from outside T . Thus there is a corresponding path in L from \hat{s} to $st(c)$ which executes only actions from outside T . At $st(c)$, $lb(e)$ is disabled, which contradict T being a PS.

Corollary 2. *Given any $L = (S, \Sigma, \Delta, \hat{s})$ and any unfolding $lcs = (E, C, lb, st)$ of L , $T \subseteq \Sigma$ is a stubborn set at \hat{s} implies there is an extended knot $K_1 \subseteq E$ such that $lb(K_1) = T$.*

Proof. Let T be a SS at \hat{s} and K_0 be the set of events labelled by $T \cap eb(\hat{s})$ at $\{\}$. We only need to prove that, for all $a \in T \setminus eb(\hat{s})$, if $a = lb(e)$, then K_0 can block e . It is obvious since otherwise a can be enabled without taking any transition labelled by $lb(K_0)$.

Note that the implication above holds only in one direction: some extended knot may not correspond to any stubborn set. However, thanks to the proposition below, this observation does not change the fact that *the set of actions produced by a residue are indeed those produced by an extended knot that corresponds to a stubborn set*. A branching cell is therefore a long-lived stubborn set.

Proposition 5. *Given any unfolding $lcs = (cs, lb, st)$ of L and any branching cell B of cs , we have, for all $a \in lb(B) \setminus lb(B \cap eb(\{\}))$ and $e \in E$, $lb(e) = a$ implies $B \cap eb(\{\})$ can block e .*

Proof. Use Lemma 6 and the freedom of auto-concurrency.

6 Conclusion

In this paper we have shown that the theory of partial order reduction developed for interleaving setting can be lifted into the world of non-interleaving. A stubborn set (or persistent set) corresponds to an extended knot (or knot), a new notion we developed using well-formed configuration structures. Extended knots are closely related to the existing notion of branching cells from non-interleaving literatures. We prove that a branching cell is actually a ‘long-lived’ extended knot. Thus we establish an important link between non-interleaving semantic theory and partial order reduction, making possible cross-fertilisation for future works.

Acknowledgements We thank anonymous referees whose detailed comments greatly improve the presentation of the paper.

References

1. Samy Abbes and Albert Benveniste. True-concurrency probabilistic models: Branching cells and distributed probabilities for event structures. *Information and Computation*, 204(2):231–274, 2006.
2. E. M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *STTT*, 2(3):279–287, 1999.

3. P. Godefroid. *Partial-order Methods for the Verification of Concurrent Systems: an Approach to the State-explosion Problem*, volume 1032 of *LNCS*. Springer, 1996.
4. J.F. Groote and M. P. A. Sellink. Confluence for process verification. *Theoretical computer science*, 170(1-2):47–81, 1996.
5. Jeremy Gunawardena. Causal automata. *TCS*, 101(2):265–288, 1992.
6. Xinxin Liu and David Walker. Partial confluence of processes and systems of objects. *TCS*, 206(1-2):127–162, 1998.
7. A Mazurkiewicz. Trace theory. In *Petri Nets: Applications and Relationships to Other Models of Concurrency*, volume 255 of *LNCS*, pages 279–324. Springer, 1987.
8. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. In *Semantics of Concurrent Computation*, volume 70 of *LNCS*, pages 266–284. Springer, 1979.
9. D. A. Peled. All from one, one for all: on model checking using representatives. In *Proceedings of the 5th International Conference on Computer Aided Verification*, volume 697 of *LNCS*, pages 409–423. Springer, 1993.
10. Vladimiro Sassone, Mogens Nielsen, and Glynn Winskel. Models for concurrency: Towards a classification. *Theor. Comput. Sci.*, 170(1-2):297–348, 1996.
11. A. Valmari. Stubborn sets for reduced state space generation. In *ICATPN*, volume 483 of *LNCS*, pages 491–515. Springer, 1989.
12. A. Valmari. The state explosion problem. In *Lectures on Petri Nets I: Basic Models*, volume 1491 of *LNCS*, pages 429–528. Springer, 1998.
13. Antti Valmari. Stubborn set methods for process algebras. In *POMIV'96*, volume 29 of *DIMACS Series in DM&TCS*, pages 213–231, 1997.
14. R. J. van Glabbeek. History preserving process graphs. available at <http://kilby.stanford.edu/rvg/pub/history.draft.dvi>, 1996.
15. R. J. van Glabbeek and G. D. Plotkin. Configuration structures, event structures and petri nets. *Theoretical Computer Science*, 410(41):4111–4159, 2009.
16. Rob van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Informatica*, 37(4):229–327, 2001.
17. Daniele Varacca, Hagen Völzer, and Glynn Winskel. Probabilistic event structures and domains. In *CONCUR*, volume 3170 of *LNCS*, pages 481–496. Springer, 2004.
18. Glynn Winskel. Event structures. In *Advances in Petri Nets*, volume 255 of *LNCS*, pages 325–392. Springer, 1987.
19. Glynn Winskel and Mogens Nielsen. Models for concurrency. In *Handbook of logic in Computer Science*, volume 4. Clarendon Press, 1995.
20. A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with or causality. *FMSD*, 9(3):189–233, 1996.