**Formal Methods in System Design manuscript No.**
(will be inserted by the editor)

# Compact and Efficiently Verifiable Models for Concurrent Systems

**Hernán Ponce-de-León · Andrey Mokhov**

Received: date / Accepted: date

**Abstract** Partial orders are a fundamental mathematical structure capable of representing concurrency and causality on a set of atomic events. In many applications it is essential to consider multiple partial orders, each representing a particular behavioral scenario or an operating mode of a system. With the exploding growth of the complexity of systems that software and hardware engineers design today, it is no longer feasible to represent each partial order of a large system explicitly, therefore compressed representations of sets of partial orders become essential for improving the scalability of design automation tools. In this paper we study two well known mathematical formalisms capable of the compressed representation of sets of partial orders: Labeled Event Structures and Conditional Partial Order Graphs. We discuss their advantages and disadvantages and propose efficient algorithms for transforming a set of partial orders from a given compressed representation in one formalism into an equivalent representation in another formalism without explicitly enumerating every partial order. The proposed algorithms make use of an intermediate mathematical formalism which we call Conditional Labeled Event Structures that combines the advantages of both structures. Finally, we compare these structures on a number of benchmarks coming from concurrent software and hardware domains.

**Keywords** Concurrency · Graph transformation · Partial orders · Event structures

## 1 Introduction

*Concurrent systems*, which comprise multiple components that can operate and interact simultaneously, are notoriously difficult to design, control and reason about.

Hernán Ponce-de-León
fortiss GmbH, Guerickestraße 25 80805 München Germany Tel.: +49 (89) 3603522 587 Fax: +49 (89) 3603522 50 E-mail: ponce@fortiss.org

Andrey Mokhov
School of Engineering, Merz Court, Newcastle University, Newcastle upon Tyne, NE1 7RU, UK Tel.: +44 (0) 191 208 7727 Fax: +44 (0) 191 208 8180 E-mail: andrey.mokhov@ncl.ac.uk

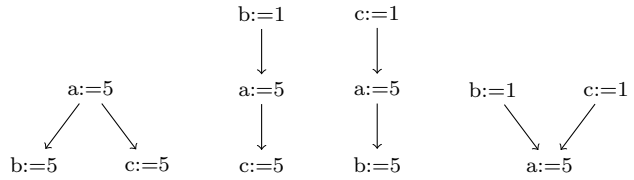


Fig. 1: Executions of a multithreaded program represented as partial orders.

The complexity of concurrent systems grows exponentially with the number of constituent components [10] and a lot of research effort is currently being invested in finding new methods for *conquering the complexity* by academia and industry [5]. Potential impact of this research is truly ubiquitous since concurrent systems are everywhere around, inside and above us: from common electronic gadgets like mobile phones, to biological and biomedical systems, to planetary exploration robots. Our society's sustainability is increasingly dependent on concurrent systems and the goal of concurrency theory is to lay foundations for the design of correct and efficient concurrent systems.

Partial orders – the protagonists of this paper – play a fundamental role in the concurrency theory. A partial order is a reflexive, antisymmetric and transitive relation $\leq$ on a set of elements $S$. Two distinct elements $a, b \in S$ can be either ordered ($a \leq b$ or $b \leq a$) or independent ($a \not\leq b$ and $b \not\leq a$). Partial orders arise in numerous application areas such as model checking, analysis of concurrent programs and VLSI design to name but a few. In this paper we do not focus on a particular application area, however, we use examples and benchmarks where partial orders represent possible execution paths of multithreaded programs and concurrent activations of hardware components in a microprocessor.

Consider a multithreaded program where different threads access some shared variables. Direct representation of all its possible execution paths is infeasible even for very small programs due to the well-known *state space explosion problem* [29]. If one intends, for example, to detect errors such as assertion violations or deadlocks, it is sufficient to analyze only one representative for each Mazurkiewicz trace [7], whereby the independence between some instructions is taken into account (e.g., two concurrent read instructions accessing the same shared variable). Each execution path can therefore be represented as a partial order. Consider the following program with three threads accessing a shared variable $a$:

```
int a = 1;   Thread 1:      Thread 2:    Thread 3:
             local b = a;   a = 5;       local c = a;
```

The values of local variables $b$ and $c$ depend on whether `Thread` 1 and `Thread` 3 are executed before or after `Thread` 2 modifies the value of the shared variable. The three instructions can be executed in six different orders; however, since consecutive reads of the same variable do not alter the final values of the local variables, they can be considered as independent thereby reducing the number of possible executions to four. These executions are represented by the partial orders in Fig. 1.

A partial order can capture a single Mazurkiewicz trace of a concurrent system, however, real-life systems rarely consist of only one such a trace. Modern concurrent programs targeted for many-core hardware platforms often exhibit a large number of different possible executions that can be represented as partial orders

defined on a set of instructions the program may perform. How do we represent all of those partial orders? One can, of course, simply list them explicitly as we just did in Fig. 1, but this is clearly not a scalable solution – 6.6 trillion different partial orders can be defined on just 10 actions! This motivated computer scientists to search for compact representations for families of partial orders. *Petri Nets* [8] and numerous *process algebras* [15] provide the most compact representations; however, their compactness comes at the price of high computational complexity of the associated problems. Almost any interesting question[1] about a Petri Net is at least PSPACE-hard [8].

*Petri Net unfoldings* [14] and later more condensed *merged processes* [13] were introduced to address this issue: they are less compact but the associated questions are 'merely' NP-complete and hence can be efficiently resolved in practice by modern SAT-solvers. The advances in Boolean satisfiability solving technology have made it possible to handle concurrent systems of real-life size, thereby motivating the continued research focus on models whose properties are NP-complete. For example, Petri Net unfoldings and event structures have recently been employed as a compact representation for execution paths of multithreaded programs to apply partial order reduction techniques [12,26,28].

In this paper we study two mathematical formalisms that *compactly* represent sets of partial orders[2] while still allowing *efficient* verification techniques: Labeled Event Structures (LESs) and Conditional Partial Order Graphs (CPOGs). We introduce the formalisms in Sections 2.1 and 2.2 respectively and show how to synthesize them from sets of partial orders. We also demonstrate that both formalisms are compositional, that is, one can combine compressed sets of partial orders into bigger sets without uncompressing them. Even though these formalisms are not the most compact ones to represent concurrent behaviors (as mentioned above, Petri nets or process algebras are more compact in general), they are supported by a rich collection of theoretical results and tools, making them convenient for the representation and verification of concurrent systems.

The two formalisms are significantly different from each other, hence one cannot directly use them together: conversion from one formalism to another without an intermediate uncompression step is non-trivial. As will be demonstrated in Section 4, different formalisms may be preferable in different application domains. For example, LESs can typically be obtained from Petri Net specifications via unfolding or from concurrent programs as explained in [12,26]. CPOGs naturally come from hardware specifications and implementations, where partial orders are pre-encoded with Boolean vectors (control signals, instruction opcodes, etc).

There are several algorithms and tools that allow to reason about either LESs or CPOGs, each of them having advantages due to the properties of each formalism (e.g. the acyclicity of LESs). In order to allow the end user to exploit the trade-off between their compactness and efficiency, we believe that it is crucial to have efficient transformation algorithms that allow to go from one representation of partial orders to another, therefore integrating into multi-formalism design automation tools such as Workcraft [1,22]. For example, to translate a given CPOG to a Petri net, one can first convert a CPOG into a LES using the algorithm

[1] Examples of interesting questions about a concurrent system are: *Can the system reach this dangerous state? Will it work indefinitely or eventually terminate? Are these two systems equivalent?*

[2] We call a representation 'compact' when it does not list all partial orders explicitly.

presented in this paper and then apply an existing transformation algorithm from LESs to Petri nets, e.g. from [24]. We present two direct transformation algorithms (Section 5) for converting compressed sets of partial orders from LESs to CPOGs and from CPOGs to LESs without an intermediate uncompression. The presented transformations reveal the superior expressive power of CPOGs as well as the cost of this expressive power: CPOG algorithms often have higher computational complexity. The proposed algorithms make use of a new mathematical formalism called Conditional Labeled Event Structures (CLESs) that combines the advantages of LESs and CPOGs. The CLES formalism makes it possible to directly combine sets of partial orders represented in LESs and CPOGs, thereby improving their interoperability and compositionality.

This paper extends our previous work [23]; the new contributions are: *(i)* a synthesis algorithm for LES given a set of partial orders; *(ii)* a merging algorithm to reduce the complexity of a LES while preserving the partial orders it represents; *(iii)* complete proofs of the correctness, optimality and compositionality results of the transformation algorithms; *(iv)* optimization techniques for one of the transformation algorithms; *(v)* experimental evaluation of the presented algorithms on a realistic set of examples including benchmarks coming from the domains of multithreaded programming and microprocessor design.

## 2 Preliminaries

This section introduces two formalisms that are used represent partial orders in this paper: Labeled Event Structures and Conditional Partial Order Graphs.

### 2.1 Labeled Event Structures

*Event Structures*[3] [21] can be seen as a generalization of trees where each branch is a partial order. They represent multiple scenarios of a concurrent system by means of so-called *configurations*. We study their widely-used extension, called *Labeled Event Structures*, whose events are labeled with actions from a fixed alphabet $L$.

**Definition 1** A *labeled event structure* over alphabet $L$ is a tuple $\mathcal{E} = (E, \leq, \#, \lambda)$ where $E$ is a set of events; $\leq \subseteq E \times E$ is a partial order (called *causality*) satisfying the property of *finite causes*, i.e. $\forall e \in E : |\{e' \in E \mid e' \leq e\}| < \infty$; $\# \subseteq E \times E$ is an irreflexive symmetric relation (called *conflict*) satisfying the property of *conflict heredity*, i.e. $\forall e, e', e'' \in E : e \# e' \wedge e' \leq e'' \Rightarrow e \# e''$; and $\lambda : E \to L$ is a labeling function.

In most cases one only needs to consider reduced versions of relations $\leq$ and $\#$, which we will denote $\leq_r$ and $\#_r$, respectively. Formally, $\leq_r$, which we call *direct causality*, is the transitive reduction of $\leq$, and $\#_r$, which we call *direct conflict* is the smallest relation inducing $\#$ through the property of conflict heredity. In practice $|\leq_r|$ and $|\#_r|$ are often a lot smaller than $|\leq|$ and $|\#|$, however, in the worst case $|\leq_r| = \Theta(|\leq|)$ and $|\#_r| = \Theta(|\#|)$, therefore the speed up gained by using

[3] In this article, we restricts to prime event structures.

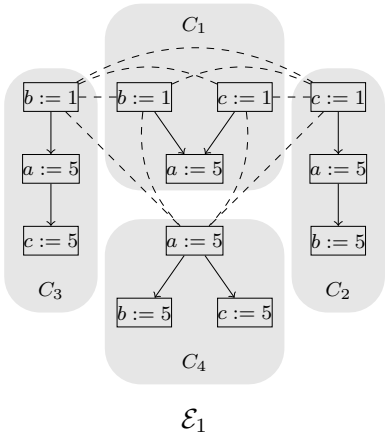


Fig. 2: A Labeled Event Structure and its maximal configurations.

the reduced relations does not affect the worst case performance of the presented algorithms.

A configuration is a computation state of a LES. It is represented by a set of events that have occurred in the computation. If an event is present in a configuration, then so must all the events on which it causally depends. Moreover, a configuration does not contain conflicting events.

**Definition 2** A *configuration* of a LES $\mathcal{E} = (E, \leq, \#, \lambda)$ is a set $C \subseteq E$ that is *causally closed*, i.e. $e \in C \Rightarrow \forall e' \leq e : e' \in C$, and *conflict-free*, i.e. $e \in C$ and $e \# e'$ imply $e' \notin C$. The set of maximal (w.r.t. set inclusion) configurations of $\mathcal{E}$ is denoted by $\Omega(\mathcal{E})$. The *local configuration* $[e]$ of an event $e$ is a set of events on which it causally depends, i.e. $[e] \triangleq \{e' \in E \mid e' \leq e\}$.

In this paper we only deal with LESs whose configurations do not contain two events with the same label. With such a restriction one can associate to every configuration $C$ a partial order whose elements are $\lambda(C)$ (where $\lambda$ is lifted to sets) and causality is inherited from $\leq$. We will denote such partial order as $\pi(C)$ and lift $\pi$ to sets of configurations. Since configurations together with the causality relation (restricted to those events) form partial orders, one can consider a LES $\mathcal{E}$ as a compressed representation of the set of partial orders induced by the maximal configurations $\Omega(\mathcal{E})$.

Fig. 2 shows an example of a LES defined over the alphabet $\{a := 5, b := 1, b := 5, c := 1, c := 5\}$ and representing the partial orders of Fig. 1. This LES contains four maximal configurations $C_1$-$C_4$. Notice that throughout this paper we only show direct causality (by arrows) and direct conflicts (by dashed lines) on diagrams for clarity: events that belong to different configurations $C_1$-$C_4$ are all in conflict pairwise, and showing all these conflicts would make the diagram unreadable. Not much compression is achieved by the LES shown in Fig. 2; indeed, this can be significantly improved as discussed below.

**Synthesis and Optimization of LESs.** Given a set of partial orders, the objective is to synthesize a compact LES (one with as few events as possible) that

**Algorithm 1** LES optimization

**Require:** An augmented LES $\mathcal{E} = (E, \leq, \#, \lambda)$ (see Remark 1)
**Ensure:** $\mathcal{E}'$ such that $\pi(\Omega(\mathcal{E})) = \pi(\Omega(\mathcal{E}'))$
1: **while** $\exists e_1, e_2 \in E : e_1 \# e_2 \wedge \lambda(e_1) = \lambda(e_2) \wedge [e_1] = [e_2]$ **do**
2: $E = E \backslash \{e_1, e_2\} \cup \{e\}$ for $e \notin E$
3: $\lambda(e) = \lambda(e_1)$
4: **for** $e' \in E$ **do**
5: **if** $e' \leq e_1 \vee e' \leq e_2$ **then**
6: set $e \leq e'$
7: **if** $e_1 \leq e' \vee e_2 \leq e'$ **then**
8: set $e' \leq e$
9: **if** $e' \# e_1 \wedge e' \# e_2$ **then**
10: set $e \# e'$
11: **return** $\mathcal{E} = (E, \leq, \#, \lambda)$

represents them. The idea behind the synthesis approach presented below is to create a LES whose events and causality relations are the union of all the partial orders (assuming the set of the events are disjoint) and where a conflict is added for any two events coming from different partial orders (see Fig. 2). The next step is to compress the LES by merging events with the same label that also have the same local configuration – see Algorithm 1.

*Remark 1* Merging events as described above may lead to a situation, when a previously maximal configuration ceases to be maximal. For example, if we start with two partial orders $po_1 = (\{a\}, \emptyset)$ and $po_2 = (\{a, b\}, \{a \leq b\})$, each represented by a corresponding maximal configuration, and then merge events $a$, then configuration $\{a\}$ ceases to be maximal being superseded by configuration $\{a, b\}$, thereby leading to removal of $po_1$ from the set of partial orders. To avoid this we add a maximal event $\top$ to each partial order: $po'_1 = (\{a, \top\}, \{a \leq \top\})$ and $po'_2 = (\{a, b\}, \{a \leq b, a \leq \top, b \leq \top\})$. Now configuration $\{a, \top\}$ will remain maximal because the events labeled by $\top$ will never get merged since they have different local configurations. In the rest of the paper we assume all partial orders to be augmented with $\top$, which we will usually omit in the diagrams.

Fig. 3 illustrates the application of Algorithm 1 to the LES from Fig. 2. One can see that the compressed LESs at each step are smaller. In fact, the resulting LES is the smallest (w.r.t the number of events) that can represent this set of partial orders.

Line 1 in Algorithm 1 generates equivalence classes between the events to be merged. Consider the following equivalence relation:

$$e_1 \equiv_{A_1} e_2 \text{ iff } \lambda(e_1) = \lambda(e_2) \wedge [e_1] = [e_2].$$

The algorithm uses it to generate a LES with one event per equivalence class. Two classes (events) are related by causality if there is at least one representative event in each equivalence class for which the relation holds; two classes are related by conflict if there are representatives for both classes where the relation holds. Algorithm 1 is correct in the sense that the resulting LES represents the same set of partial orders as the original LES.

**Theorem 1** *Let $\mathcal{E}$ and $\mathcal{E}'$ be the input and output event structures of the Algorithm 1, respectively, then $\pi(\Omega(\mathcal{E})) = \pi(\Omega(\mathcal{E}'))$.*

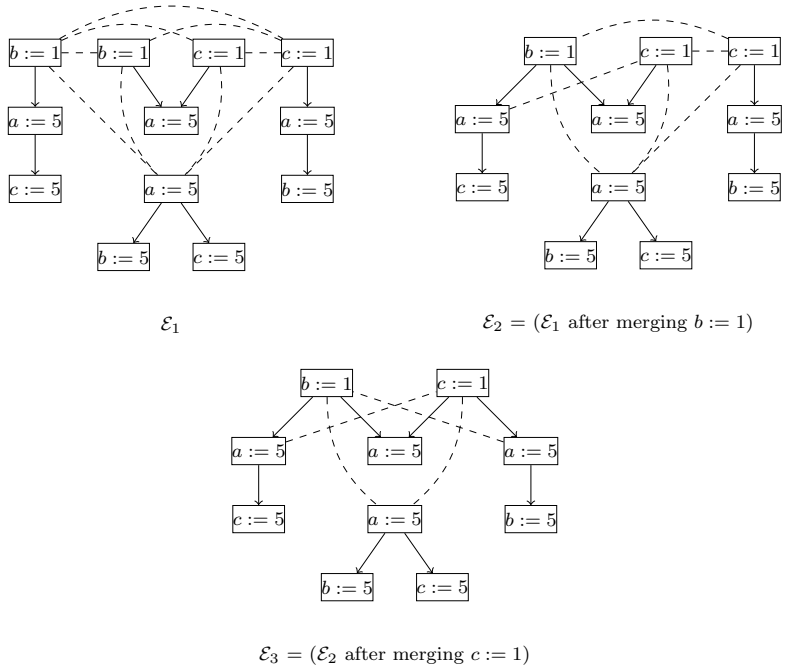


Fig. 3: Compressing a LES by merging events.

*Proof* The results follows from [3] where it has be proven that maximal configurations of the event structure having as events the equivalence classes defined by $\equiv_{A_1}$ coincide with the original set of partial orders when one assumes that each partial order has a maximal $\top$ event. □

One can use the same merging procedure for combining two sets of partial orders $S_1$ and $S_2$ represented by LESs. This is done by creating a LES whose events, causality and conflict relations are the union of both LESs (assuming the set of the events are disjoint) and where a conflict is added for any two events coming from a different LES. Finally this LES is compressed using Algorithm 1. The resulting LES represents partial orders in $S_1 \cup S_2$. Notice that it is not required to uncompress the set of partial orders, hence we argue that LESs have good compositionality.

## 2.2 Conditional Partial Order Graphs

Conditional Partial Order Graphs [20] were introduced for the compact specification of concurrent systems comprising multiple behavioral scenarios.

**Definition 3** A *Conditional Partial Order Graph* is a tuple $H = (V, A, X, \phi, \rho)$, where $V$ is a set of vertices, $A$ is a set of arcs between them, and $X$ is a set of Boolean variables. An *opcode* is an assignment $(x_1, x_2, \ldots, x_{|X|}) \in \{0,1\}^{|X|}$ of

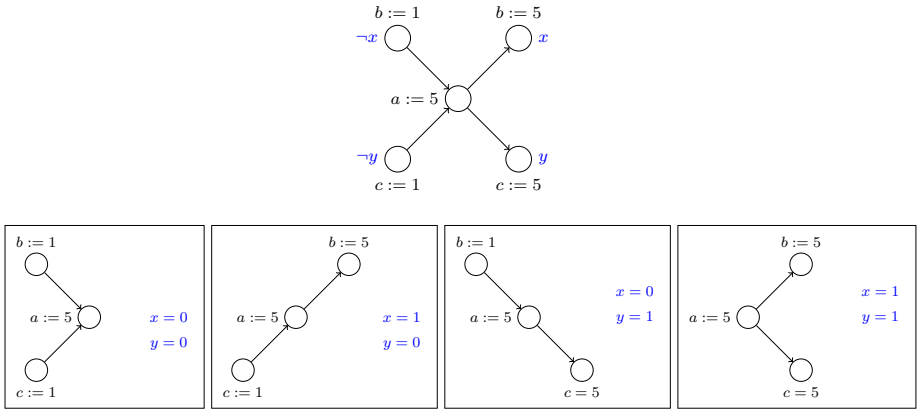


Fig. 4: Conditional Partial Order Graph and the corresponding set of partial orders

these variables; $X$ can be assigned only those opcodes which satisfy the restriction function $\rho$ of the graph, i.e., $\rho(x_1, x_2, \ldots, x_{|X|}) = 1$. Function $\phi$ assigns a Boolean condition $\phi_z$ to every vertex and arc $z \in V \uplus A$ of the graph.

Fig. 4 (above) shows an example of a CPOG which contains $|V| = 5$ vertices and $|A| = 4$ arcs; there are two variables $x$ and $y$; the restriction function is $\rho = 1$, hence, four opcodes $x, y \in \{0, 1\}$ are allowed. Vertices and arcs labeled by 1 are called unconditional (conditions equal to 1 are not depicted in the graph); conditions are shown next to vertices.

The purpose of vertex and arc conditions is to 'switch off' some vertices and/or arcs in the graph according to the given opcode. This makes CPOGs capable of containing multiple *projections* as shown in Fig. 4 (below). If we keep in the graph only those vertices and arcs whose conditions evaluate to Boolean 1 after substitution of the operational variables $x$ and $y$ with Boolean 0, vertices $b := 5$ and $c := 5$ disappear. The arcs $a := 5 \to b := 5$ and $a := 5 \to c := 5$ also disappear because some of their vertices are not part of the remaining graph. Each projection is treated as a partial order specifying a behavioral scenario of a modeled system. Potentially, a CPOG $H = (V, A, X, \phi, \rho)$ can specify an exponential number of different partial orders on vertices $V$ according to $2^{|X|}$ possible opcodes.

We will use notation $H_{|\psi}$ to denote a projection of a CPOG $H$ under opcode $\psi = (x_1, x_2, \ldots x_{|X|})$. A projection $H_{|\psi}$ is called *valid* if opcode $\psi$ is allowed by the restriction function, i.e. $\rho(x_1, x_2, \ldots x_{|X|}) = 1$, and the resulting graph is acyclic. The latter requirement guarantees that the graph defines a partial order. A CPOG $H$ is *well-formed* if every allowed opcode produces a valid projection. The graph $H$ in Fig. 4 is well-formed, because $H_{|x,y=0}, H_{|x=0,y=1}, H_{|x=1,y=0}$ and $H_{|x,y=1}$ are valid. A well-formed graph $H$ defines a set of partial orders $P(H)$.

CPOGs have good compositionality: two given graphs $H_1 = (V_1, A_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, A_2, X_2, \rho_2, \phi_2)$ can be combined using the *overlay operation* in the *algebra of graphs* [19] into $H = (V_1 \cup V_2, A_1 \cup A_2, X_1 \cup X_2, \rho_1 + \rho_2, \phi)$, where conditions are defined as $\forall z \in V_1 \cup V_2 \cup A_1 \cup A_2 : \phi_z \triangleq \rho_1 \phi_{1z} + \rho_2 \phi_{2z}$. The result contains the

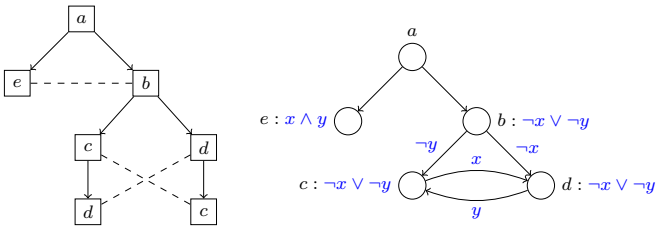


Fig. 5: A LES and a CPOG representing the same scenarios.

union of the sets of partial orders defined by $H_1$ and $H_2$, i.e. $P(H) = P(H_1) \cup P(H_2)$. For further details on CPOG composition, see [18][19].

**Complexity.** The original definition of CPOG complexity [20] is simply the total count of literals used in all the conditions: $\sum_{e \in V \uplus A} |\phi_e|$, where $|\phi|$ denotes the count of literals in condition $\phi$, e.g., $|\neg x \wedge \neg y| = 2$ and $|1| = 0$. The complexity of the CPOG shown in Fig. 5 is thus equal to 12 according to this definition. We argue that this definition is not very useful in practice, because it does not take into account the fact that some of the conditions coincide and can therefore be shared. Intuitively, since $\phi_b = \phi_c = \phi_d = \neg x \vee \neg y$ we can compute condition $\neg x \vee \neg y$ only once and reuse the result three times. Furthermore, one can notice that conditions $\phi_b = \neg x \vee \neg y$ and $\phi_e = x \wedge y$ are not very different from each other; in fact $\phi_b = \neg \phi_e$, therefore having computed $\phi_e$ we can efficiently compute $\phi_b$ by a single inversion operation. In Section 4 we introduce an improved measure of complexity (based on Boolean circuits) which is free from the above shortcomings. The new complexity measure more adequately reflects how CPOG conditions are used in hardware design, where they are synthesized into circuits that share intermediate terms [16]. Similarly, in software CPOG conditions are represented either by using Binary Decision Diagrams or by encoding them into a CNF Boolean formula [17], both of which also permit sharing of intermediate terms [30].

## 3 Enriched and Conditional LES

A LES can represent several partial orders by means of its maximal configurations. CPOGs provide an additional mapping between partial orders and the corresponding opcodes, that is, given an opcode $\psi$ satisfying the restriction function of a well-formed CPOG $H$, one can obtain the corresponding partial order as a projection $H_{|\psi}$. In order to compare LESs into CPOGs we therefore need to enrich the definition of a LES with additional information.

### 3.1 Enriched Labeled Event Structures

Partial orders are represented by maximal configurations of a LES, therefore, in order to extract a partial order from it, one needs to resolve conflicts[4] in a certain

[4] Notice that it is enough to consider direct conflicts only.

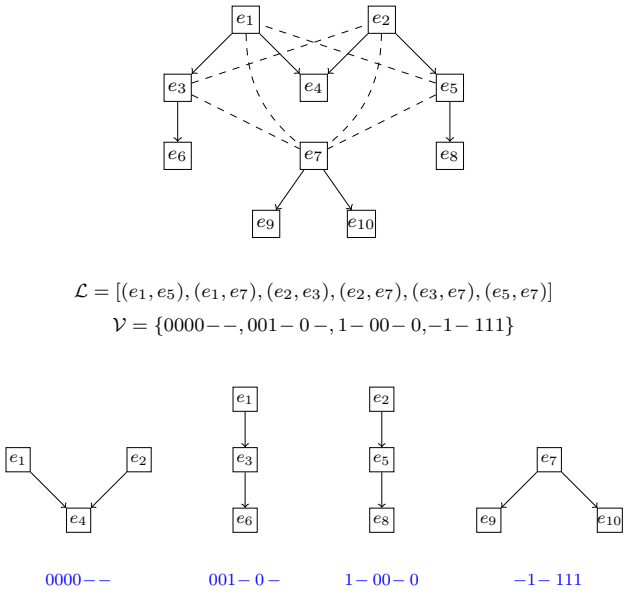


Fig. 6: An ELES and its conflict solvers

way. We enrich LESs with a total order on the conflicts and restrict the way conflicts can be resolved, leading to *Enriched Labeled Event Structures*.

**Definition 4** An Enriched Labeled Event Structure (ELES) over an alphabet $L$ is a tuple $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$ where $(E, \leq, \#, \lambda)$ is a labeled event structure over alphabet $L$, $\mathcal{L}$ is a total order on $\#_r$ and $\mathcal{V}$ is a set of vectors $\{v \mid v \in \{0,1\}^{|\mathcal{L}|}\}$.

A *conflict solver* is a vector $v \in \{0,1\}^{|\mathcal{L}|}$ that determines which event is chosen in each pair of conflicting events. Let $\mathcal{L}[i]$ represents the $i^{th}$ element (which is a pair) in the total order and $\mathcal{L}[i][j]$ with $j \in \{0,1\}$ one of the elements of the pair. Then conflict $\mathcal{L}[i]$ is resolved by the value $v[i]$. Notice that not every conflict solver is acceptable as illustrated in Fig. 6: any solver that chooses $e_3$ over $e_7$ and $e_2$ must also choose $e_1$ because $e_5$ is in future of $e_2$ and cannot be chosen. This is not the only restriction to take into account: if an event is a part of more than one conflict, whenever we choose it w.r.t. one conflict, we must also choose it w.r.t. to the others. For example if $e_3$ is chosen in the conflict $(e_2, e_3)$, it should also be chosen in $(e_3, e_7)$. Let $\overline{E}$ denote events that are not selected by a conflict solver $v$, i.e. $\overline{E} = \{e \in E \mid \exists i, j : v[i] = j \ \wedge \ \mathcal{L}[i][1-j] = e\}$, then the conflict solver is *valid* if it generates a maximal configuration, i.e. $E \backslash \lfloor \overline{E} \rfloor \in \Omega(\mathcal{E})$. The set $\mathcal{V}$ in the definition of ELESs contains all valid conflict solvers.

**Proposition 1** *Let $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$ be an ELES such that for every $v \in \mathcal{V}$, if $v[i] = j$ and $\mathcal{L}[i][j] = e$, then $\forall h, k : \mathcal{L}[h][k] = e$ implies $v[h] = k$ and $\forall e' \in [e], h, k : \mathcal{L}[h][k] = e'$ implies $v[h] = k$. Then $\mathcal{V}$ is a set of valid conflict solvers.*

*Proof* Consider $\mathcal{V}$ satisfying the hypothesis of the proposition and suppose there exists $v' \in \mathcal{V}$ which is not valid. We have then that $E \backslash \lfloor \overline{E} \rfloor \notin \Omega(\mathcal{E})$, meaning that either *(i)* the set is not causally closed; *(ii)* it is not conflict free; or *(iii)* it is a configuration, but not a maximal one. If *(i)* holds, it means an event was removed but not its future which is not possible as for every event in $\overline{E}$, its future is also removed. If we have *(ii)*, a conflict $e_1 \# e_2$ was not resolved which is not possible as $|v'| = |\mathcal{L}|$. Finally, if *(iii)* holds, the configuration can be extended by an event $e'$ from $\lfloor \overline{E} \rfloor$. From the definition of $\overline{E}$, we know that $e'$ is in conflict with the events of $E \backslash \lfloor \overline{E} \rfloor$ and then the extended set cannot be a configuration, leading to a contradiction. □

Since each conflict solver is related to a maximal configuration of the LES, a set of valid conflict solvers corresponds to the solutions of the SAT encoding used to compute configurations of a LES [9] and therefore each LES can be easily extended into the corresponding ELES. This means that both ELESs and CPOGs can be used when one needs to store partial orders in a compressed form and access them by providing the corresponding conflict-solver/opcode. In the rest of the paper we will focus on LESs; however, all presented results also hold for their enriched counterparts.

### 3.2 Conditional Labeled Event Structures

The acyclicity of LESs often introduces 'redundancy' in events, e.g. vertex $a := 5$ from the CPOG in Fig. 4 needs to be represented by four events in the LESs from Fig. 3. In order to avoid this, we follow ideas of CPOGs and label elements of a LES (events and relations) by Boolean conditions in order to represent several LESs with one *Conditional Labeled Event Structure*. Section 5 shows that CLESs are of particular interest when transforming LESs into CPOGs and vice versa.

**Definition 5** A Conditional Labeled Event Structure over alphabet $L$ is a tuple $H = (E, \leq, \#, \lambda, X, \phi, \rho)$ where $E$ are events; $\leq$ is a set of arcs; $\#$ represents conflicts; $\lambda$ labels events; X is a set of operational variables; $\phi$ assigns Boolean conditions to $E, \leq$ and $\#$; and $\rho$ is the restriction function.

A *well-formed* CLES is such that its projection on a valid opcode (allowed by the restriction function) generates a LES, i.e. $\leq$ becomes acyclic. CLESs generalize both CPOGs and LESs: if conflicts are dropped we get a CPOG, and if the structure is acyclic and conditions are dropped, we get a LES.

## 4 Parameterized Structures

The formalisms we presented in the previous sections can be used for the compressed representation of sets of partial orders. The key feature of these formalisms is the support for *conditional elements* labeled with Boolean conditions.

**Definition 6** A mathematical structure over a set of elements $S$ is called a *parameterized structure* if the elements are labeled with Boolean conditions $\phi : S \to \Phi$, where $\Phi$ is a set of predicates (Boolean functions) on $X$, that is $\Phi \subseteq X \to \{0, 1\}$.

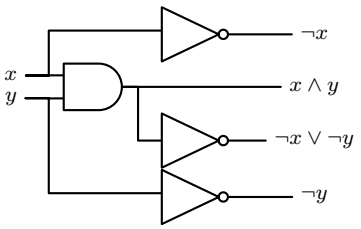


Fig. 7: Circuits computing conditions for the CPOG in Fig. 4.

A CPOG is a parameterized structure whose elements are vertices and arcs. Events and causality/conflict relations are elements of both LESs and CLESs, but every LES element is labeled by 1, while CLES elements can be labeled by arbitrary conditions. Below we define a complexity measure for parameterized structures that can be used to compare compactness of CPOGs, LESs and CLESs.

**Complexity measure.** Instead of treating each predicate in $\Phi$ separately let us construct a *Boolean circuit* [30] that computes all of them together and makes use of shared intermediate terms. This is exactly what happens in practice when a parameterized structure is used for verification purposes (when it is typically converted into a Circuit-SAT instance) or in hardware synthesis (when conditions are evaluated by a circuit comprised of logic gates). The *decoding complexity* of a predicate set $\Phi$ is the number of variables in $\Phi$ plus the number of gates in the smallest circuit[5] computing all predicates.

**Definition 7** The *Complexity* of a parameterized structure with predicate set $\Phi$ on a set of elements $S$ is the decoding complexity of $\Phi$ plus the total number of elements in $S$.

A simple circuit with two negation gates can compute the predicates in Fig. 4. We do not need a circuit to compute conditions of a LES which are always 1. The complexity of the CPOG in Fig. 4 is equal to 13 (2 variables + 2 gates + 5 vertices + 4 arcs) and the complexity of $\mathcal{E}_3$ in Fig. 3 is 22 (10 events + 8 direct causality arcs + 4 direct conflicts). Fig. 7 shows a circuit that computes predicates in $\Phi = \{\neg x, \neg y, x \wedge y, \neg x \vee \neg y\}$ required for the CPOG shown in Fig. 5. If one compares the complexity of the structures in Fig. 5, the LES has complexity 16 (7 events + 6 direct causality arcs + 3 direct conflicts) while the CPOG has complexity 17 (2 variables + 4 gates + 5 vertices + 6 arcs).

**Comparison of Parameterized Structures.** We present three synthetic examples to compare the complexity of the corresponding LESs, CPOGs and CLESs. We observed that a CPOG often has a lower complexity than a corresponding LES, however, the opposite can also be true. Since every CPOG is a CLES with no conflict and every LES is a CLES with $\phi = 1$, CLESs have at most the same complexity as CPOGs and LESs.

[5] In our experiments we restrict the number of inputs of each gate to 2. Since finding the smallest possible circuit is computationally expensive, we use approximation of the circuit complexity measure computed by logic minimisation tools such as ABC [4].

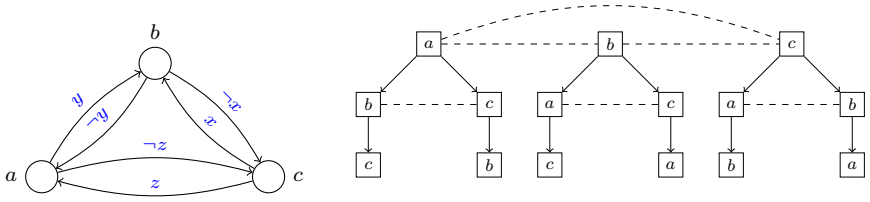


Fig. 8: Phase encoder for $n = 3$ represented by a CPOG (left) and a LES (right).

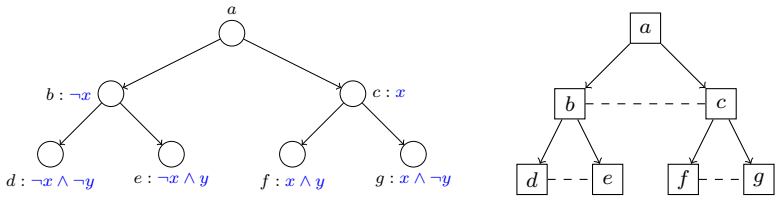


Fig. 9: A decision tree represented by a CPOG (left) and a LES (right).

*Example 1 Phase encoders* [6] are communication controllers capable of generating all permutations of $n$ events. They are very badly handled by acyclic structures as can be seen in Fig. 8 (right). The LES for a phase encoder with $n = 3$ has complexity 33 while the corresponding CPOG has complexity 15. The complexity of CPOGs for phase encoders grows quadratically with $n$ (one literal per arc [16]). One can see that the LES for a phase encoder of size $n$ must have $n!$ events on its lowest level, therefore the complexity of LESs grows exponentially.

*Example 2 Decision trees* [25] are binary trees that can be used to model choices and their consequences. The LESs for encoding decision trees are smaller than the corresponding CPOGs as the number of direct conflicts is smaller than the decoding complexity for conditions needed to encode such decisions. This is illustrated in Fig. 9 where the LES has complexity 16, while the complexity of the CPOG is 21. Asymptotically the complexity of both LESs and CPOGs grows linearly with the size of decision trees, so in this example LESs are better by just a constant factor. In general, as we will demonstrate by an explicit construction algorithm in Section 5, the complexity of a CPOG never exceeds the complexity of the corresponding LES by more than just a constant factor.

*Example 3 Trees of phase encoders* are a combination of decision trees of height $h$ and phase encoders with $n$ actions: after $h$ choices are made, all permutations of $n$ events are possible. For this example, CLESs are strictly smaller than both CPOGs and LESs as demonstrated in Fig. 10 (where $h = 2, n = 2$): the CPOG has complexity 35, the LES has complexity 52, and the CLES has complexity 30.

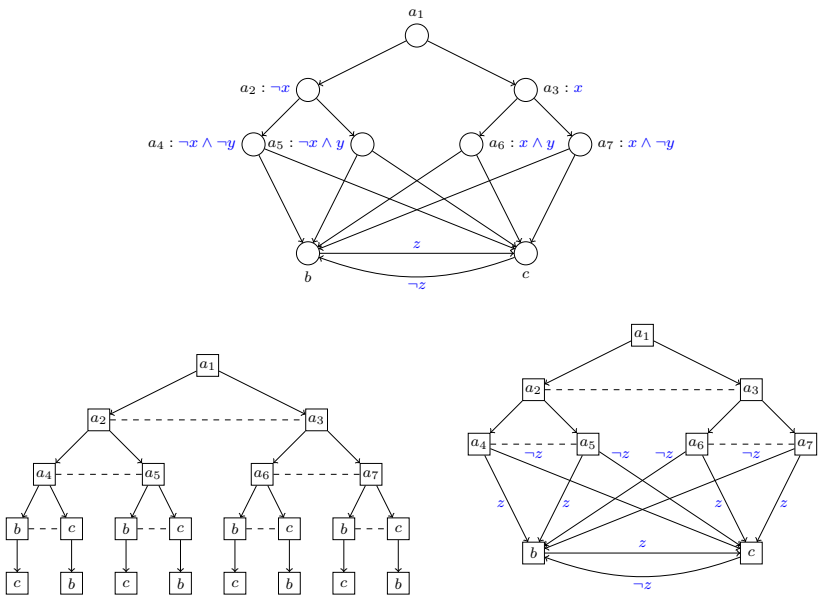


Fig. 10: A tree of phase encoders represented by a CPOG (above), a LES (below left) and a CLESs (below right).

## 5 Transformations

This section presents two algorithms (L2C and C2L) for transforming LESs into CPOGs and vice versa without performing an intermediate uncompression step. Both algorithms make use of CLESs as an intermediate representation. Avoiding such uncompression is essential since the number of represented partial orders may be exponential w.r.t the size of the structure (e.g. see the phase encoders example). Such transformations allow us to study how compact both formalisms are in different application domains, and also allow to reuse synthesis and verification techniques developed for only one of the formalisms.

### 5.1 From LESs to CPOGs (L2C)

Every LES can be seen as an acyclic CLES where vertices and arcs are labeled by 1. If conflicts are removed from the CLES, an acyclic CPOG with redundant vertices (i.e. repeated labels) is obtained which can then be folded to remove such redundancy. In order to preserve the information about conflicts, conflicting events need to be labeled by Boolean conditions in such a way that they cannot belong to the same projection. Proposition 1 shows that whenever an event is selected in one conflict, it must be selected in all other conflicts it participates in, along with all of its causal predecessors. This can be encoded in the restriction function of

the resulting CPOG as follows[6]:

$$\rho = (\bigwedge_{e\#f} \neg\phi_e \vee \neg\phi_f) \wedge (\bigwedge_{e\leq f} \phi_f \Rightarrow \phi_e) \tag{1}$$

For the example shown in Fig. 5 this generates the following restriction function:

$$(\phi_b \Rightarrow \phi_a) \wedge (\phi_e \Rightarrow \phi_a) \wedge (\phi_{c_1} \Rightarrow \phi_b) \wedge (\phi_{d_1} \Rightarrow \phi_b) \wedge$$
$$(\phi_{d_2} \Rightarrow \phi_{c_1}) \wedge (\phi_{c_2} \Rightarrow \phi_{d_1}) \wedge (\neg\phi_e \vee \neg\phi_b) \wedge$$
$$(\neg\phi_{c_1} \vee \neg\phi_{c_2}) \wedge (\neg\phi_{d_1} \vee \neg\phi_{d_2})$$

By employing a SAT solver one can easily check that the above is satisfied by the following assignments which correspond to maximal configurations of the LES:

$$\phi_a = \phi_b = \phi_{c_1} = \phi_{d_1} = 1, \phi_{c_2} = \phi_{d_2} = \phi_e = 0$$
$$\phi_a = \phi_b = \phi_{d_1} = \phi_{c_2} = 1, \phi_{c_1} = \phi_{d_2} = \phi_e = 0$$
$$\phi_a = \phi_b = \phi_{c_1} = \phi_{d_2} = 1, \phi_{c_2} = \phi_{d_1} = \phi_e = 0$$
$$\phi_a = \phi_e = 1, \phi_b = \phi_{c_1} = \phi_{c_2} = \phi_{d_1} = \phi_{d_2} = 0$$

Alas, not only maximal configurations satisfy the function, for example, the empty configuration clearly satisfies it too: $\phi_a = \phi_b = \phi_{c_1} = \phi_{c_2} = \phi_{d_1} = \phi_{d_2} = \phi_e = 0$.

Since we do not want such non-maximal configurations to be allowed by the restriction function, we need to further elaborate it. A configuration is maximal if for every event $e \in E$ one of the following conditions holds: *(i)* event $e$ belongs to the configuration; or *(ii)* there exist an event $f$ which belongs to the configuration and prevents $e$. An event preventing $e$ is called a *spoiler* [11]. Any event $e$ spoils itself as the event cannot occur twice. The set of spoilers of an event $e$ is defined as $spoilers(e) \triangleq \{e\} \cup \{f \in E \mid f\#e\}$. The restriction function (1) can be now refined to allow only maximal configurations:

$$\rho = (\bigwedge_{e\#f} \neg\phi_e \vee \neg\phi_f) \wedge (\bigwedge_{e\leq f} \phi_f \Rightarrow \phi_e) \wedge (\bigwedge_{e\in E} (\phi_e \vee \bigvee_{e\#f} \phi_f)) \tag{2}$$

Coming back to the example in Fig. 5, the additional constraint is:

$$\phi_a \wedge (\phi_b \vee \phi_e) \wedge (\phi_{c_1} \vee \phi_{c_2}) \wedge (\phi_{d_1} \vee \phi_{d_2})$$

The refined restriction function has only four satisfying assignments that represent the four maximal configurations of the LES.

Once conditions are assigned to events, arcs also need to be labeled before folding the result into a CPOG. We label each arc by the conjunction of the conditions of the events it connects to make sure an arc appears only if both of the events do. The resulting CLES may contain several events labeled by the same action, which is redundant for CPOGs. Such events can be merged and the resulting condition is the disjunction of conditions of the original events. See Algorithm 2 for the pseudocode of the folding algorithm L2C.

The set of partial orders represented by the CPOG obtained by the L2C algorithm coincides with the set of maximal configurations of the original LES.

[6] Optimization techniques presented below allow to consider only direct causality and direct conflicts. We make use of this observation in our further examples.

**Algorithm 2** L2C

**Require:** $\mathcal{E} = (E, \leq, \#, \lambda)$ and a set of Boolean variables $\{x_1, \ldots, x_{|E|}\}$
**Ensure:** $H = (V, A, X, \phi, \rho)$ such that $P(H) = \Omega(\mathcal{E})$
1: $V = E, A = \leq$
2: **for** $v \in V$ **do**
3: $\quad \phi_v = x_v$
4: **for** $v_1 \to v_2 \in A$ **do**
5: $\quad \phi_{v_1 \to v_2} = \phi_{v_1} \wedge \phi_{v_2}$
6: **while** $\exists v_1, v_2 \in V : \lambda(v_1) = \lambda(v_2)$ **do**
7: $\quad V = V \backslash \{v_1, v_2\} \cup \{v\}$ for $v \notin V$
8: $\quad$ **for** $v' \in V$ **do**
9: $\quad\quad$ **if** $v' \leq v_1 \vee v' \leq v_2$ **then**
10: $\quad\quad\quad$ add $v' \to v$ to $A$ and set $\phi_{v' \to v} = \phi_{v' \to v_1} \vee \phi_{v' \to v_2}$
11: $\quad\quad$ **if** $v_1 \leq v' \vee v_2 \leq v'$ **then**
12: $\quad\quad\quad$ add $v \to v'$ to $A$ and set $\phi_{v \to v'} = \phi_{v_1 \to v'} \vee \phi_{v_2 \to v'}$
13: $\quad \phi_v = \phi_{v_1} \vee \phi_{v_2}$
14: $\rho = (\bigwedge_{e \# f} \neg\phi_e \vee \neg\phi_f) \wedge (\bigwedge_{e \leq f} \phi_f \Rightarrow \phi_e) \wedge (\bigwedge_{e \in E} \phi_e \vee \bigvee_{e \# f} \phi_f)$
15: **return** $H = (V, A, X, \phi, \rho)$

**Theorem 2** *Given a LES $\mathcal{E} = (E, \leq, \#, \lambda)$ and a set of Boolean variables, algorithm* L2C *constructs a CPOG $H = (V, A, X, \phi, \rho)$ such that $P(H) = \pi(\Omega(\mathcal{E}))$.*

*Proof* The first steps of the algorithm (lines 1-5) add conditions to events and arcs, i.e. they transform the LES into a CLES. We proceed by showing that *i)* all the projections of such a CLES over every valid opcode coincide with its maximal configurations; and *ii)* $P(H) = P(H')$ where $H$ is the CLES before merging events (lines 6-13) and $H'$ the CLES obtained by merging them.

*i)* Let $H$ be the CLES obtained after steps 1-5; $\psi$ is a valid opcode

$$\begin{aligned}
&\Leftrightarrow \psi \models (\bigwedge_{e \# f} \neg\phi_e \vee \neg\phi_f) \wedge (\bigwedge_{e \leq f} \phi_f \Rightarrow \phi_e) \wedge (\bigwedge_{e \in E} \phi_e \vee \bigvee_{e \# f} \phi_f) \\
&\Leftrightarrow H_{|\psi} \text{ is conflict free, causally closed and maximal} \\
&\Leftrightarrow H_{|\psi} \text{ form a maximal configuration of the CLES} \\
&\Leftrightarrow H_{|\psi} \text{ form a maximal configuration of the LES}
\end{aligned}$$

*ii)* Let $e_1, e_2$ be the events of $H$ which are replaced by $e$ in $H'$. Since $\phi_e = \phi_{e_1} \vee \phi_{e_2}$, for every valid opcode $\psi$ such that $e \in H_{|\psi}$ we have $e_1 \in H'_{|\psi}$ or $e_2 \in H'_{|\psi}$, but since we are only interested in the label of the events in the partial order and $\lambda(e) = \lambda(e_1) = \lambda(e_2)$, we can conclude $\lambda(e) \in {H'}_{|\psi}$ iff $\lambda(e_1) \in H_{|\psi}$ ; for every $v \in V$ we have $v \leq e_1$ or $v \leq e_2$ iff $v \leq e$ (analogously for $e_1 \leq v$ or $e_2 \leq v$). Since labels and causal dependences are preserved, we can conclude that $P(H) = P(H')$.

□

The complexity of the CPOG constructed by L2C is linear w.r.t. the size of the original LES.

**Theorem 3** *Given a LES $\mathcal{E} = (E, \leq, \#, \lambda)$, algorithm* L2C *constructs a CPOG $H = (V, A, X, \phi, \rho)$ of complexity $\Theta(|\mathcal{E}|)$.*

*Proof* By Definition 7 we need to consider the number of vertices, arcs and the decoding complexity of $\phi$ and $\rho$. Clearly $|V| \leq |E|$ as some events are merged. For every event $e$ obtained by merging two events $e_1, e_2$ and any other event $f$, we have $e \rightarrow f \Leftrightarrow e_1 \leq f \vee e_2 \leq f$, therefore we can conclude $|A| \leq |\leq|$. We label each event with a different variable ($|X| = |E|$), hence the decoding circuit is trivial. The condition of each arc is the conjunction of the conditions of the events it connects, which requires $|\leq|$ AND gates to compute $\phi$. To compute $\rho$ as explained in (2), a NAND gate is needed for each conflict to assure that only one event is selected; an implication (a NOT and an OR gate) for each arc in $\leq$; each conflict is listed exactly two times in the maximality encoding (once for each event in the conflict), thus we need $2 * |\#| + |E|$ gates to compute it. Finally, we need $|\leq| + |\#| + |E| - 1$ AND gates to join all constraints together. The overall size of the $\rho$ function is therefore $4 * |\#| + 3 * |\leq| + 2 * |E| - 1 \leq 4 * |\mathcal{E}|$. To conclude, the complexity of the resulting CPOG is $\Theta(|\mathcal{E}|)$. □

Algorithm L2C is optimal w.r.t. the size of the underlying graph.

**Theorem 4** *Given a LES, algorithm* L2C *constructs a CPOG* $H = (V, A, \_, \_, \_)$ *such that no smaller graph* $H' = (V', A', \_, \_, \_)$ *with* $|V'| < |V|$ *or* $|A'| < |A|$ *can represent the same set of partial orders.*

*Proof* The **while** statement, line 6 of Algorithm 2, terminates only after every pair of equally labeled nodes have been merged. Therefore, there is exactly one vertex in the resulting $V$ for every possible LES label, and any $V'$ such that $|V'| < |V|$ cannot represent the same set of partial orders as the original LES. We thereby assume that $V = V'$ and $|A'| < |A|$. Under this assumption there exist two vertices $a, b \in V$ such that $a \rightarrow b \in A$, but $a \rightarrow b \notin A'$. Any partial order represented by $H'$ does not relate labels $a$ and $b$ because the edge does not belong to $A'$. Since this edge belongs to $A$, there exists at least one partial order (obtained by any projection where $\phi_{a \rightarrow b} = 1$) represented by $H$ where $a$ and $b$ are related, which shows that $H$ and $H'$ cannot represent the same set of partial orders. □

Notice that even if the transformation algorithm is optimal w.r.t the the graph size, it might not be optimal w.r.t its complexity which also considers the complexity of Boolean predicates. Finding an optimal CPOG encoding is still an open research question with ongoing research [17]. Our algorithm uses the simplest encoding approach where each event is initially labeled by a unique variable.

**Optimization techniques.** Below we describe several important optimization techniques that improve L2C.

1. *Arc reduction:* the proof of Theorem 4 uses $|\leq|$ AND gates to compute $\phi$. This is because each arc $e \leq f$ is labeled by $\phi_e \wedge \phi_f$. However, as we know from (2), $\phi_f \Rightarrow \phi_e$ and therefore the arc conditions can be simplified to just $\phi_f$ and no gates are needed.
2. *Transitive reduction:* the relation $\leq$ is causally closed, that is, it contains transitive arcs $a \leq c$ whenever $a \leq b$ and $b \leq c$. The clauses corresponding to transitive arc $(\phi_c \Rightarrow \phi_a)$ are clearly redundant in presence of $(\phi_c \Rightarrow \phi_b)(\phi_b \Rightarrow \phi_a)$ and can be dropped. Therefore, in the transformation algorithm we can use the transitively reduced relation $\leq_r$ instead of $\leq$.

3. *Conflict inheritance reduction:* consider two events $a$ and $b$ in direct conflict and two events in their future, i.e. $a \leq c$ and $b \leq d$. Clearly $c\#d$, but this conflict does not need to be encoded by $\neg\phi_c \vee \neg\phi_d$. If the conflict $a\#b$ and both $a \leq c, b \leq d$ are encoded, we have $\phi_a \Rightarrow \neg\phi_b, \phi_c \Rightarrow \phi_a$ and $\phi_d \Rightarrow \phi_b$, thus $\phi_c \Rightarrow \phi_a \Rightarrow \neg\phi_b \Rightarrow \neg\phi_d$ which prevents to select both $c$ and $d$. Therefore, we only need to consider $\#_r$.
4. *Spoilers reduction:* consider the example from Fig. 6 and the spoiler sets of events $c_1$ and $c_2$. These sets generate the clauses $\Phi_1 = \phi_{c_1} \wedge \phi_{c_2} \wedge \phi_e$ and $\Phi_2 = \phi_{c_2} \wedge \phi_{c_1} \wedge \phi_{d_2} \wedge \phi_e$. Clearly $\Phi_1 \Rightarrow \Phi_2$ and the information about the spoilers of $c_2$ is redundant. For every pair of events $e, f$ such that $spoilers(e) \subseteq spoilers(f)$, the constraints generated for $f$ are redundant and can be dropped.
5. *Multiway conflicts:* notice that if several events $\{e_1, \cdots, e_k\}$ are pairwise in conflict (i.e., the conflict subgraph induced by them is a clique), Algorithm 2 will generate a quadratic number of constraints $\bigwedge_{e_i \# e_j} (\neg\phi_{e_i} \vee \neg\phi_{e_j})$. This can be simplified into $(\sum_i e_i \leq 1)$ which can be encoded by a set of constraints of linear size [8].

### 5.2 From CPOGs to LESs (C2L)

To transform a CPOG into a LES, the former needs to be unfolded (in order to obtain an acyclic structure) while keeping conditions that will be replaced by conflicts in the final LES. For this, a CLES is constructed as an intermediate structure. We start from an empty CLES (i.e. one with $E = \emptyset$) and at each iteration, we compute the set of possible extensions. To decide if an instance of vertex $a \in V$ is a possible extension, we need to find a set of predecessor events $P \subseteq E$ such that *(i)* the vertex is active; *(ii)* instances of its predecessors and their corresponding arcs are active; *(iii)* if an event is not a predecessor, then either it is not active or its corresponding arc is not active; *(iv)* the instance of the vertex is different to any other in the prefix. This is captured by the formula (3) displayed below where each conjunction corresponds to one of the points mentioned above. For each vertex $a \in V$ we have:

$$\phi = \phi_a \wedge \Big( \bigwedge_{\substack{e_b \in P \\ b \to a \in A}} \phi_{e_b} \wedge \phi_{b \to a} \Big) \wedge \Big( \bigwedge_{\substack{e_b \in E \setminus P \\ b \to a \in A}} \neg\phi_{e_b} \vee \neg\phi_{b \to a} \Big) \wedge \Big( \bigwedge_{e_a \in E} \neg\phi_{e_a} \Big) \tag{3}$$

If the formula is unsatisfiable, then there exists no possible extension representing vertex $a$. If the formula is satisfiable, we add the event to the unfolding, appropriately connecting it to $P$ and labeling by $\phi$. The unfolding procedure is finished when (3) is no longer satisfiable; this is always the case since the CPOG is well-formed and thus any cycle comprises arcs with mutually exclusive conditions and the predicate $\phi_{b \to a}$ in the second conjunction eventually makes the formula unsatisfiable.

*Remark 2* In the resulting unfolding an event $e$ is labeled by the conjunction of the conditions of the vertex it represents and the conditions of vertices and arcs on the path from a minimal vertex to $e$. For example, if a CPOG contains $a, b \in V, a \to b \in A$ with $\phi_a = x, \phi_b = y$ and $\phi_{a \to b} = z$, then event $e_b$ is labeled by $x \wedge y \wedge z$.

**Proposition 2** *Given a CPOG and a prefix of its unfolding, deciding whether an instance of a vertex is a possible extension is NP-hard.*

*Proof* Consider a CPOG containing a single vertex $v$ with condition $\phi_v$. As explained by Remark 1, there exists an event $\top$ such that $v \to \top$ and $\psi_\top = 1$. We need to be able to decide if $\phi_v$ is equal to 0 or 1. If $\phi_v = 0$, then the unfolding should only contain $\top$; if $\phi_v = 1$, the unfolding should contain $v$ and $\top$ with $v \leq \top$; otherwise, it has three events $v, \top_1, \top_2$ with $v \leq \top_1$ and $v \# \top_2$ (that is, $v$ either happens or not). Deciding if $\phi_v$ is equal to 0 or 1 is an NP-hard problem. □

We use a SAT-solver to 'guess' a combination of a predecessor set $P$ satisfying (3). Finally, Boolean conditions are replaced by conflicts: for every pair of mutually exclusive events $e_a, e_b$, their Boolean conditions are removed and conflict $e_a \# e_b$ is added instead. As the set $\mathcal{V}$ does not depend on the graph, it can be constructed from the LES itself, while $\mathcal{L}$ is obtained as a linearization of $\#_r$. We refer to the unfolding procedure followed by adding the conflict relations as the C2L algorithm.

The final LES resulting from the unfolding does not depend on the order in which events are added, i.e. C2L is deterministic.

**Proposition 3** *Let $E$ be the current set of events of the unfolding and $e_a \neq e_b$ two possible extensions, then $e_b$ is a possible extension of $E \cup \{e_a\}$.*

*Proof* We need to prove that (3) is still satisfiable for vertex $b$ when $e_a$ is added to the unfolding; we focus in each of the four conjuncts of the formula: *i)* as $e_b$ is a possible extension from $E$, $\phi_b = 1$ and this is also true from $E \cup \{e_a\}$; *ii-iii)* since $P \subseteq E \subseteq E \cup \{e\}$ the second and third conjunction of (3) are still satisfied; *iv)* since $e_a \neq e_b$ and there was not an instance of $b$ in $E$, there is neither in $E \cup \{e_a\}$. □

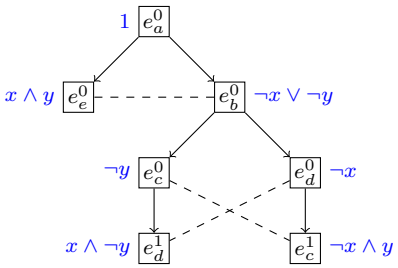


Fig. 11: Transformation of a CPOG into a LES.

*Example 4* Consider the CPOG shown in Fig. 5. The unfolding procedure starts with $E = \emptyset$ and keeps checking vertices of the CPOG for possible extensions (see Fig. 11). At start, only vertex $a$ can be added. For example, the constraint imposed by non-predecessors in (3) will include $\neg\phi_{a \to b} = \neg 1 = 0$ for vertex $b$, hence it is not a possible extension at start. We proceed by adding event $e_a^0$ to the unfolding with $\phi_{e_a^0} = 1$. When we recompute the possible extensions, formula (3) reduces to $\neg x \vee \neg y$ and $x \wedge y$ for vertices $b$ and $e$, respectively, therefore events $e_b^0$ and $e_e^0$ are added with $e_a^0$ as their predecessor and with $\phi_{e_b^0} = \neg x \vee \neg y$ and $\phi_{e_e^0} = x \wedge y$.

At this point $E = \{e_a^0, e_b^0, e_e^0\}$ and we find that $c$ and $d$ are possible extensions adding events $e_c^0$ and $e_d^0$ with event $e_b^0$ as the predecessor and conditions

$\phi_{e_c^0} = \neg y$ and $\phi_{e_d^0} = \neg x$. Now $E = \{e_a^0, e_b^0, e_c^0, e_d^0, e_e^0\}$ and we find that $c$ and $d$ are possible extensions again. Two new events $e_c^1$ and $e_d^1$ are added. Finally, as $E$ grows to $\{e_a^0, e_b^0, e_c^0, e_c^1, e_d^0, e_d^1, e_e^0\}$, formula (3) becomes unsatisfiable and the unfolding procedure is finished. Conditions of events $e_b^0$ and $e_e^0$ are mutually exclusive: $(x \wedge y) \wedge (\neg x \vee \neg y) = 0$, therefore we add conflict $e_b^0 \# e_e^0$. Conflicts $e_c^0 \# e_c^1$ and $e_d^0 \# e_d^1$ are added following the same reasoning. Finally, when all Boolean conditions are removed from the CLES, we obtain the resulting LES.

As one can see, C2L is significantly more computationally intensive than L2C: unravelling CPOGs requires the use of a SAT solver. Fortunately, the SAT instances that need to be solved are similar to each other, therefore one can use incremental SAT solving techniques [31] to speed up the algorithm.

Below we show that C2L is correct, i.e. it preserves the set of partial orders.

**Theorem 5** *Let $H = (V, A, X, \phi, \rho)$ be a well-formed CPOG and $\mathcal{E} = (E, \leq, \#, \lambda)$ the LES obtained by* C2L*, then $\pi(\Omega(\mathcal{E})) = P(H)$.*

*Proof* We first show that the intermediate CLES preserves the set of partial orders and then that Boolean conditions can be safely replaced by conflicts.

*i)* Let $H$ be a well formed CPOG, $G$ the CLES obtained by the unfolding procedure before conditions are removed and $\psi$ a valid opcode. We prove that the actions and dependences of $H_{|\psi}$ and $G_{|\psi}$ coincide:

$\Leftarrow$) If an event $e_v$ belongs to $G_{|\psi}$ then $\psi \models e_v$ and since (3) implies $\phi_{e_v} \Rightarrow \phi_v$, we know that $\psi \models \phi_v$ and $v$ belongs to the events of $H_{|\psi}$. If in addition $e_{v'} \leq e_v$ is part of the causality of $G_{|_\psi}$, using again (3) we have $\psi \models \phi_{v'} \wedge \phi_{v' \to v}$ and then both the vertex $v'$ and the edge $v' \to v$ are part of $H_{|\psi}$. Suppose there are vertices of $H_{|\psi}$ that do not have a corresponding event in $G_{|\psi}$. If such vertices exist, we can always find one (lets call it $v$) such that there exist instances of its predecessors (which can be empty if it has no predecessors) in the CLES and the corresponding arcs are active. Then event $e_v$ does not belong to the CLES if there is another instance of vertex $v$ in $G_{|\psi}$ which leads to a contradiction.

$\Rightarrow$) Suppose there are vertices of $H_{|\psi}$ that are not represented by an event in $G_{|\psi}$ and consider a minimal one w.r.t $\rightarrow$ ($H_{|\psi}$ is acyclic). By Remark 2 we know that if the vertex is not there then the conjunction of the vertices and edges in the path of $H_{|\psi}$ is unsatisfiable which leads to a contradiction since $\psi$ is a valid assignment. Clearly there is no events in the CLES that are not instances of a vertex and (3) imposes that there are not two instances of a vertex in $H_{|\psi}$. Since the events of $H_{|\psi}$ and $G_{|\psi}$ coincide and the unfolding algorithm appropriately connects the events with its set of predecessors, causality imposed by $\rightarrow$ is preserved by $\leq$.

*ii)* Let $G$ be the CLES defined above and $\mathcal{E}$ be the LES obtained when Boolean conditions are replaced by conflicts. We show that the projections of $G$ coincide with the maximal configurations of $\mathcal{E}$. Two events $e_1, e_2$ belong to some projection if there exists some $\psi$ such that $\psi \models \psi_{e_1} \wedge \psi_{e_1}$, but then $e_1$ and $e_2$ are not mutually exclusive and therefore they are not in conflict in $\mathcal{E}$. Since two events belong to a maximal configuration if they are not in conflict, $e_1, e_2$ belong to the same set of maximal configurations.

□

The LES constructed by C2L can have at most exponential size w.r.t. to the size of the original CPOG (to fit an exponential number of partial orders), and it is optimal: no smaller LES can represent the same set of partial orders.

**Theorem 6** *The algorithm* C2L *constructs a LES* $\mathcal{E} = (E, \_, \_, \_)$ *such that no LES* $\mathcal{E}' = (E', \_, \_, \_)$ *with* $|E'| < |E|$ *can represent the same set of partial orders.*

*Proof* Let $\mathcal{E} = (E, \_, \_, \_)$ and $\mathcal{E}' = (E', \_, \_, \_)$ be two LES representing the same set of partial orders with $|E'| < |E|$. Since both LESs represent the same partial orders, there must be two maximal configurations of $\mathcal{E}$ representing the same partial order as a single configuration of $\mathcal{E}'$. Let $e_1, e_2$ be two conflicting events representing the same label in the two configurations of $\mathcal{E}$; it follows that the predecessors of $e_1, e_2$ are equally labeled. Let $e'_1, e'_2$ be the $\leq$-minimal such events, then they are in direct conflict. Call $P$ the predecessors of $e'_1$ and $P'$ the predecessors of $e'_2$; it follows that $P$ and $P'$ are not in conflict, if not the conflict between $e'_1, e'_2$ would not be direct. This implies that the predicate conditions of the events in $P$ and $P'$ before transforming conditions into conflicts were not mutually exclusive. Since the predicates of $\phi_{e'_1}$ and $\phi_{e'_2}$ only differ on the parts $\bigwedge_{e_b \in P} \phi_{e_b}$ and $\bigwedge_{e'_b \in P'} \phi_{e'_b}$, it follows that $\phi_{e'_1}$ and $\phi_{e'_2}$ cannot be mutually exclusive and then $e'_1, e'_2$ cannot be in conflict, leading to a contradiction. □

### 5.3 Composition of algorithms L2C and C2L

In programming languages (particularly in functional ones), *fusion* or *deforestation* is a technique that avoids constructing a data structure if the structure will eventually be consumed in a subsequent computation. A *hylomorphism* is a transformation formed by an unfolding followed by a fold function; its counterpart, a *metamorphism*, first folds and then unfolds. In this section we study the composition of the presented transformation algorithms L2C and C2L that, in essence, perform folding and unfolding, respectively. We show that the unfolding algorithm reverts the effects of the folding one, i.e. the initial and resulting LESs after applying both algorithms are the same. However, applying the transformations in the opposite order does not guarantee to obtain the initial CPOG since Boolean conditions are reintroduced in a naive way, by using one variable per event. However, the structure of the underlying graph is still preserved. These notions are formalized by the following results.

**Theorem 7** *Consider* L2C *and* C2L *as functions between parametrized structures, then* L2C $\circ$ C2L $=$ ID.

*Proof* The proof is by induction on the number of events of $\mathcal{E}$.

**Base case:** $n = 0$; this case is trivial since $E = \emptyset$ implies $V = \emptyset$ and the unfolding algorithm returns the empty LES.

**Inductive case:** assume the result holds for a LES with at most $n$ events. Let $e$ be any 'maximal' event of $\mathcal{E}$ (one that precedes a $\top$ event) and denote as $\mathcal{E}_{\setminus e}$ the LES resulting by removing $e$. Clearly $\mathcal{E}_{\setminus e}$ satisfies the inductive hypothesis and $\text{L2C}(\mathcal{E}_{\setminus e}) = H_{\setminus e}$ and $\text{C2L}(H_{\setminus e}) = \mathcal{E}_{\setminus e}$ where $H_{\setminus e}$ is the CPOG constructed by Algorithm 2. Let $H$ be the CPOG obtained by folding $\mathcal{E}$, i.e. $\text{L2C}(\mathcal{E}) = H$,

we will show that $\mathcal{E}_{\setminus e}$ has a possible extension representing $e$ and computed from $H$. Notice that $H$ and $H_{\setminus e}$ may differ only on a vertex (if $e$ is the only event labeled by $\lambda(e)$) or one of its arcs.
Assume $\lambda(e) = a$ and this label is different from any other label in $H_{\setminus e}$ (i.e. the folding algorithm will not merge $e$ with any other vertex). Call $P$ the predecessor of $e$ in $\mathcal{E}$; every event in $P$ belongs to $\mathcal{E}_{\setminus e}$. We need to show that there exists a satisfying assignment of (3). We use individual variables as conditions for each event, hence $\phi_e$ cannot be mutually exclusive with any other predicate; also since there is no other event labeled with the same action as $e$ the last conjunction of (3) is trivially satisfiable. There cannot be conflicts between events in $P$ and $e$ (since they were its predecessors) and thus the restriction function $\rho$ of Algorithm 2 does not impose that their conditions are mutually exclusive. Any event $e_b \notin P$ and labeled by $b$ was either in conflict with $e$ (and by the restriction function $\neg\phi_{e_b}$ holds) or concurrent with it, meaning that the there was no edge between $b$ and $a$ and thus $\neg\phi_{b\to a}$. We can conclude that (3) has a satisfiable assignment and $e$ can be added to $\mathcal{E}_{\setminus e}$.
If $e$ shares the same label with another vertex in $H_{\setminus e}$, all the above also holds except the fact that the last conjunction is trivially satisfiable. Since there are other events with the same label, we need to prove that the instance of the vertex is different to any other in the prefix. Let $e_a$ be another event with the same label, then it holds that $e \# e_a$ since two equally labeled events cannot belong to the same configuration (if not the partial order it represents would have two occurrences of label $a$) and then $\phi_e$ and $\phi_{e_a}$ are mutually exclusive by the restriction function. Observe that the path from a minimal event (one with no predecessors) to any event has been labeled by the conjunction of the predicates of the events in the path. Thus the paths to $e$ and to $e_a$ are also labeled by mutually exclusive predicates. Since the predicate assigned to any possible extension consider the entering arcs (see the second conjunction of (3)) it can be concluded that $\neg\phi_{e_a}$ holds and the instance of the vertex corresponding to $e$ is different to any other in $\mathcal{E}_{\setminus e}$.

□

Applying L2C to the result produced by C2L preserves the structure of the underlying graph, but in general does not preserve the conditions.

**Theorem 8** *Let $H = (V, A, \_, \_, \_)$ and $H' = (V', A', \_, \_, \_)$ be two CPOGs such that $H$ does not contain vertices or arcs with False condition, $\text{C2L}(H) = \mathcal{E}$ and $\text{L2C}(\mathcal{E}) = H'$, then $V = V'$ and $A = A'$.*

*Proof* Since a vertex is uniquely identify by its label, to prove that $V = V'$ we only need to show that the unfolding algorithm adds to the LES at least one event for each label (assuming no vertex has False condition – such redundant events are not added to the LES); since C2L folds every pair of equally labeled events, the preservation on folding step is trivial. We have already proved in Theorem 5 that the partial orders represented by the LES and the CPOG are the same, therefore for each label there must be at least one event representing it and we can conclude $V = V'$. For proving the preservation of arcs, assume there exists some arc $b \to a \in A$ such that $b \to a \notin A'$ and let $\psi$ be any opcode reducing $\phi_{b\to a}$ to True, i.e. the arc is part of the partial order represented by the opcode $\psi$ (such an opcode must exist since $\phi_{b\to a}$ cannot be False). Since the unfolding algorithm is guaranteed to preserve the represented partial orders (see Theorem 5), there

must exist some configuration representing the same partial order than $H_\psi$ (which includes arc $b \to a$) and thus we can assume the existence of events $e_b, e_a$ with $e_b \leq e_a$ and $\lambda(e_a) = a, \lambda(e_b) = b$. The merging algorithm will transform those events into vertices $b$ and $a$ and since they are related according to $\leq$, the edge $b \to a$ will be added to $A'$ by line 10 or line 12 in Algorithm 2, contradicting our initial assumption. □

**Corollary 1** *The transformation algorithms* C2L *and* L2C *are adjoint functors, that is:* C2L ∘ L2C ∘ C2L = C2L *and* L2C ∘ C2L ∘ L2C = L2C.

*Proof* Immediate from Theorem 7 and Theorem 8. □

## 6 Experiments

In this section we compare the complexity of LESs and CPOGs on a number of benchmarks. Besides the synthetic examples introduced in Section 4, we use benchmarks coming from Java and C code for multithreaded programs [12,26] (FileSystem, ParallelPi, Synth, Ssb, Stf and Ccnf) and the VLSI design domain, in particular, on-chip communication controllers [6] and processor micro-architectures [18] (ARM Cortex M0 and Intel 8051). The processor benchmarks come from the instruction sets available in [2] and [27], respectively.

| Benchmark | \|POs\| | CPOG | LES | LPO |
|---|---|---|---|---|
| PhaseEnc | 24 | **24** | 158 | 168 |
| | 120 | **35** | 825 | 1080 |
| | 720 | **48** | 5001 | 7920 |
| DecTree | 8 | 44 | **36** | 56 |
| | 16 | 97 | **76** | 144 |
| | 32 | 195 | **156** | 352 |
| TreePhaseEnc | 16 | **70** | 92 | 176 |
| | 24 | **43** | 160 | 264 |
| | 48 | **136** | 324 | 624 |
| ARM Cortex M0 | 9 | **28** | 46 | 60 |
| | 10 | **29** | 48 | 64 |
| | 11 | **30** | 53 | 67 |
| Intel 8051 | 9 | **46** | 99 | 138 |
| | 10 | **47** | 123 | 158 |
| | 11 | **51** | 144 | 176 |
| FileSystem | 2 | 2952 | **457** | - |
| | 8 | 4188 | **1065** | - |
| | 32 | 5520 | **1485** | - |
| ParallelPi | 720 | 3273 | **2129** | - |
| | 5040 | **15769** | 26678 | - |
| Synth | 1316 | **43961** | 54589 | - |
| Ssb | 4 | 3201 | **3180** | - |
| Stf | 6 | **16064** | 21003 | - |
| Ccnf | 1024 | 40 | **30** | 10240 |
| | 2048 | 44 | **33** | 22528 |
| | 4096 | 48 | **36** | 49152 |

Table 1: Experimental results on the complexity of CPOGs and LESs.

Table 1 provides our experimental comparison[7]. Column '| POs |' reports the number of partial orders in the benchmark; we additionally report on the complexity of both CPOGs and LESs. Where the scenarios were known a priori, we also report the size of their uncompressed representation as a list of partial orders (column 'LPO'). For the phase encoders, decision trees, trees of phase encoders and processor instructions (i.e. benchmarks where the scenarios were known a priori), the CPOGs where synthesized using the synthesis procedure introduced in [17] and the LESs were constructed using the synthesis and optimization introduced in Section 2.1. The implementation of the latter can be found at `https://github.com/hernanponcedeleon/Partial-Orders-Bridges`. For the multithreaded programs benchmarks, the inputs were programs written in Java or C code. The LESs were extracted from the programs using the multithreaded testing approaches from [12] and [26], and the CPOGs were obtained by transforming the LESs using Algorithm L2C. The transformation algorithm L2C is available at `https://github.com/tuura/les-to-cpog`.

The table shows that CPOGs handle permutations of activities very well (PhaseEnc and TreePhaseEnc benchmarks), while LESs can represent branching (DecTree example) with lower complexity. This is coherent with the results of Section 4 where the examples were introduced. For processors instructions, the compactness obtained by CPOGs is better than the one obtained by LESs. This is not surprising since CPOGs were designed for compact representation of microprocessor instructions. For the benchmarks coming from multithreaded programs, the results are more balanced since the CPOG was obtained by the transformation algorithm and not using a synthesis algorithm (since the scenarios were not known a priori); this is consistent with Theorem 4 guaranteeing that the size of the CPOG obtained using our translation procedure is linear w.r.t the LES.

For the cases where LESs have higher complexity, this is due to an explosion in the number of events or causalities (the LESs have up to 100× more events or dependencies than the corresponding numbers of CPOG vertices or arcs). For the multithreaded programs, since the CPOGs were computed by the transformation algorithm introduced in this article, we believe that better reductions could be achieved if the algorithm for direct synthesis of CPOGs was applied instead. This is due to the fact that the transformation algorithm may introduce redundant literals in the computation of the $\rho$ function. A possible solution to this is suggested by an observation below.

Original Java benchmarks were first transformed into Petri Net unfoldings (i.e. occurrence nets) using the approach from [12] and then converted into isomorphic LESs. The LESs obtained from C code by the approach from [26] could also be transformed into saturated occurrence nets [21] by adding one condition between each causal pair of events and one condition consumed by each conflicting pair. We measured the complexity of the resulting occurrence nets (number of events, conditions and the size of the flow relation) and in several examples they provided a better compression than the corresponding CPOGs and LESs. This occurred in particular in examples where LESs had lower complexity than CPOGs and the number of conditions in the corresponding occurrence nets was smaller than the

[7] CLESs are used as an intermediate representation by the translation algorithms. Obtaining CLESs of minimal size was not one of our goals and therefore we do not include them in the experiments.

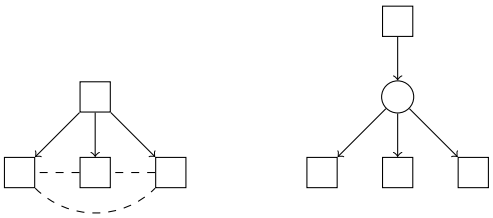

Fig. 12: A LES with complexity 10 and the corresponding occurrence net from [24] with complexity 9.

number of dependencies and conflicts (see for example Fig. 12). Further reductions can be still obtained if one uses the conversion of [24] instead of the saturated occurrence net mentioned above. We conjecture that a direct conversion algorithm from Petri Net unfoldings or occurrence nets to CPOGs, which avoids the intermediate LES construction, would be able to achieve a better compression. This is a topic of our future work.

Finally, it can be concluded that for the cases where the number of scenarios is exponential w.r.t one of the formalisms (PhaseEnc or Ccnf) the use of the transformation algorithms presented in the article avoids the memory allocation for all the partial orders in an uncompressed form.

## 7 Conclusion

The paper discusses the use of two models (LESs and CPOGs) for a compact representation of sets of partial orders. We show that LESs work well on practical examples coming from multithreaded programs, however, due to their acyclic nature they cannot efficiently handle the cases where sets of partial orders contain many permutations defined on the same set of events. These cases are very well handled by CPOGs, however, the use of Boolean conditions for resolving conflicts makes them less intuitive and more demanding from the algorithmic complexity point of view, in particular, most interesting questions about CPOGs are NP-hard. The advantages of both models are combined by CLESs which are used as an intermediate formalism by the presented algorithms, which can transform a set of partial orders from a given compressed representation in a LES or a CPOG into an equivalent compressed representation in the other formalism without the explicit enumeration of all partial orders.

Our future work includes further optimization of the presented algorithms, their integration with Workcraft modeling and verification framework [1, 22], and the validation on larger case studies. As mentioned above, direct transformations from occurrence nets and merged processes into CPOGs will be studied.

**Acknowledgements** We would like to thank the anonymous reviewers for their suggestions and comments which have allowed us to improve our manuscript. This work was partially supported by EPSRC research grants A4A (EP/L025507/1) and POETS (EP/N031768/1).

## References


1. Workcraft webpage (2017). `www.workcraft.org`
2. ARM Ltd.: ARMv6-M Architecture Reference Manual. (2010)
3. van Beest, N.R.T.P., Dumas, M., García-Bañuelos, L., Rosa, M.L.: Log delta analysis: Interpretable differencing of business process event logs. In: Business Process Management - 13th International Conference, BPM 2015, Innsbruck, Austria, August 31 - September 3, 2015, Proceedings, pp. 386–405 (2015)
4. Berkeley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification, Release 70930. `http://www.eecs.berkeley.edu/~alanmi/abc/`
5. Brookes, S., O'Hearn, P.W.: Concurrent separation logic. ACM SIGLOG News **3**(3), 47–65 (2016)
6. D'Alessandro, C., Mokhov, A., Bystrov, A.V., Yakovlev, A.: Delay/phase regeneration circuits. In: 13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC 2007), 12-14 March 2006, Berkeley, California, USA, pp. 105–116. IEEE Computer Society (2007). DOI 10.1109/ASYNC.2007.14
7. Diekert, V., Rozenberg, G. (eds.): The Book of Traces. World Scientific Publishing Co., Inc. (1995)
8. Esparza, J.: Decidability and complexity of petri net problem – an introduction. In: Lectures on Petri Nets I: Basic Models, pp. 374–428. Springer (1998)
9. Esparza, J., Heljanko, K.: Unfoldings - A Partial-Order Approach to Model Checking. Monographs in Theoretical Computer Science. An EATCS Series. Springer (2008)
10. Godefroid, P., Van Leeuwen, J., Hartmanis, J., Goos, G., Wolper, P.: Partial-order methods for the verification of concurrent systems: an approach to the state-explosion problem, vol. 1032. Springer Heidelberg (1996)
11. Haar, S., Rodríguez, C., Schwoon, S.: Reveal your faults: It's only fair! In: ACSD, pp. 120–129 (2013)
12. Kähkönen, K., Heljanko, K.: Testing multithreaded programs with contextual unfoldings and dynamic symbolic execution. In: 14th International Conference on Application of Concurrency to System Design, pp. 142–151 (2014). DOI 10.1109/ACSD.2014.20
13. Khomenko, V., Mokhov, A.: An algorithm for direct construction of complete merged processes. In: International Conference on Application and Theory of Petri Nets and Concurrency, pp. 89–108. Springer (2011)
14. McMillan, K.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: Computer Aided Verification, pp. 164–177. Springer (1993)
15. Milner, R.: A calculus of communicating systems, vol. 92. Springer Verlag Berlin (1980)
16. Mokhov, A.: Conditional partial order graphs. Ph.D. thesis, Newcastle University (2009)
17. Mokhov, A., Alekseyev, A., Yakovlev, A.: Encoding of processor instruction sets with explicit concurrency control. IET computers & digital techniques **5**(6), 427–439 (2011)
18. Mokhov, A., Iliasov, A., Sokolov, D., Rykunov, M., Yakovlev, A., Romanovsky, A.: Synthesis of processor instruction sets from high-level ISA specifications. IEEE Trans. Computers **63**(6), 1552–1566 (2014). DOI 10.1109/TC.2013.37
19. Mokhov, A., Khomenko, V.: Algebra of parameterised graphs. ACM Transactions on Embedded Computing Systems **13**(4s) (2014)
20. Mokhov, A., Yakovlev, A.: Conditional partial order graphs: Model, synthesis, and application. IEEE Trans. Computers **59**(11), 1480–1493 (2010)
21. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13**, 85–108 (1981)
22. Poliakov, I., Sokolov, D., Mokhov, A.: Workcraft: a static data flow structure editing, visualisation and analysis tool. In: Petri Nets and Other Models of Concurrency, pp. 505–514. Springer (2007)
23. Ponce de León, H., Mokhov, A.: Building bridges between sets of partial orders. In: Language and Automata Theory and Applications - 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings, pp. 145–160 (2015)
24. Ponce de León, H., Rodríguez, C., Carmona, J., Heljanko, K., Haar, S.: Unfolding-based process discovery. In: Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings, *Lecture Notes in Computer Science*, vol. 9364, pp. 31–47. Springer (2015)
25. Quinlan, J.R.: Induction of decision trees. Machine Learning **1**(1), 81–106 (1986)
26. Rodríguez, C., Sousa, M., Sharma, S., Kroening, D.: Unfolding-based partial order reduction. In: 26th International Conference on Concurrency Theory, CONCUR 2015, Madrid, Spain, September 1.4, 2015, pp. 456–469 (2015)

27. Rykunov, M.: Design of asynchronous microprocessor for power proportionality. Ph.D. thesis, Newcastle University (2013)
28. Saarikivi, O., Ponce de León, H., Kähkönen, K., Heljanko, K., Esparza, J.: Minimizing test suites with unfoldings of multithreaded programs. ACM Trans. Embedded Comput. Syst. **16**(2), 45:1–45:24 (2017)
29. Valmari, A.: The state explosion problem. In: Lectures on Petri nets I: Basic models, pp. 429–528. Springer (1998)
30. Wegener, I.: The Complexity of Boolean Functions. Johann Wolfgang Goethe-Universitat (1987)
31. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Computer Aided Verification, pp. 17–36. Springer (2002)