# Building Bridges Between Sets of Partial Orders

Hernan Ponce de Leon, Andrey Mokhov

# Building Bridges Between Sets of Partial Orders

Hernán Ponce-de-León[1]⋆ and Andrey Mokhov[2]

[1]Helsinki Institute for Information Technology HIIT and Department of Computer Science and Engineering, School of Science, Aalto University
hernan.poncedeleon@aalto.fi
[2]School of Electrical and Electronic Engineering, Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk

**Abstract.** Partial orders are a fundamental mathematical structure capable of representing true concurrency and causality on a set of atomic events. In this paper we study two mathematical formalisms capable of the compressed representation of sets of partial orders: Labeled Event Structures (LESs) and Conditional Partial Order Graphs (CPOGs). We demonstrate 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 the explicit enumeration of each scenario. These transformations reveal the superior expressive power of CPOGs as well as the cost of this expressive power. The proposed algorithms make use of an intermediate mathematical formalism, called Conditional Labeled Event Structures (CLESs), which combines the advantages of LESs and CPOGs. All three formalisms are compared on a number of benchmarks.

## 1 Introduction

Partial orders – the protagonists of this paper – play a fundamental role in the concurrency theory. The concept has a very simple definition: 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 concurrent ($a \not\leq b$ and $b \not\leq a$). Partial orders arise in numerous application areas such as model checking, process mining, concurrent programming, and VLSI design to name but a few. In this paper we do not focus on a particular application area, however, we use partial orders coming from the VLSI design domain as real-life benchmarks (specifically we use partial orders corresponding to processor instructions and on-chip communication protocols).

A single partial order can capture a single behavioral scenario of a modeled system. However, real-life systems rarely exhibit just a single scenario; in fact, we routinely design systems exhibiting millions of scenarios, each being a partial order defined on a subset of events that may occur in a system. How do we represent all of those partial orders? One can, of course, simply list them explicitly but this is clearly not a scalable solution – 6.6 trillion different partial orders can be defined on just 10 events!

In this paper we study two mathematical formalisms to compactly represent sets of partial orders: Labeled Event Structures (LESs) [1] and Conditional Partial Order Graphs (CPOGs) [2]. 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, while CPOGs naturally come from hardware specifications and implementations, where partial orders are pre-encoded with Boolean vectors (low-level signals, instruction opcodes, etc.).

⋆ This research was done while the author was preparing his thesis at INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France.

This brings us to the main contribution of this paper: 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: CPOGs are often more demanding from the algorithmic complexity point of view. The proposed algorithms make use of a new mathematical formalism, called Conditional Labeled Event Structures (CLESs), which 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.

To the best of the authors' knowledge, no other mathematical model has been directly used for the task of compressed representation of sets of partial orders, hence we only build one (bidirectional) bridge between LESs and CPOGs. If one would like to use other models for this task (for example Petri Nets or Message Sequence Charts), it is possible to reuse existing bridges to connect to the body of our work, e.g., one can obtain a LES from a Petri Net via its unfolding [3].

## 2 Preliminaries

This section introduces two formalisms that compactly represent partial orders: Labeled Event Structures [1] and Conditional Partial Order Graphs [2].

### 2.1 Labeled Event Structures

*Event Structures*[1] can represent several execution scenarios of a system by means of so called *configurations*. We study their widely used extension, called *Labeled Event Structures*, whose events are labeled with actions over a fixed alphabet $L$.

**Definition 1.** *A labeled event structure (LES) 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.*

*Remark 1.* Note that 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 $\#_r$ (*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 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 depend. 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 $\Omega(\mathcal{E})$.*

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

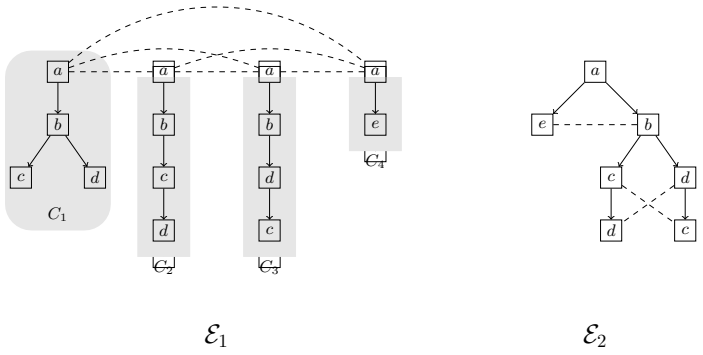


Fig. 1: Two Labeled Event Structures representing the same set of partial orders.

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.

The local configuration $\lceil e \rceil$ of an event $e$ is a set of events on which it causally depends, i.e. $\lceil e \rceil \triangleq \{e' \in E \mid e' \leq e\}$; and its future $\lfloor e \rfloor$ is the set of events that causally depend on it, i.e. $\lfloor e \rfloor \triangleq \{e' \in E \mid e < e'\}$. Since a configuration together with the causality relation form a partial order, 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. 1 shows an LES $\mathcal{E}_1$ defined on alphabet $L = \{a, b, c, d, e\}$ which contains four maximal configurations $C_1$-$C_4$. Note 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). It can be observed that not much compression is achieved by $\mathcal{E}_1$. The LES $\mathcal{E}_2$ represents the same set of partial orders, i.e. $\pi(\Omega(\mathcal{E}_1)) = \pi(\Omega(\mathcal{E}_2))$, and it is more compact.

### 2.2 Conditional Partial Order Graphs

A *Conditional Partial Order Graph* (CPOG) is a quintuple $H = (V, A, X, \phi, \rho)$, where $V$ is a set of vertices, $A \subseteq V \times V$ is a set of arcs between them, and $X$ is a set of operational variables. An opcode is an assignment $(x_1, x_2, \ldots, x_{|X|}) \in \{0, 1\}^{|X|}$ of 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. 2 (top) shows an example of a CPOG containing 5 vertices and 6 arcs; there are two operational variables $x$ and $y$; the restriction function is $\rho = 1$, hence, all four opcodes $(x, y) \in \{0, 1\}^2$ are allowed. Vertices and arcs labeled by 1 are called unconditional (conditions equal to 1 are not depicted in the graph). 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. 2 (bottom). The leftmost projection is obtained by keeping 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. Hence, vertex $e$ disappears, because its condition evaluates to 0: $\phi_e = x \wedge y = 0$. Arcs $\{c \to d, d \to c\}$ disappear for the same reason. Note also that although the condition on arc $a \to e$ evaluates to 1 (in fact it is constant 1) the arc is still excluded from the projection because one of the vertices it connects (vertex $e$) is excluded and an arc cannot appear in a graph without one of its adjacent vertices.

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

[1] In this article, we restrict to prime event structures.

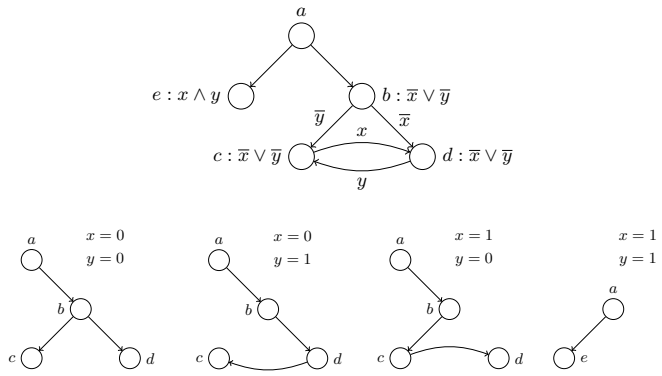


Fig. 2: Conditional Partial Order Graph and the corresponding set of partial orders

of different partial orders on events $V$ according to $2^{|X|}$ possible opcodes. We will use notation $H_{|\psi}$ to denote a projection of a CPOG $H$ under an opcode $\psi = (x_1, x_2, \dots x_{|X|})$. A projection $H_{|\psi}$ is called *valid* iff opcode $\psi$ is allowed by the restriction function, i.e. $\rho(x_1, x_2, \dots x_{|X|}) = 1$, and the resulting graph is acyclic. The latter requirement guarantees that the graph defines a partial order. A CPOG $H$ is *well-formed* iff every allowed opcode produces a valid projection. The graph $H$ in Fig. 2 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 $H$ therefore defines a set of partial orders $P(H)$.

**Complexity.** The original definition of CPOG complexity [2] 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., $|\overline{x} \wedge \overline{y}| = 2$ and $|1| = 0$. The complexity of the CPOG shown in Fig. 2 is thus equal to 10 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. Intuitively, since $\phi_b = \phi_c = \phi_d = \overline{x} \vee \overline{y}$ we can compute condition $\overline{x} \vee \overline{y}$ only once and reuse the result three times. Furthermore, one can notice that conditions $\phi_b = \overline{x} \vee \overline{y}$ 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.

## 3 Enriched and Conditional LESs

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 the next subsection we show that a similar correspondence between opcodes and partial orders can be established by LESs if we enrich them with additional information on conflict resolution.

### 3.1 Enriched Labeled Event Structures

Partial orders are represented by maximal configurations of a LES, therefore to extract a partial order from a LES one needs to resolve event conflicts in a certain 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 3.** *An Enriched Labeled Event Structure (ELES) over alphabet $L$ is a tuple $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$ where $(E, \leq, \#, \lambda)$ is a labeled event structure, $\mathcal{L}$ is a total order on $\#$ and $\mathcal{V}$ is a set of vectors of length $|\mathcal{L}|$.*

A *conflict solver* is a vector $v \in \{0,1\}^{|\mathcal{L}|}$ indicating which event is chosen in each conflicting pair (conflict $\mathcal{L}[i]$ is resolved by $v[i]$'s event in the conflict). Not every conflict solver is acceptable as illustrated in Fig. 3: any solver that chooses $d_2$ over $d_1$ must also choose $c_1$, because $c_2$ is in future of $d_1$; therefore vector $111$ is disallowed. This is not the only restriction. 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. Let $\overline{E}$ denote events that are not selected by a conflict solver $v$, i.e. $\overline{E} = \{e \in E \mid \exists i, j : v[i] = j \ \wedge \ \mathcal{L}[i][1-j] = e\}$, the conflict solver is *valid* iff 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.

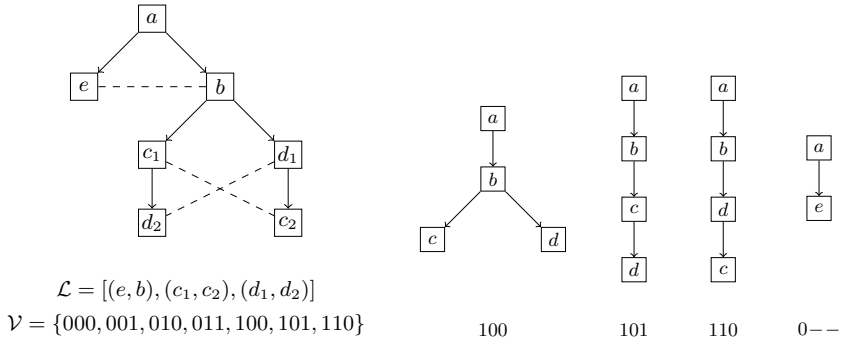


Fig. 3: An ELES and its conflict solvers

*Example 1.* Consider the ELES shown in Fig 3. If event $e$ is chosen ($v[1] = 0$), the resolution of the other conflicts becomes unimportant since the configuration obtained is already maximal. However if event $b$ is chosen ($v[1] = 1$), other conflicts need to be resolved, hence $\mathcal{V} = \{000, 001, 010, 011, 100, 101, 110\}$.

The following proposition characterizes the set of valid conflict solvers for a given labeled event structure (the proof can be found in [4]).

**Proposition 1.** *Let $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$ be 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.*

The above result shows how to compute a set of valid conflicts solvers for a LES and therefore each LES can be easily extended into the corresponding ELES. This means that both LESs and CPOGs can be used when one needs to store partial orders in a compressed form and access them by providing the corresponding opcodes. 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: vertex $c$ from the CPOG in Fig. 2 needs to be represented by two events ($c_1$ and $c_2$) in the LES of Fig. 3. In order to avoid this redundancy, 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*. The next section shows that CLESs are of particular interest when transforming LESs into CPOGs and vice versa.

**Definition 4.** *A Conditional Labeled Event Structure (CLES) over alphabet $L$ is a tuple $\mathcal{E} = (E, \leq, \#, \lambda, X, \phi, \rho)$ 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.*

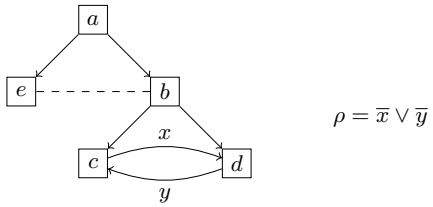


Fig. 4: Conditional Labeled Event Structure.

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; if the structure is acyclic and conditions are dropped, we get a LES.

The CLES in Fig. 4 represents the same partial orders as the CPOG and the LES in Figs. 2 and 3. If we compare it to the CPOG, a conflict is introduced, but the number of Boolean conditions is reduced. Comparing it to the LES, one can see that not only the number of events is reduced, but also the number of conflicts. The cardinality of the causality relation is preserved, but the information about Boolean labeling needs to be stored, i.e. $\phi_{c \leq d} = x$ and $\phi_{d \leq c} = y$. In the next section we introduce a complexity measure for such conditional, or parameterized, structures that we will use to compare (Enriched) LESs, CPOGs and CLESs to each other.

## 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,* i.e. elements labeled with Boolean conditions.

**Definition 5.** *A mathematical structure over a set of elements $S$ is called a parameterized structure if the elements are labeled with Boolean conditions $\phi : S \rightarrow \Phi$, where $\Phi$ is a set of predicates (Boolean functions) on $X$, that is $\Phi \subseteq X \rightarrow \{0, 1\}$.*

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 we will use to compare compactness of CPOGs, LESs and CLESs in our experiments.

**Complexity measure.** Instead of treating each predicate in $\Phi$ separately let us construct a *Boolean circuit* [5] that computes all of them together and makes use of shared intermediate results. This is exactly what happens in practice regardless of whether a parameterized structure is used for verification purposes or in hardware synthesis. The *decoding complexity* of a predicate set $\Phi$ is the number of variables in $\Phi$ plus the number of gates in the smallest circuit[2] computing all predicates.

**Definition 6.** *The Complexity of a parameterized structure with predicate set $\Phi$ on a set of elements $S$ is the decoding complexity of $\Phi$ plus the number of elements in $S$.*

Fig. 5 shows a circuit that computes predicates in $\Phi = \{1, \overline{x} \vee \overline{y}, x \wedge y, x, y, \overline{x}, \overline{y}\}$ required for the CPOG shown in Fig. 2. Note that trivial conditions $1$, $x$ and $y$ require no

[2] In our experiments we restrict the number of inputs of each gate to 2. Since finding the smallest circuit is a very hard problem, we use approximation of the circuit complexity measure [6].

computation at all and are therefore omitted in the diagram. We do not need a circuit to compute conditions of a LES which are always 1; only a single NAND gate is required for the CLES in Fig. 4. Therefore, the CPOG complexity is considered to be equal to 17 (2 variables + 4 gates + 5 vertices + 6 arcs); the LES complexity is 16 (7 events + 6 direct causality arcs + 3 direct conflicts); finally, the CLES complexity is 15 (2 variables + 1 gate + 5 vertices + 6 direct causality arcs + 1 direct conflict).

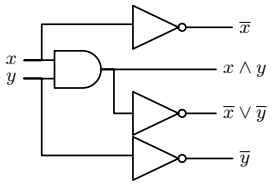


Fig. 5: Circuit computing conditions for CPOG in Fig. 2.

**Comparison of Parameterized Structures.** We compare LESs, CPOGs and CLESs on a number of benchmarks coming from the VLSI design domain, in particular, on-chip communication controllers [7] and processor microarchitectures [8]. 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 $\# = \emptyset$ and every LES is a CLES with $\phi = 1$, CLESs have at most the same complexity as CPOGs and LESs.

*Example 2. Phase encoders* [7] are communication controllers capable of generating all permutations of $n$ events. They are badly handled by acyclic structures as can be seen in Fig. 6 (right). The LES for a phase encoder with $n = 3$ has complexity 33 while its corresponding CPOG has complexity 15. In general, the complexity of CPOGs for phase encoders grows quadratically with $n$, while the complexity of LESs grows exponentially: one can see that the LES for a phase encoder of size $n$ must have $n!$ events on its lowest level. In fact, a LES must contain at least as many events as there are partial orders in it.

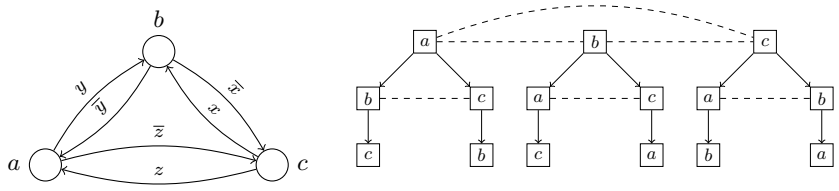


Fig. 6: Phase encoder for $n = 3$ represented by a CPOG (left) and a LES (right).

*Example 3.* Decision trees [9] are binary trees that can be used to model choices and their consequences. LESs for decision trees are smaller than CPOGs as the number of direct conflicts is smaller than the decoding complexity for conditions. This is illustrated in Fig. 7 where the LES on the right 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 in Section 5, the complexity of a CPOG never exceeds the complexity of the corresponding LES by more than just a constant factor.

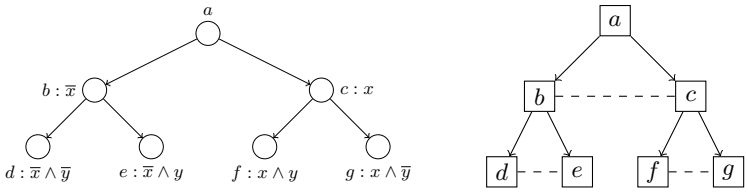


Fig. 7: A decision tree represented by a CPOG (left) and a LES (right).

*Example 4.* 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. CLESs are strictly smaller than both CPOGs and LESs in this example, as shown in Fig. 8: the CPOG, the LES and the CLES have complexity 35, 52, and 30, respectively.

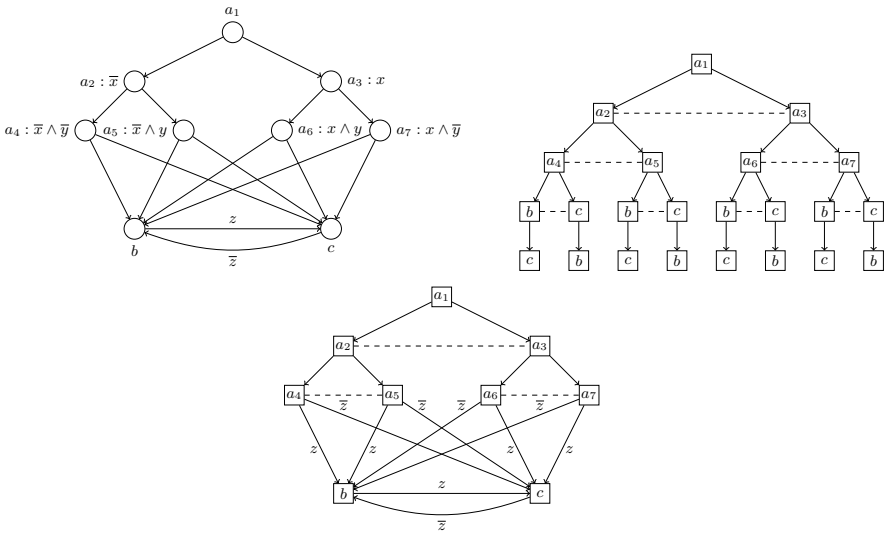


Fig. 8: A tree of phase encoders represented by a CPOG, a LES and a CLESs.

Table 1 provides a summary of our experimental comparison of the complexity of CPOGs, LESs and CLESs. We compressed different sets of partial orders: phase encoders, decision trees, trees of phase encoders, as well as several sets of processor instructions (from ARM Cortex M0 and Intel 8051 processors [8]).

## 5 Transformations

This section presents the main contribution of this work: algorithms for transforming LESs into CPOGs and vice versa without performing an intermediate uncompression step. Note that both algorithms make use of CLESs as an intermediate representation, which is not essential but convenient. The proofs of the results in this section can be found in [4].

**From LESs to CPOGs.** 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 is obtained which can then be folded to remove redundant vertices. 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

| Name | Scenarios | Complexity | | |
|---|---|---|---|---|
| | | CPOG | LES | CLES |
| Phase encoder | 24 | 24 | 158 | 24 |
| | 120 | 35 | 825 | 35 |
| | 720 | 48 | 5001 | 48 |
| Decision tree | 8 | 44 | 39 | 36 |
| | 16 | 97 | 76 | 76 |
| | 32 | 195 | 156 | 156 |
| Trees of phase encoders | 16 | 70 | 108 | 58 |
| | 29 | 43 | 158 | 41 |
| | 32 | 136 | 220 | 114 |

| Name | Scenarios | Complexity | | |
|---|---|---|---|---|
| | | CPOG | LES | CLES |
| ARM Cortex M0 | 5 | 26 | 28 | 26 |
| | 6 | 27 | 35 | 27 |
| | 7 | 26 | 38 | 26 |
| | 8 | 28 | 43 | 28 |
| | 9 | 28 | 46 | 28 |
| | 10 | 29 | 46 | 29 |
| | 11 | 30 | 50 | 30 |
| Intel 8051 | 5 | 34 | 48 | 34 |
| | 6 | 35 | 52 | 35 |
| | 7 | 36 | 56 | 36 |
| | 8 | 37 | 56 | 37 |
| | 9 | 46 | 71 | 46 |
| | 10 | 47 | 81 | 47 |
| | 11 | 51 | 90 | 51 |

Table 1: Experimental results

with all of its causal predecessors. This can be encoded in the restriction function of the resulting CPOG as follows[3]:

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

For the example shown in Fig. 3 this generates the following restriction function:

$$(v_b \Rightarrow v_a) \wedge (v_e \Rightarrow v_a) \wedge (v_{c_1} \Rightarrow v_b) \wedge (v_{d_1} \Rightarrow v_b) \wedge (v_{d_2} \Rightarrow v_{c_1}) \wedge (v_{c_2} \Rightarrow v_{d_1}) \wedge (\overline{v}_e \vee \overline{v}_b) \wedge (\overline{v}_{c_1} \vee \overline{v}_{c_2}) \wedge (\overline{v}_{d_1} \vee \overline{v}_{d_2})$$

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:

$$v_a = v_b = v_{c_1} = v_{d_1} = 1, v_{c_2} = v_{d_2} = v_e = 0$$
$$v_a = v_b = v_{d_1} = v_{c_2} = 1, v_{c_1} = v_{d_2} = v_e = 0$$
$$v_a = v_b = v_{c_1} = v_{d_2} = 1, v_{c_2} = v_{d_1} = v_e = 0$$
$$v_a = v_e = 1, v_b = v_{c_1} = v_{c_2} = v_{d_1} = v_{d_2} = 0$$

However not only maximal configurations satisfy the function, for example, the empty configuration clearly satisfies it as well: $v_a = v_b = v_{c_1} = v_{c_2} = v_{d_1} = v_{d_2} = v_e = 0$.

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 and only if, for every event $e \in E$ one of the following conditions holds: *(i)* event $e$ belongs to the configuration; *(ii)* there exist an event $f$ which belongs to the configuration and prevents $e$. The restriction function (1) can now be refined to allow only maximal configurations:

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

Coming back to the example in Fig. 3, the refined restriction function (2) 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. This transformation method is summarized in Algorithm 1.

**Algorithm 1** Transforming a LES into a CPOG

**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: $\forall v \in V : \phi_v = x_v$
3: $\forall e = (v_1, v_2) \in A : \phi_e = v_1 \wedge v_2$
4: **while** $\exists v_1, v_2 \in V : \lambda(v_1) = \lambda(v_2)$ **do** (for $v \notin V$)
5: $\quad V = V \backslash \{v_1, v_2\} \cup \{v\}$
6: $\quad \forall v' \in V : v' \leq v \Leftrightarrow v' \leq v_1 \vee v' \leq v_2$
7: $\quad \forall v' \in V : v \leq v' \Leftrightarrow v_1 \leq v' \vee v_2 \leq v'$
8: $\quad \phi_v = \phi_{v_1} \vee \phi_{v_2}$
9: $\rho = (\bigwedge_{e \# f} \neg\phi_e \vee \neg\phi_f)(\bigwedge_{e \leq f} \phi_f \Rightarrow \phi_e)(\bigwedge_{e \in E} \phi_e \vee \bigvee_{e \# f} \phi_f)$
10: **return** $H = (V, A, X, \phi, \rho)$

The scenarios represented by the CPOG obtained by merging events coincide with the maximal configurations of the LES.

**Theorem 1.** *Given an LES $\mathcal{E} = (E, \leq, \#, \lambda)$, Algorithm 1 constructs a CPOG $H = (V, A, X, \phi, \rho)$ such that $P(H) = \pi(\Omega(\mathcal{E}))$.*

*Proof.* The algorithm transforms the LES $\mathcal{E}$ into a CLES $G$ and the later into a CPOG $H$. The proof shows that *i)* projections of $G$ coincide with maximal configurations of $\mathcal{E}$; and *ii)* $G_{|\psi} = H_{|\psi}$ for any valid opcode $\psi$. The complete proof can be found in [4].

The complexity of the CPOG constructed by this procedure is linear with respect to the size of the original LES, as stated by the following theorem.

**Theorem 2.** *Given an LES $\mathcal{E} = (E, \leq, \#, \lambda)$, Algorithm 1 constructs a CPOG $H = (V, A, X, \phi, \rho)$ of complexity $\Theta(|\mathcal{E}|)$.*

*Proof.* The resulting CPOG contains at most the same number of vertices and arcs as the original LES. Moreover, $|X| = |E|$, $|\phi| = O(|E|)$, and the size of the restriction function $\rho$ is linear with respect to $|\mathcal{E}|$ as seen from (2). The complete proof can be found in [4].

**From CPOGs to LESs.** In order to transform a CPOG into an LES, the graph is 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 (that containing no events) 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 following formula for each vertex $a$:

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

Whenever such a combination exists and the formula reduces to $\phi$, we add the event to the unfolding, appropriately connecting it to $P$ and labeling it by the $\phi$. The unfolding

[3] Some optimization techniques allow to consider only direct causality and direct conflicts. We make use of this observation in our further examples.

procedure finishes when (3) is no longer satisfiable. Finally, conditions are replaced by conflicts: for every pair of mutually exclusive events, their Boolean conditions are removed and conflict $e_a \# e_b$ is added.

Algorithm 2 shows the complete transformation procedure. Function $PE(\mathcal{E}, H)$ takes the current unfolding $\mathcal{E}$ and CPOG $H$, and returns a set $pe$ of possible extensions satisfying (3), and for each $e \in pe$, its set of predecessors $P_e$, label $\lambda^{pe}(e)$, and condition $\psi_e^{pe}$.

**Algorithm 2** Transforming a CPOG into a LES

**Require:** a well-formed CPOG $H = (V, A, X, \phi, \rho)$
1: $\mathcal{E} = (E, \leq, \#, \lambda, X, \varphi, \rho) := (\emptyset, \emptyset, \emptyset, \emptyset, X, \emptyset, \rho)$
2: $(pe, \{P_e\}_{e \in pe}, \lambda^{pe}, \varphi^{pe}) := PE(\mathcal{E}, H)$
3: **while** $pe \neq \emptyset$ **do**
4: add some $e \in pe$ to $E$ and set $P_e \leq e, \lambda(e) = \lambda^{pe}(e)$ and $\varphi_e = \varphi_e^{pe}$
5: **while** $\exists e_a, e_b \in E : \neg\varphi_{e_a} \vee \neg\varphi_{e_b}$ **do**
6: set $e_a \# e_b$
**return** $(E, \leq, \#, \lambda)$

We can use a SAT-solver to 'guess' a combination of an event $a$ and a predecessor set $P$ satisfying (3).

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

*Proof.* Consider a CPOG with a single vertex $v$ having condition $\phi$. Deciding if $v$ is a possible extension requires checking $\phi$ for being a contradiction. See more details in [4].

The unfolding algorithm is deterministic: the resulting LES does not depend on the order in which events are added into the unfolding due to the following result proved in [4].

**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 $E \cup \{e_a\}$.*

*Example 5.* Consider the CPOG shown in Fig. 2. The unfolding procedure starts with $E = \emptyset$ and keeps checking vertices of the CPOG for possible extensions. At start, only vertex $a$ can be added. For example, the constraint imposed by non-predecessors in (3) will include $\neg\phi_{a \to b} = 0$ 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 $\overline{x} \vee \overline{y}$ 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 $\phi_{e_b^0} = \overline{x} \vee \overline{y}$ 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 $\phi_{e_c^0} = \overline{y}$ and $\phi_{e_d^0} = \overline{x}$. 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: $(x \wedge y) \wedge (\overline{x} \vee \overline{y}) = 0$, therefore we add conflict $e_b^0 \# e_e^0$. Due to the same reasoning, conflicts $e_c^0 \# e_c^1$ and $e_d^0 \# e_d^1$ are added. Finally, when all Boolean conditions are removed from the CLES, the resulting LES is that of Fig. 3.

The result below shows that the unfolding algorithm is correct, i.e. it preserves set of partial orders.

**Theorem 3.** *Let $H = (V, A, X, \phi, \rho)$ be a well-formed CPOG and $\mathcal{E} = (E, \leq, \#, \lambda)$ the LES obtained by the unfolding procedure, then $\pi(\Omega(\mathcal{E})) = P(H)$.*

*Proof.* The algorithm transforms the CPOG $H$ into a CLES $G$ and the later into a LES $\mathcal{E}$ by replacing conditions by conflicts. The proof shows that *i)* projections of $H$ and $G$ over a valid opcode coincide; and *ii)* projections over $G$ and maximal configurations of $\mathcal{E}$ coincide. The complete proof can be found in [4].

As one can see, the transformation procedure from CPOGs to LESs is significantly more computationally intensive: 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 [10] to speed up the algorithm.

## 6 Conclusion

The paper discusses the use of two models (LESs and CPOGs) for the compressed representation of sets of partial orders. We show that LESs work well on most practical examples, 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 transforming 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.

Further work includes optimization of the presented algorithms, their integration with Workcraft EDA suite [**?**], and validation on larger case studies coming from process mining and VLSI design domains.

## References


1. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, part I. Theoretical Computer Science **13** (1981) 85–108
2. Mokhov, A., Yakovlev, A.: Conditional partial order graphs: Model, synthesis, and application. IEEE Trans. Computers **59**(11) (2010) 1480–1493
3. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan's unfolding algorithm. In: Tools and Algorithms for the Construction and Analysis of Systems. Springer (1996) 87–106
4. Ponce de León, H., Mokhov, A.: Building bridges between sets of partial orders. `http://hal.inria.fr/hal-01060449` (2014) Technical report. Visited on 4/9/2014.
5. Wegener, I.: The Complexity of Boolean Functions. Johann Wolfgang Goethe-Universitat (1987)
6. Berkeley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification, Release 70930. `http://www.eecs.berkeley.edu/~alanmi/abc/`
7. 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, IEEE Computer Society (2007) 105–116
8. 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) (2014) 1552–1566
9. Sung-hyuk, C.: A genetic algorithm for constructing compact binary decision trees. In: International Journal of Information Security and Privacy. (2010) 32–60
10. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Computer Aided Verification, Springer (2002) 17–36
11. 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. Springer (2007) 505–514