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]INRIA and LSV, École Normale Supérieure de Cachan and CNRS, France
ponce@lsv.ens-cachan.fr
[2]School of Electrical and Electronic Engineering, Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk

Abstract. Partial order is a fundamental mathematical structure capable of representing true 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 modeled 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 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 transformation of 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, which we call Conditional Labeled Event Structures (CLESs), that combines the advantages of LESs and CPOGs. Finally, 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 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 capable of the compressed representation of sets of partial orders: Labeled Event Structures (LESs) [1] and Conditional Partial Order Graphs (CPOGs) [2]. We introduce the formalisms in Sections 2 and 3 respectively, and show how to synthesize LESs and CPOGs 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.

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, and the software tools are incompatible. As will be demonstrated in Section 5, 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 brings us to the main contribution of this paper: we present two direct transformation algorithms (Section 6) 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), see Section 4.2, 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.

To the best of the authors' knowledge, no other mathematical models have 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 often 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].

The presented compression and transformation techniques are not only applicable to partial orders, but also to other related concurrency models, such as combined traces [4].

2 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 an alphabet L is a 4-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

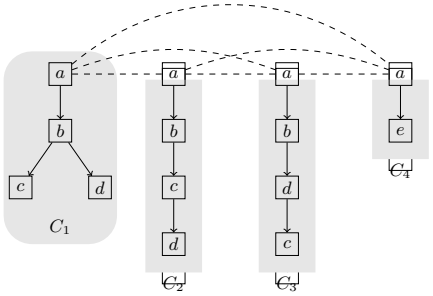

Fig. 1: A Labeled Event Structure and its maximal configurations.

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 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 causality depends, i.e. $[e] \triangleq \{e' \in E \mid e' \leq e\}$ and its* future *$\lfloor e \rfloor$ is the set of events that causally depend on it, i.e. $\lfloor e \rfloor \triangleq \{e' \in E \mid e < e'\}$.*

Events of a configuration together with the causality relation (restricted to those events) form a partial order. One can therefore consider a LES $\mathcal{E}$ as a compressed representation of a set of partial orders formed by maximal configurations $\Omega(\mathcal{E})$.

Fig. 1 shows an example of a LES 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, and showing all these conflicts would make the diagram unreadable). An alert reader may notice that not much compression is achieved by the LES shown in Fig. 1. Indeed, this can be significantly improved upon as discussed below.

Synthesis and Optimization of LESs. Given a set of partial orders, the objective is to synthesize a compact LES that represents them. The idea behind the synthesis approach presented below is to start by putting minimal events of each partial order in conflict as shown in Fig. 1, and then to compress the LES by merging events with the same label that also have the same local configuration – see Algorithm 1.

Remark 2. 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 $\{a\}$

Algorithm 1

Require: $\mathcal{E} = (E, \leq, \#, \lambda)$
Ensure: $\mathcal{E}'$ such that $\lambda(\Omega(\mathcal{E})) = \lambda(\Omega(\mathcal{E}'))$
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: $\forall e' \in E : e' \leq e \Leftrightarrow e' \leq e_1 \vee e' \leq e_2$
4: $\forall e' \in E : e \leq e' \Leftrightarrow e_1 \leq e' \vee e_2 \leq e'$
5: $\forall e' \in E : e \# e' \Leftrightarrow e' \# e_1 \vee e' \# e_2$
6: $\lambda(e) = \lambda(e_1)$

ceases to be maximal being superseded by configuration $\{a, b\}$, thereby leading to removal of po_1 from the set of partial orders. To fix 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. 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. 2 illustrates the application of Algorithm 1 to the LES from Fig. 1. One can see that the compressed LES (bottom left) is significantly smaller. In fact, it is the smallest LES containing the given maximal configurations C_1-C_4. Moreover, Algorithm 1 is denotational deterministic and therefore the result is unique (up to isomorphisms) [5].

Proposition 1. *Algorithm 1 is deterministic, i.e. the order in which events are merged is not important and the resulting LES is unique.*

One can use the same approach for combining two sets of partial orders S_1 and S_2 represented by LESs. This is done by putting the minimal events of the LESs in conflict and compressing the result by Algorithm 1. The resulting LES *canonically* represents partial orders in $S_1 \cup S_2$ (duplicates are removed automatically). Note that it is not required to uncompress given sets, hence we argue that LESs have good compositionality.

3 Conditional Partial Order Graphs

A *Conditional Partial Order Graph* (CPOG) [2] is a quintuple $H = (V, A, X, \phi, \rho)$, where V is a set of vertices, A is a set of arcs between them, and X is a set of 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 ρ of the graph, i.e., $\rho(x_1, x_2, \ldots, x_{|X|}) = 1$. Function ϕ assigns a Boolean condition $\phi(z)$ to every vertex and arc $z \in V \uplus A$ of the graph.

Fig. 3 (above) shows an example of a CPOG. This CPOG contains $|V| = 5$ vertices and $|A| = 6$ arcs; there are two operational 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).

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. 3 (below). 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

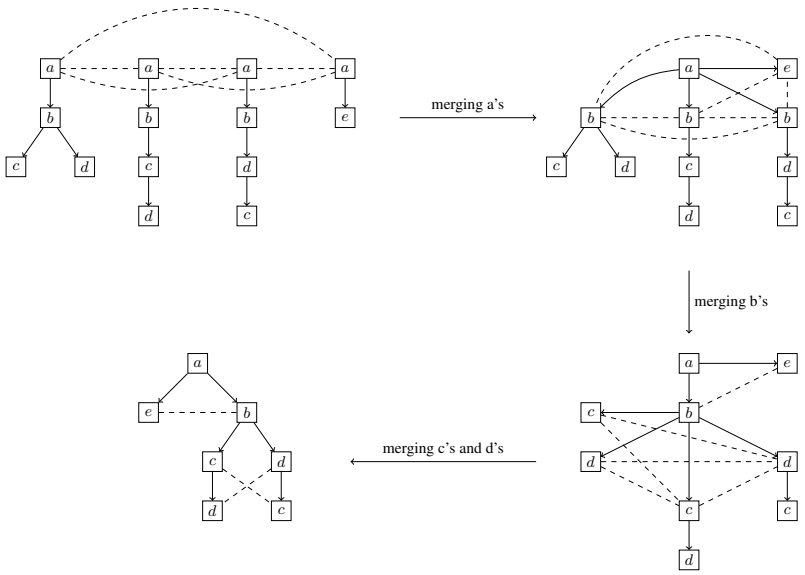

Fig. 2: Compressing a LES by merging events.

e disappears, because its condition evaluates to 0: $\phi_e = x \wedge y = 0 \wedge 0 = 0$. 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 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 opcode $\psi = (x_1, x_2, \dots x_{|X|})$. A projection $H_{|\psi}$ is called *valid* iff opcode ψ 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. 3 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 $H_1 = (V_1, A_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, A_2, X_2, \rho_2, \phi_2)$ can be combined into $H = (V_1 \cup V_2, A_1 \cup A_2, X_1 \cup X_2, \rho_1 + \rho_2, \phi)$, where conditions ϕ are defined as $\forall z \in V_1 \cup V_2 \cup A_1 \cup A_2 : \phi(z) \triangleq \rho_1\phi_1(z) + \rho_2\phi_2(z)$. The result contains the union of the sets of partial orders defined by H_1 and H_2, i.e. $P(H) = P(H_1) \cup P(H_2)$ [6].

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 ϕ, e.g., $|\overline{x} \wedge \overline{y}| = 2$ and $|1| = 0$. The complexity of the CPOG shown in Fig. 3 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

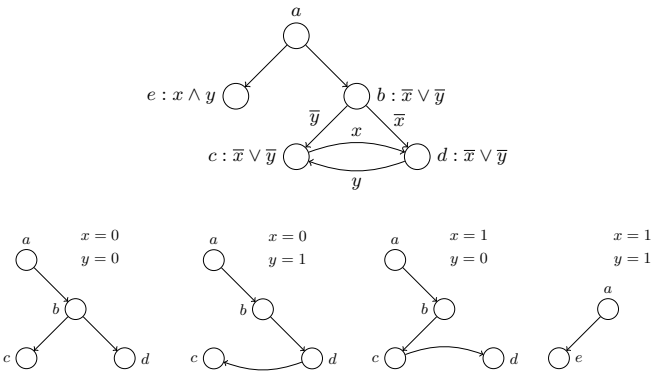

Fig. 3: Conditional Partial Order Graph and the corresponding set of partial orders

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 ϕ_e we can efficiently compute ϕ_b by a single inversion operation. In Section 5 we introduce an improved measure of complexity (based on Boolean circuits) which is free from the above shortcomings.

4 Enriched and Conditional LES

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

4.1 Enriched Labeled Event Structures

Partial orders are represented by maximal configurations of a LES, therefore, in order 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 $\#$[1] and $\mathcal{V}$ is a set of vectors of length $|\mathcal{L}|$.*

A *conflict solver* is a vector $v \in \{0,1\}^{|\mathcal{L}|}$ that determines which event is chosen in each conflicting pair (conflict $\mathcal{L}[i]$ is resolved by the value v_i). Not every conflict solver is acceptable as illustrated in Fig. 4: any solver that chooses d_2 over d_1 must also choose c_1, because c_2 is in future of d_1 and cannot be chosen, therefore vector 111 is disallowed. 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. 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* 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.

[1] Note that it is enough to consider direct conflicts $\#_r$ only.

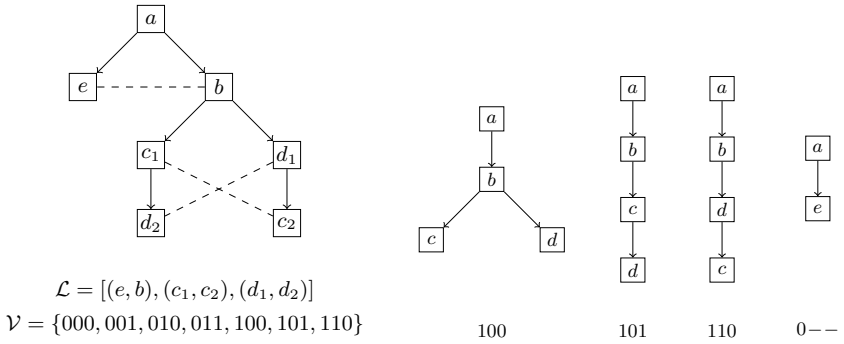

Fig. 4: An ELES and its conflict solvers

Example 1. Consider the ELES shown in Fig 4. If we have $v_1 = 0$ (event e is chosen), the resolution of the other conflicts becomes unimportant: the configuration obtained is already maximal. However if $v_1 = 1$ (event b is chosen), other conflicts need to be resolved, hence $\mathcal{V} = \{000, 001, 010, 011, 100, 101, 110\}$.

Proposition 2. *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.*

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 can not be a configuration, leading to a contradiction. □

4.2 Conditional Labeled Event Structures

The acyclicity of LESs often introduces the redundancy in events: vertex c from the CPOG in Fig. 3 needs be represented by two events (c_1 and c_2) in the LES in Fig. 4. 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 $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V}, X, \phi, \rho)$ where E are events; $\leq$ is a set of arcs; $\#$ represents conflicts; $\mathcal{L}$ is a total order on conflicts and $\mathcal{V}$ the set of valid conflict solvers; X is a set of operational variables; ϕ assigns Boolean conditions to $E, \leq$ and $\#$; and ρ is the restriction function.*

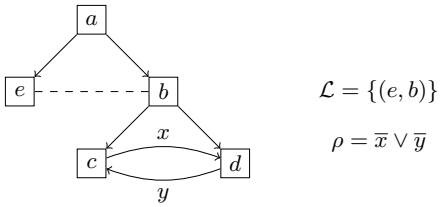

Fig. 5: Conditional Labeled Event Structure.

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

The CLES in Fig. 5 represents the same partial orders as the CPOG and the ELES in Figs. 3 and 4. If we compare it to the CPOG, a conflict is introduced, but the number of Boolean conditions is reduced. Comparing it to the ELES, one can see that not only the number of events is reduced, but also the number of conflicts (and therefore the length of conflict solvers). 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 ELESs, CPOGs and CLESs to each other.

5 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 $\phi : S \to \Phi$, where Φ is a set of predicates (Boolean functions) on X, that is $\Phi \subseteq X \to \{0, 1\}$.*

A CPOG is a parameterized structure whose elements are vertices and arcs. Events and causality/conflict relations are elements of both ELESs and CLESs, but every ELES 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, ELESs and CLESs in our experiments.

Complexity measure. Instead of treating each predicate in Φ separately let us construct a *Boolean circuit* [7] 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 (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 Φ is the number of variables in Φ plus the number of gates in the smallest circuit[2] computing all predicates.

[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 [8].

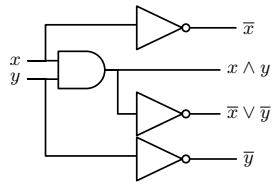

Fig. 6: Circuit computing conditions for CPOG in Fig. 3.

Definition 6. Complexity *of a parameterized structure with a predicate set Φ on a set of elements S is the decoding complexity of Φ plus the total number of elements in S.*

Fig. 6 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. 3. Note that trivial conditions 1, x and y require no computation at all and are therefore omitted in the diagram. We do not need a circuit to compute conditions of an ELES which are always 1. Only a single NAND gate is required for the CLES in Fig. 5. Therefore, the CPOG complexity is considered to be equal to 17 (2 variables + 4 gates + 5 vertices + 6 arcs); the ELES 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).

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 [9] and processor microarchitectures [6]. 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 ELES is a CLES with $\phi = 1$, CLESs have at most the same complexity as CPOGs and ELESs.

Example 2. Phase encoders [9] 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. 7 (right). The ELES 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 ELESs grows exponentially: one can see that the ELES for a phase encoder of size n must have $n!$ events on its lowest level. In fact, an ELES requires at least as many events as there are partial orders in a given set.

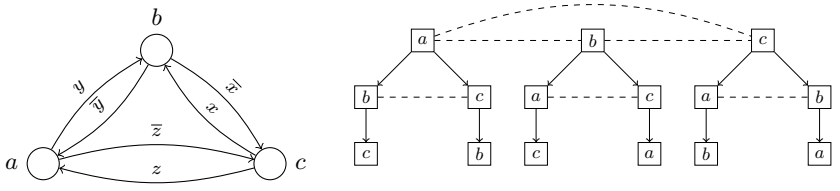

Fig. 7: Phase encoder for $n = 3$ represented by a CPOG (left) and an ELES (right).

Example 3. Decision trees [10] are binary trees that can be used to model choices and their consequences. ELESs 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. 8 where the ELES on the left has complexity 16, while the complexity of the CPOG is 21. Asymptotically the complexity of both ELESs and CPOGs grows linearly with the size of decision trees, so in this example ELESs are better by just a constant factor. In general, as we will demonstrate in Section 6, the complexity of a CPOG never exceeds the complexity of the corresponding ELES by more than just a constant factor.

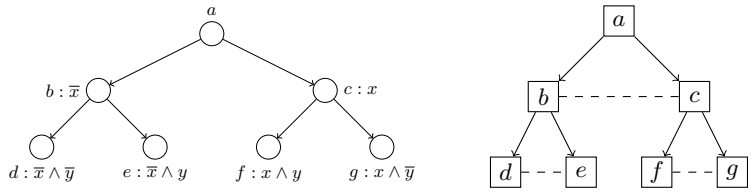

Fig. 8: A decision tree represented by a CPOG (left) and an ELES (right).

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 ELESs in this example, as demonstrated in Fig. 9: the CPOG has complexity 35, the ELES has complexity 52, and the CLES has complexity 30.

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, phase encoding trees, as well as several sets of processor instructions (from ARM Cortex M0 and Intel 8051 processors [6]).

6 Transformations

This section presents the main contribution of this work: algorithms for transforming ELESs into CPOGs and vice versa without performing an intermediate uncompression step. Both algorithms make use of CLESs as an intermediate representation.

From ELESs to CPOGs. Every ELES 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 2 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[3]:

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

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

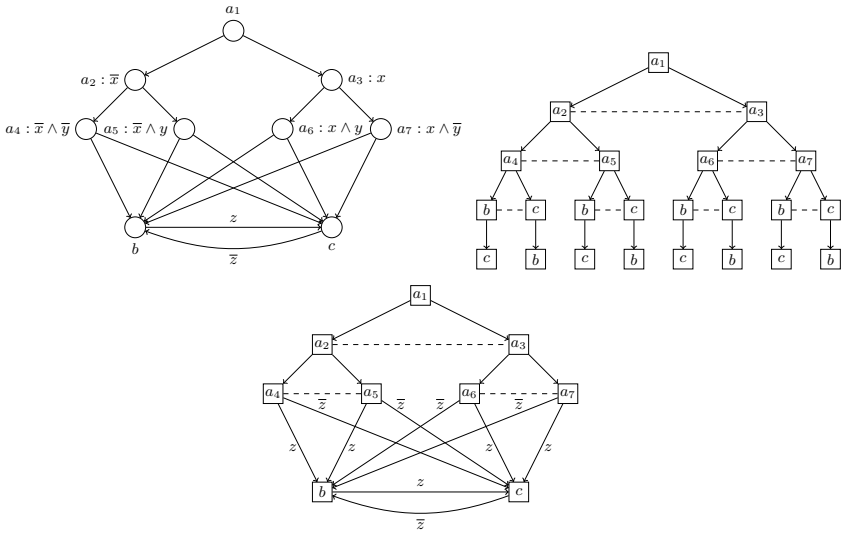

Fig. 9: A tree of phase encoders represented by a CPOG, an ELES and a CLESs.

For the example shown in Fig. 4 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 ELES:

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

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. An event preventing e is called a *spoiler* [11]. Any event e spoils itself as the event can not occurs twice. The set of spoilers of an event e can therefore be 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:

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

		Complexity		
Name	nº of Scenarios	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
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

Coming back to the example in Fig. 4, the additional constraint is:

$$v_a \wedge (v_b \vee v_e) \wedge (v_{c_1} \vee v_{c_2} \vee v_e) \wedge (v_{d_1} \vee v_{d_2} \vee v_e) \wedge$$
$$(v_{c_2} \vee v_{c_1} \vee v_{d_2} \vee v_e) \wedge (v_{d_2} \vee v_{d_1} \vee v_{c_2} \vee v_e) \wedge (v_e \vee v_b \vee v_{c_1} \vee v_{c_2} \vee v_{d_1} \vee v_{d_2})$$

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

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.

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

Theorem 1. *Given an ELES $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$, we can construct a CPOG $H = (V, A, X, \phi, \rho)$ of complexity $\Theta(|\mathcal{E}|)$.*

Proof. By Definition 6 we need to consider the number of vertices, arcs and the decoding complexity of ϕ and ρ. 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 $e \to f \Leftrightarrow e_1 \leq f \vee e_2 \leq f$, therefore we can conclude $|A| \leq |\leq|$. We label each event with a different variable ($|X| = |E|$), hence 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 ϕ. To compute ρ 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 a OR gate) for each arc in $\leq$; each conflict is listed exactly two times in the maximality

Algorithm 2 Transforming an ELES to a CPOG

Require: $\mathcal{E} = (E, \leq, \#, \lambda, \mathcal{L}, \mathcal{V})$ and a set of Boolean variables $\{x_1, \ldots, x_{|E|}\}$
Ensure: H 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** $H = (V, A, X, \phi, \rho)$

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 ρ function is therefore $4 * |\#| + 3 * |\leq| + 2 * |E| - 1 \leq 4 * |\mathcal{E}|$. To conclude, the complexity of the resulting CPOG is $\Theta(|\mathcal{E}|)$. □

Optimization techniques. Below we describe several important optimization techniques that improve the above transformation procedure.

1. *Arc reduction:* the proof above used $|\leq|$ AND gates to compute ϕ. 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 ϕ_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. 4 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.

From CPOGs to ELESs. In order to transform a CPOG into an ELES, the graph is unfolded (in order to obtain an acyclic structure) while keeping conditions that will be replaced by conflicts in the final ELES. For this, a CLES is constructed as an intermediate structure. We start from an empty ELES (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:

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

We use a SAT-solver to 'guess' a combination of an event a and a predecessor set P satisfying (3). Whenever such a combination exists, we add the event to the unfolding, appropriately connecting it to P. The unfolding procedure is finished 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 instead. As the set $\mathcal{V}$ does not depend on the graph, it can be constructed from the event structure itself using Proposition 2, while $\mathcal{L}$ is obtained as a linearization of $\#$. The algorithm is deterministic: the resulting ELES does not depend on the order in which events are added into the unfolding.

Remark 3. As explained in Remark 2, if a partial order po_1 is a prefix of another one po_2, maximal configurations of the obtained ELES will not represent po_1. We deal with this by adding a maximal (no outgoing arcs) vertex $\top$ into the original CPOG with unconditional arcs $a \to \top$ from any vertex a.

Example 5. Consider the CPOG shown in Fig. 3. The unfolding procedure starts with $E = \emptyset$ and keeps checking vertices of the CPOG for possible extensions (see Fig. **??**). 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^0_a to the unfolding with $\phi(e^0_a) = 1$. 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^0_b and e^0_e are added with e^0_a as their predecessor and with $\phi(e^0_b) = \overline{x} \vee \overline{y}$ and $\phi(e^0_e) = x \wedge y$.

At this point $E = \{e^0_a, e^0_b, e^0_e\}$ and we find that c and d are possible extensions adding events e^0_c and e^0_d with event e^0_b as the predecessor and conditions $\phi(e^0_c) = \overline{y}$ and $\phi(e^0_d) = \overline{x}$. Now $E = \{e^0_a, e^0_b, e^0_c, e^0_d, e^0_e\}$ and we find that c and d are possible extensions again. Two new events e^1_c and e^1_d are added. Finally, as E grows to $\{e^0_a, e^0_b, e^0_c, e^1_c, e^0_d, e^1_d, e^0_e\}$, formula (3) becomes unsatisfiable and the unfolding procedure is finished. Conditions of events e^0_b and e^0_e are mutually exclusive: $(x \wedge y) \wedge (\overline{x} \vee \overline{y}) = 0$, therefore we add conflict $e^0_b \# e^0_e$. Due to the same reasoning, conflicts $e^0_c \# e^1_c$ and $e^0_d \# e^1_d$ are added. Finally, when all Boolean conditions are removed from the CLES, the resulting ELES is that of Fig. 4.

As one can see the transformation procedure from CPOGs to ELESs 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 [12] to speed up the algorithm.

7 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-complete. 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.

The further work includes optimization of our implementations 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. Mikulski, L., Koutny, M.: Folded hasse diagrams of combined traces. Inf. Process. Lett. **114**(4) (2014) 208–216
5. Rensink, A.: Denotational, causal, and operational determinism in event structures. In: CAAP. (1996) 272–286
6. 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
7. Wegener, I.: The Complexity of Boolean Functions. Johann Wolfgang Goethe-Universitat (1987)
8. Berkeley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification, Release 70930. `http://www.eecs.berkeley.edu/~alanmi/abc/`
9. 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
10. Sung-hyuk, C.: A genetic algorithm for constructing compact binary decision trees. In: International Journal of Information Security and Privacy. (2010) 32–60
11. Haar, S., Rodríguez, C., Schwoon, S.: Reveal your faults: It's only fair! In: ACSD. (2013) 120–129
12. Zhang, L., Malik, S.: The quest for efficient boolean satisfiability solvers. In: Computer Aided Verification, Springer (2002) 17–36