Optimal Encoding of Partial Orders

Andrey Mokhov, Ulan Degenbaev, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2009-142

February 2009

Contact:

Andrey.Mokhov@ncl.ac.uk

ulan@cs.uni-sb.de

Alex.Yakovlev@ncl.ac.uk

Supported by EPSRC grants EP/C512812/1 and EP/F016786/1

NCL-EECE-MSD-TR-2009-142

School of Electrical, Electronic & Computer Engineering,
Merz Court,
University of Newcastle upon Tyne,
Newcastle upon Tyne, NE1 7RU, UK

`http://async.org.uk/`

Optimal Encoding of Partial Orders

Andrey Mokhov[†], Ulan Degenbaev[‡], Alex Yakovlev[†]

[†]Microelectronics System Design Group, Newcastle University, UK

[‡]Universität des Saarlandes, Germany

February 2009

Abstract

This paper presents a method for optimal encoding of a given set of partial orders so that a Conditional Partial Order Graph containing all of them has the least possible total count of literals in all of its vertex and arc conditions, thus leading to the smallest and fastest controller.

The presented method is implemented in a software tool for automated CPOG synthesis.

1 Introduction

Conditional Partial Order Graphs (CPOGs) introduced in [4, 3] can describe a set of partial orders in a compact graph form. In order to distinguish between partial orders within a CPOG they are given different *encodings* represented with Boolean vectors. These encodings can either be assigned arbitrary during CPOG synthesis or be provided by a designer as a part of system specification. The latter case has been studied in [4] which presented a set of optimisation techniques aiming to reduce the size of synthesised CPOG without altering the given encodings of partial orders. This paper approaches the former synthesis problem: it introduces a method for optimal encoding of a given set of partial orders in order to obtain a CPOG of minimum size.

The problem of optimal encoding can be demonstrated on the following example.

Example 1.1. Consider a processing unit that has an accumulator register A and a general purpose register B, and computes four different arithmetic functions: $(-a)$, $(a+b)$, $(a-b)$, $(-a-b)$. The event domain of the system consists of the following five events:

a) Load register A from memory;

b) Load register B from memory;

c) Compute negation $-A$ and store the result in A;

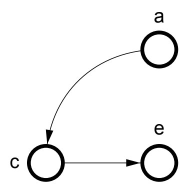
d) Compute sum $A+B$ and store the result in A;

e) Save register A into memory.

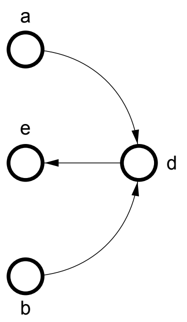
Partial orders of the four behavioural scenarios can be described with DAGs[1] shown in Figure 1. For instance, the second scenario (computation of $(a+b)$ shown in Figure 1(b)) consists of events a and b happening concurrently (loading of registers A and B), which are followed by event d (addition) and finally by event e (saving the result).

Before synthesis of a CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ containing these partial orders it is necessary to encode them, i.e. to derive a set of control signals $X = \{x_1,\ x_2,\ ...,\ x_m\}$ and a set of Boolean vectors $\{\psi_1,\ \psi_2,\ ...,\ \psi_n\}$, $\psi_k \in \{1,\ 0\}^n$ (encodings), each of them corresponding to a particular partial order.

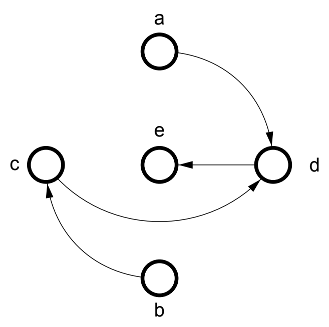
[1]Formally, a DAG $G(V,\ E)$ specifies a partial order $P(V,\ E^*)$, where $G^*(V,\ E^*)$ is the transitive closure of G [3].



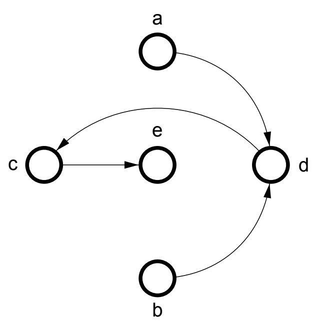
(a) Negation $(-a)$



(b) Addition $(a+b)$

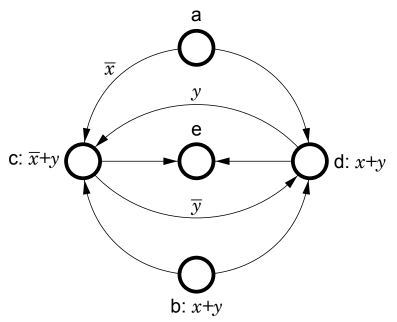


(c) Subtraction $(a-b)$

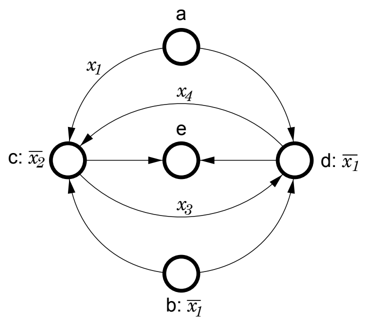


(d) Negated addition $(-a-b)$

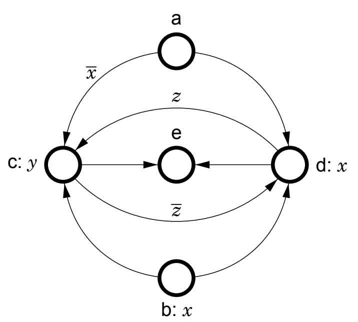
Figure 1: Four DAGs specifying the given scenarios



(a) Binary encoding



(b) One-hot encoding



(c) Optimal encoding

Figure 2: CPOGs synthesised using different encoding schemes

Note, that in this paper we consider restriction function ρ to be equal to disjunction of allowed encodings for clarity. Optimisation of ρ is a separate problem because its size does not affect the size of the specified controller, though it can be critical for verification algorithms where ρ appears as a term in SAT instances [5].

Let's examine several possible encoding schemes to see how a particular scheme affects the resultant CPOG.

Binary encoding scheme

This scheme uses the least possible number of control variables $m = \lceil \log_2 n \rceil$ to encode n given partial orders. In our example, we use two control variables $X = \{x,\ y\}$ and the encoding vectors are $\psi_1 = (00)$, $\psi_2 = (01)$, $\psi_3 = (10)$ and $\psi_4 = (11)$. Figure 2(a) shows the synthesised CPOG. It has been significantly optimised using the techniques presented in [4], the overall number of literals in vertex/arc conditions is 9.

One-hot encoding scheme

This is one of the most straightforward encoding schemes. It uses $m = n$ control variables to encode n given partial orders such that $\psi_k[k] = 1$ and $\psi_k[j] = 0,\ j \neq k$. In our example, the four one-hot encodings are: $\psi_1 = (1000)$, $\psi_2 = (0100)$, $\psi_3 = (0010)$ and $\psi_4 = (0001)$. The synthesised CPOG is shown in Figure 2(b). Note that conditions on vertices $\{b,\ c,\ d\}$ became simpler, and the total literal count is reduced to 6. The price for that is the increase in the number of control variables from 2 to 4.

Optimal encoding scheme

It turns out that there is a middle-ground solution which uses the same number of literals but only 3 control variables $X = \{x,\ y,\ z\}$. The partial orders are encoded as $\psi_1 = (010)$, $\psi_2 = (100)$, $\psi_3 = (111)$ and $\psi_4 = (110)$. The synthesised CPOG is shown in Figure 2(c). We call this encoding scheme optimal, because it is better than the binary scheme (the resultant CPOG has fewer literals which leads to a simpler controller implementation), and it is better than the one-hot scheme (it uses fewer control variables thus reducing the number of control wires coming to the controller from environment).

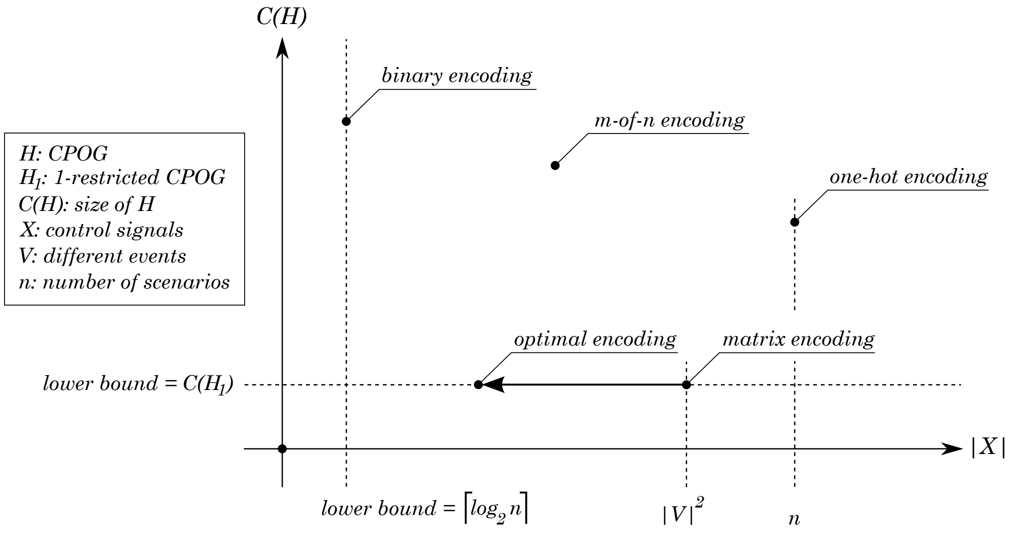


Figure 3: Size of specification vs number of control signals diagram

Figure 3 shows a Pareto diagram comparing different encoding schemes according to the number of control variables they use and the size of the resultant CPOG. One can see that the matrix encoding scheme [2] produces CPOGs with minimum possible size, but uses significantly more control variables than the binary encoding scheme. This paper aims to reduce the number of control variables but stay on the lower bound of CPOG complexity. In practice, the choice of a particular encoding scheme is crucial. The resultant CPOGs (and controllers) can differ in size by several orders of magnitude depending on the chosen encodings of partial orders [2]. The next section provides a formal definition of CPOG optimality criteria.

2 CPOG optimality criteria

A common measure of complexity of a Boolean function f is denoted as $C(f)$ and is defined to be the total count of literals in it [6], e.g. $C(x \cdot z + y \cdot \overline{z}) = 4$, $C(1) = 0$, etc[2].

A CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ is called *k-restricted* iff vertex and arc conditions ϕ are functions having at most k literals, i.e. $\forall z \in V \cup E,\ C(\phi(z)) \leq k$. Note, that this definition does not explicitly state whether H actually contains such z that $C(\phi(z)) = k$ or not. If this fact needs to be emphasised we use a stronger definition.

A *k-restricted* CPOG is called *strongly k-restricted* iff there is a vertex or an arc with condition having exactly k literals, i.e. $\exists z \in V \cup E,\ C(\phi(z)) = k$.

A 0-restricted CPOG is therefore a DAG because it does not contain any conditional vertices or arcs. A 1-restricted CPOG contains only functions with single literals (possibly inverted) or constants as its vertex and arc conditions.

[2]We use '+' and '·' to denote Boolean OR and AND operations, respectively.

Let $C(H)$ denote the total count of literals in vertex/arc conditions of a CPOG H:

$$C(H) \stackrel{\mathrm{df}}{=} \sum_{v \in V} C(\phi(v)) + \sum_{e \in E} C(\phi(e)) \tag{1}$$

$C(H)$ strongly correlates with size and speed of the physical implementation of a controller specified with H, because controllers are obtained by a direct mapping of vertex/arc conditions into logic gates. Hence, we use $C(H)$ is an adequate estimate of a CPOG H efficiency.

The following proposition states that any optimal (with respect to measure C) CPOG is bound to be 1-restricted.

Proposition 2.1. *(Optimality). For any strongly k-restricted ($k > 1$) CPOG H there exists an equivalent*[3] *1-restricted CPOG H_1 such that $C(H_1) < C(H)$.*

Proof. (Constructive). Let n be the number of partial orders contained in $H(V,\ E,\ X,\ \rho,\ \phi)$, and $\{\psi_1,\ \psi_2,\ ...,\ \psi_n\}$ be their encodings. It is possible to select a $z^+ \in V \cup E$ such that $C(\phi(z^+)) > 1$ (such z^+ must exist because H is strongly restricted).

Consider a CPOG $H'(V,\ E,\ X \cup \{x\},\ \rho',\ \phi')$ with the extended control set (a new control variable x is added), where

$$\forall z \in V \cup E,\ \phi'(z) = \begin{cases} \phi(z) & \text{if } z \neq z^+ \\ x & \text{if } z = z^+ \end{cases}$$

In other words, we replaced function $\phi(z^+)$ which had more than one literal with function $\phi'(z^+) = x$ which consists of a single literal, thus reducing the overall CPOG size by $C(\phi(z^+)) - 1 > 0$ literals.

New encodings $\{\psi'_1,\ \psi'_2,\ ...,\ \psi'_n\}$ are obtained by adding an extra element a_k (which is an assignment of x in k-th partial order) to the original encoding vectors: $\psi'_k = \psi_k \circ a_k$[4], where $a_k = \phi(z^+)|_{\psi_k}$.

The above procedure simplifies the original graph by relaxing one of the conditions, without affecting any contained partial orders. We can repeat it iteratively until the resultant graph becomes 1-restricted (let it be denoted as H_1). Note, that every iteration reduces the size by at least 1 (because $C(\phi(z^+)) - 1 > 0$), which proves the proposition. □

Although Proposition 2.1 provides a polynomial algorithm for reduction of any CPOG to a 1-restricted form, it is rather naive in terms of the resultant size of control variables set X. It adds as many new control variables as there are 'heavy' ($C(\phi(z^+)) > 1$) conditions in the original CPOG. In the worst case it uses $|V|^2$ additional variables which can be impractical. The next section introduces a method which uses the least possible number of control variables.

3 Optimal encoding and synthesis

Let $\{P_1,\ P_2,\ ...,\ P_n\}$ be a given set of n partial orders. The objective is to synthesise a 1-restricted CPOG $H(V,\ E,\ X,\ \rho,\ \phi)$ and to generate encodings $\{\psi_1,\ \psi_2,\ ...,\ \psi_n\}$ such that

$$\forall_{1 \leq k \leq n}\ \mathbf{po}(\mathbf{dg}\ H|_{\psi_k}) = P_k$$

i.e. H contains all the given partial orders as its projections. The size of control signals set $|X|$ should be minimised.

For every vertex and arc $z \in (V \cup E)$ we generate an *encoding constraint* $\mathbf{e}(z) \in \{0,\ 1,\ -\}^n$, a vector of n elements each corresponding to one of n given partial orders. Element $\mathbf{e}(z)[k]$, $1 \leq k \leq n$ is equal to 1 iff condition $\phi(z)$ should evaluate to 1 in projection $H|_{\psi_k}$ in order to produce correct P_k; $\mathbf{e}(z)[k] = 0$

[3]Two CPOGs are equivalent iff they describe the same set of partial orders [3].
[4]Operation '$\circ$' denotes concatenation of Boolean vectors, e.g. $1011 \circ 0 = 10110$.

iff the condition should evaluate to 0; and $\mathbf{e}(z)[k] = -$ iff $\phi(z)$ can evaluate either to 1 or to 0 (a *don't care* value).

Table 1 shows all the encoding constraints for the synthesis problem from Example 1.1. For instance, vertices a and e appear in all the four scenarios, so $\mathbf{e}(a) = \mathbf{e}(e) = 1111$ which means that $\phi(a) = \phi(e) = 1$. On the other hand, vertices b and d are not present in the first scenario, and therefore their encoding constraint is $\mathbf{e}(b) = \mathbf{e}(d) = 0111$. First 11 rows of the table contain *trivial encoding constraints*, i.e. constraints which do not contain $\mathbf{e}(z)[j] = 0$ and $\mathbf{e}(z)[k] = 1$ simultaneously ($j \neq k$). Vertices/arcs $z \in V \cup E$ with trivial encoding constraints can be encoded with a Boolean constant ($\phi(z) = 0$ or $\phi(z) = 1$, cf. column 'Encoding' in the table). *Non-trivial encoding constraints* (the last 5 rows of the table) cannot be satisfied with a constant value, and therefore we need to introduce control variables to encode them.

Don't care values may appear in an encoding constraint in two cases:

- An arc $e = (v \prec u)$ is not present in a partial order together with one of its vertices. In this case function $\phi(e)$ is allowed to evaluate not only to 0 but also to 1, because if one of its vertices is excluded from the partial order ($\phi(v) = 0$ or $\phi(u) = 0$) then the arc is also excluded regardless of $\phi(e)$ value (by definition of CPOGs).
- An arc $e = (v \prec u)$ is transitive (i.e. $v \prec t \prec u$ for some $t \in V$). In this case function $\phi(e)$ is allowed to evaluate not only to 1 but also to 0, because the transitive dependency $v \prec t \prec u$ is enough to guarantee $v \prec u$ ordering.

Encoding constraint $\mathbf{e}(a \prec c) = 1{-}0{-}$ combines these two cases. The first don't care is due to exclusion of vertex c in the second scenario ($\mathbf{e}(c) = 1\underline{0}11$). And the second don't care is due to transitive dependency $a \prec d \prec c$ in the fourth scenario ($\mathbf{e}(a \prec d) = -11\underline{1}$ and $\mathbf{e}(d \prec c) = --0\underline{1}$).

Vertices/arcs	Encoding constraint $\mathbf{e}(z)$				Encoding $\phi(z)$	
$z \in V \cup E$	$(-a)$	$(a+b)$	$(a-b)$	$(-a-b)$	w/o inversions	with inversions
$a,\ e$	1	1	1	1	1	1
$e \prec a$	0	0	0	0	0	0
$c \prec b$	–	–	0	0	0	0
$c \prec a,\ e \prec c$	0	–	0	0	0	0
$a \prec b,\ b \prec a,\ d \prec a,\ d \prec b\ e \prec b,\ e \prec d$	–	0	0	0	0	0
$a \prec d$	–	1	1	1	1	1
$a \prec e,\ b \prec e$	–	–	–	–	0	0
$b \prec c$	–	–	1	–	1	1
$b \prec d$	–	1	–	1	1	1
$c \prec e$	1	–	–	1	1	1
$d \prec e$	–	1	1	–	1	1
$b,\ d$	0	1	1	1	x	x
c	1	0	1	1	y	y
$a \prec c$	1	–	0	–	z	$\overline{x}$
$c \prec d$	–	–	1	0	w	z
$d \prec c$	–	–	0	1	z	$\overline{z}$

Table 1: Encoding constraints

Two encoding constraints $\mathbf{e}_1$ and $\mathbf{e}_2$ are called *conflicting* iff

$$\exists 1 \le k \le n,\ (\mathbf{e}_1[k] = 1 \wedge \mathbf{e}_2[k] = 0) \vee (\mathbf{e}_1[k] = 0 \wedge \mathbf{e}_2[k] = 1)$$

In other words, it is impossible to find a Boolean vector satisfying both constraints (up to don't cares). For example, $\mathbf{e}(a \prec c) = 1{-}0{-}$ and $\mathbf{e}(c \prec d) = {-}{-}10$ are conflicting, because $\mathbf{e}(a \prec c)[3] = 0$ and

$\mathbf{e}(c \prec d)[3] = 1$. On the other hand, $\mathbf{e}(a \prec c) = 1{-}0{-}$ and $\mathbf{e}(d \prec c) = {-}{-}01$ are not conflicting since vector 1001 satisfies both of them. It means that they both can be resolved by the same literal (literal z in Table 1, see the first sub-column of 'Encoding' column).

Optimal encoding problem can now be formulated in terms of conflicting encoding constraints: find an assignment of literals $\{x_1,\ x_2,\ ...,\ x_m\}$ to encoding constraints such that all pairs of conflicting constraints are assigned different literals and m is minimised.

Proposition 3.1. *(NP-completeness). Optimal encoding problem is NP-complete.*

Proof. We prove NP-completeness by reduction of the vertex colouring problem to the optimal encoding problem.

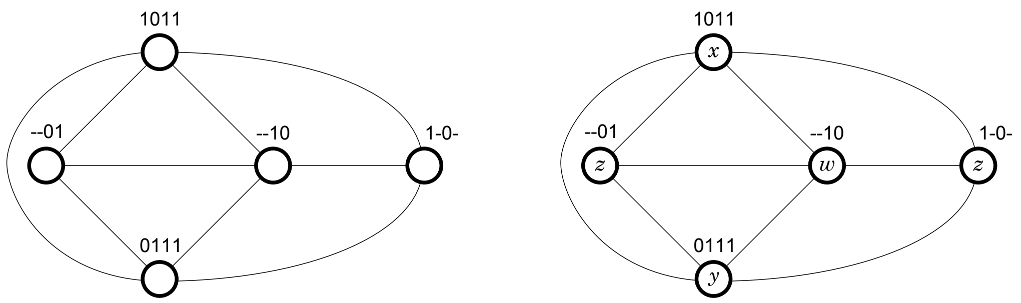
A *vertex colouring* of a DAG $G(V,\ E)$ is an assignment $clr:\ V \rightarrow \{1,\ 2,\ ...,\ m\}$ of labels to vertices such that any adjacent vertices have different labels, i.e. $\forall (x,\ y) \in E,\ clr(x) \neq clr(y)$. The problem of finding a vertex colouring with the minimum number of colours m is known to be NP-complete [1].

In order to obtain an instance of the optimal encoding problem, consider an incidence matrix M of a graph $G(V,\ E)$. The size of matrix M is $|V| \times |E|$. Thus, it has a row for each vertex and a column for each edge. An element $M_{i,j}$ indicates how a vertex v_i is incident to an arc e_j (here we assume an arbitrary numbering of vertices and edges):

$$M_{i,j} = \begin{cases} 1 & \text{if } \exists x \in V,\ (v_i,\ x) = e_j \\ 0 & \text{if } \exists x \in V,\ (x,\ v_i) = e_j \\ - & \text{otherwise} \end{cases}$$

One can see that M represents an instance of the optimal encoding problem: any two rows $i,\ j$ of M can be assigned the same literal if and only if vertices v_i and v_j are not adjacent in G. So any optimal literal assignment for M induces an optimal vertex colouring of G, and vice versa.

The described reduction uses polynomial number of steps, and, therefore, proves the NP-completeness of the optimal encoding problem. □



(a) Conflict graph (b) Optimal vertex coloring

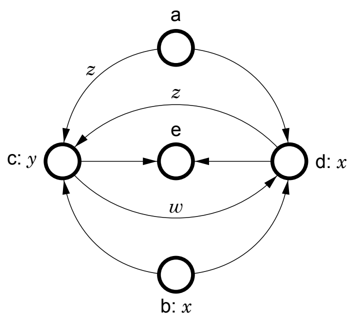
Figure 4: Conflict graph and its optimal colouring

Figure 4(a) demonstrates a conflict graph which corresponds to non-trivial encoding constraints from Table 1. One of its optimal colourings is shown in Figure 4(b) and it can also be found in the first sub-column of column 'Encoding' in Table 1 (the colouring uses 4 'colours' $\{w,\ x,\ y,\ z\}$). The synthesised CPOG is shown in Figure 5(a), the partial order encodings are $\psi_1 = (0011)$, $\psi_2 = (0100)$, $\psi_3 = (1110)$ and $\psi_4 = (0111)$.

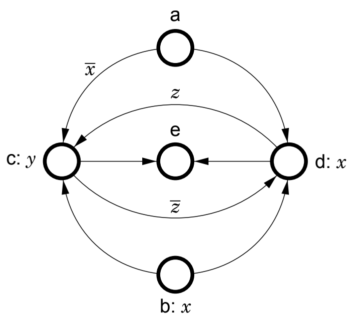
One can notice that although encoding constraints $\mathbf{e}(c \prec d) = --01$ and $\mathbf{e}(d \prec c) = --10$ are conflicting they complement each other, which opens another optimisation opportunity: it is possible to resolve both constraints with a single control variable x using its inversion. In order to exploit this, we build an *extended conflict graph* which contains two vertices for every encoding constraint $\mathbf{e}$: one for the original constraint and one for its inversion $\overline{\mathbf{e}}$ which is defined as

$$\forall 1 \le k \le n,\ \overline{\mathbf{e}}[k] = \begin{cases} 0 & \text{if } \mathbf{e}[k] = 1 \\ 1 & \text{if } \mathbf{e}[k] = 0 \\ - & \text{if } \mathbf{e}[k] = - \end{cases}$$

We need to color only one vertex among such pair. If we choose to colour the vertex corresponding to an inverted constraint with literal x it would mean that the constraint is resolved with function $\overline{x}$. See Figure 6 for the extended conflict graph of the example problem and its optimal colouring. This colouring can be found in the second sub-column of column 'Encoding' in Table 1. The synthesised CPOG is shown in Figure 5(b), the partial order encodings are $\psi_1 = (010)$, $\psi_2 = (100)$, $\psi_3 = (111)$ and $\psi_4 = (110)$.



(a) Optimal encoding (no inversions)



(b) Optimal encoding (with inversions)

Figure 5: Synthesised CPOGs

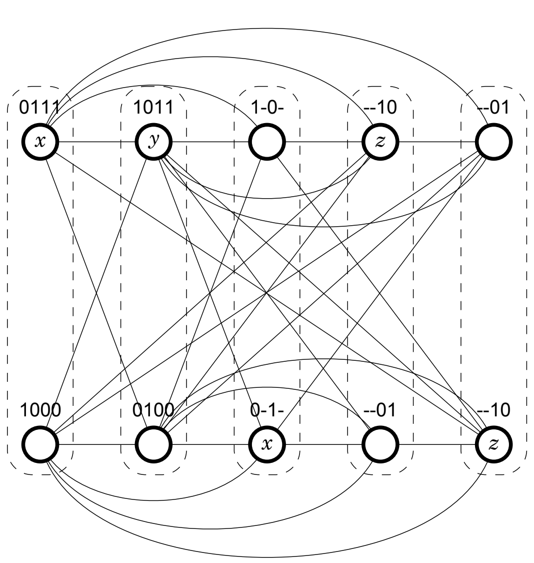


Figure 6: Extended conflict graph and its optimal colouring

Vertex colouring problem can be converted into a SAT problem instance for efficient solution.

4 Conclusions

The paper presents a motivation for the problem of optimal encoding of partial orders. It defines CPOGs optimality criteria and reduces the problem to well-studied vertex colouring problem. A software tool has been developed for automated partial orders encoding and CPOG synthesis.

Acknowledgement

This work was supported by EPSRC grants EP/C512812/1 and EP/F016786/1.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[2] Andrey Mokhov, Crescenzo D'Alessandro, and Alex Yakovlev. Synthesis of multiple rail phase encoding circuits. In *Proc. of International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC)*, 2009.

[3] Andrey Mokhov and Alex Yakovlev. Conditional partial order graphs algebra. Technical report, University of Newcastle upon Tyne, September 2008.

[4] Andrey Mokhov and Alex Yakovlev. Conditional Partial Order Graphs and Dynamically Reconfigurable Control Synthesis. In *Proceedings of Design, Automation and Test in Europe (DATE) Conference*, 2008.

[5] Andrey Mokhov and Alex Yakovlev. Verification of conditional partial order graphs. In *Proc. of 8th Int. Conf. on Applicatioon of Concurrency to System Design (ACSD'08)*, 2008.

[6] Ingo Wegener. *The Complexity of Boolean Functions*. Johann Wolfgang Goethe-Universitat, 1987.