School of Electrical, Electronic & Computer Engineering

Newcastle University

Conditional Partial Order Graphs Algebra

Andrey Mokhov, Alex Yakovlev

Technical Report Series

NCL-EECE-MSD-TR-2008-137

September 2008

Contact:

Andrey.Mokhov@ncl.ac.uk

Alex.Yakovlev@ncl.ac.uk

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

Conditional Partial Order Graphs Algebra

Andrey Mokhov, Alex Yakovlev

September 2008

Abstract

This paper introduces a formal algebra over conditional partial order graphs which were inrtoduced recently. The algebraic approach provides a set of 'safe' techniques operating on well-formed graphs without the need for verification of intermediate results.

The presented concepts provide the background for efficient synthesis and optimisation methods and form the natural development of conditional partial order graph model.

1 Introduction

Conditional partial order graphs (CPOGs) were recently introduced in [5]. It was shown that specification of a certain class of controllers is inefficient using conventional models (e.g. synthesis based on Signal Transition Graphs [8, 3], or on Finite State Machines [7]). The CPOG model has a distinctive feature of capturing similar behavioural patterns in a compact functional form as opposed to the existing models which either have a direct event traces representation or an explicit notion of states and transitions between the states.

The model was further developed in [6] to handle the dynamic control signals evaluation. The extended model revealed a strong need for verification support. SAT-based verification methods provided in [6] are computationally expensive what motivated the authors of this paper to introduce CPOG algebra closed over the set of well-formed CPOGs. The algebraic approach eliminates the need for verification in most cases, because the synthesis and optimisation can be based on operations which are proved to preserve the correctness of CPOGs.

The paper is organised as follows: Sections 2 and 3 introduce partial orders and directed acyclic graphs which constitute the basis for the CPOG model. The model itself and algebra over CPOGs is presented in Section 4. Section 5 summarises the paper.

2 Partial orders

Partial orders are widely used to formalise the intuitive concept of ordering of events with cause and effect relationships between them. This section contains mathematical definitions for the most important classes of partial orders.

2.1 Non-strict partial orders

Definition 2.1. A *non-strict* (or *weak*, or *reflexive*) *partial order* $P(S, \preceq)$ is a binary relation $\preceq$ over a set of elements S which satisfies the following three conditions [1, 4]:

1. *Reflexivity*: $\forall a \in S,\ (a \preceq a)$;
2. *Antisymmetry*: $\forall a,\ b \in S,\ (a \preceq b) \wedge (b \preceq a) \Rightarrow (a = b)$;
3. *Transitivity*: $\forall a,\ b,\ c \in S,\ (a \preceq b) \wedge (b \preceq c) \Rightarrow (a \preceq c)$.

In some contexts, the qualifier *non-strict* is omitted because it is often assumed that the partial order relation is reflexive. However, in this work the authors focus on the *irreflexive* partial orders (also called *strict*) which can represent the mutual dependencies between events in an asynchronous system more directly (see Subsection 2.2).

Example 2.1. Consider the following non-strict partial order.

Let set S consist of four events (or actions):

a) Read input value X;

b) Read input value Y;

c) Compute sum $Z = X + Y$;

d) Store the result value Z.

Now we would like to order these events taking into account the cause and effect relationships between them. One can see that action c depends on actions a and b (there is no way to compute sum Z without having values X and Y; actions a and b are so-called passive, or material causes for action c), and action d in turn cannot happen until action c is completed. Note that although d does not depend on a and b directly, it has them as indirect causes, what should also be reflected in the partial order. Such indirect dependencies are called *transitive* (see Definition 2.3).

The above dependencies can be captured with the following non-strict partial order $P(S, \preceq)$:

$$P = \begin{cases} S = \{a, \; b, \; c, \; d\} \\ \preceq = \{a \preceq a, \; b \preceq b, \; c \preceq c, \; d \preceq d, \; a \preceq c, \; b \preceq c, \; a \preceq d, \; b \preceq d, \; c \preceq d\} \end{cases}$$

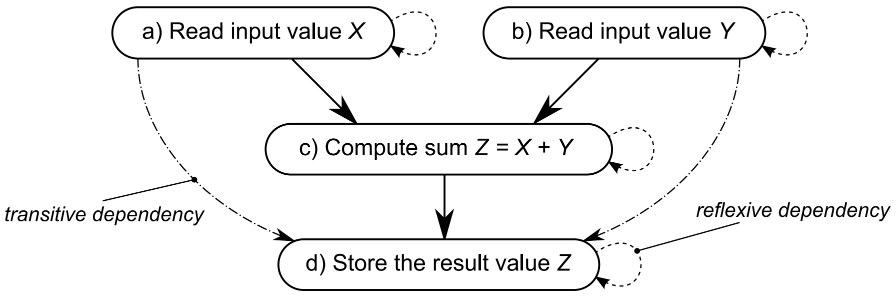


Figure 1: Non-strict partial order example

It is possible to depict the obtained partial order graphically as shown in Figure 1. The events are shown in boxes and the relationships between them – as arcs. The reflexive ones $\{a \preceq a, \; b \preceq b, \; c \preceq c, \; d \preceq d\}$ are drawn as dashed self-loops, and the transitive $\{a \preceq d, \; b \preceq d\}$ – as dash-dotted arcs.

Note, that every non-strict partial order $P(S, \preceq)$ contains $|S|$ reflexive dependencies $\forall a \in S, \; (a \preceq a)$, which do not carry any useful information about the order of the events. Moreover, for our purpose of cause and effect relationships specification, a dependency of an event on itself should be avoided. This can be done using strict partial orders defined in Subsection 2.2.

Depicting all the transitive dependencies on a partial order diagram is usually unnecessary. For example, it is possible to omit transitive arcs $\{a \prec d, \; b \prec d\}$ in the Figure 1 without losing the essential information about the depicted partial order: one can always keep the transitivity property of partial orders in mind and realise that any two relationships $a \prec b$ and $b \prec c$ $(a, \; b, \; c \in S)$ in a diagram imply the transitive relationship $a \prec c$. *Hasse diagrams* [1, 4] are widely used as a compact way of graphical representation of partial orders, as explained in Subsection 2.4.

2.2 Strict partial orders

Definition 2.2. A *strict* (or *irreflexive*) *partial order* $P(S, \prec)$ is a binary relation $\prec$ over a set of elements S which satisfies the following three conditions [1, 4]:

1. *Irreflexivity*: $\forall a \in S,\ \neg(a \prec a)$;
2. *Asymmetry*: $\forall a,\ b \in S,\ (a \prec b) \Rightarrow \neg(b \prec a)$;
3. *Transitivity*: $\forall a,\ b,\ c \in S,\ (a \prec b) \wedge (b \prec c) \Rightarrow (a \prec c)$.

Note, that the second condition (asymmetry) is redundant, because it is implied by the other two: if for some $a,\ b \in S$ both conditions $a \prec b$ and $b \prec a$ hold then transitivity leads to $(a \prec b) \wedge (b \prec a) \Rightarrow (a \prec a)$ which contradicts irreflexivity. Still, asymmetry is often considered as one of the three basic properties of strict partial orders for its importance.

Strict partial orders are irreflexive and therefore no unwanted reflexive self-dependencies are introduced into specification of cause and effect relationships between events (see Example 2.2).

Definition 2.3. Dependency (relationship, etc.) $a \prec b$ between events $a,\ b \in S$ in strict partial order $P(S, \prec)$ is called *transitive* (denoted as $a \prec\!\!\prec b$) iff there exists event $x \in S$ such that both conditions $a \prec x$ and $x \prec b$ hold:

$$(a \prec\!\!\prec b) \overset{\text{df}}{=} \exists\, x \in S,\ (a \prec x) \wedge (x \prec b)$$

Definition 2.4. Two events $a,\ b \in S$ are called *concurrent* or *parallel* (denoted as $a \parallel b$) with respect to strict partial order $P(S, \prec)$ iff they are incomparable, i.e. neither $a \prec b$ nor $b \prec a$ holds:

$$(a \parallel b) \overset{\text{df}}{=} \neg(a \prec b) \wedge \neg(b \prec a)$$

Concurrent events can happen at any time independently from each other, possibly simultaneously. Note that an event is concurrent to itself by the definition: $\forall a \in S,\ a \parallel a$.

Definition 2.5. Two events $a,\ b \in S$ are called *sequential* (denoted as $a \nparallel b$) with respect to strict partial order $P(S, \prec)$ iff they are comparable, i.e. either $a \prec b$ or $b \prec a$ holds:

$$(a \nparallel b) \overset{\text{df}}{=} (a \prec b) \vee (b \prec a) = \neg(a \parallel b)$$

Note that in an asynchronous system any two events are either concurrent or sequential and therefore they are bound to be strictly ordered: they cannot happen at exactly the same time. This is another reason that makes non-strict ordering undesirable: it implies '*less than or equal to*' relation which is not applicable to asynchronous systems (however, in a synchronous system two events can happen at exactly the same discrete moment of time – during the same clock cycle).

Example 2.2. Strict partial order helps to specify the dependencies between the events from Example 2.1 in a simpler way:

$$P = \begin{cases} S = \{a,\ b,\ c,\ d\} \\ \prec = \{a \prec c,\ b \prec c,\ a \prec d,\ b \prec d,\ c \prec d\} \end{cases}$$

Events a and b are concurrent and all the other pairs of events are sequential. Every dependency in $P(S, \prec)$ corresponds directly to a cause and effect relationship in the specified system without any self-dependencies: $\forall a \in S,\ \neg(a \prec a)$.

In the rest of the thesis the author deals only with strict partial orders and the qualifier 'strict' will be omitted for clarity.

2.3 Total orders

Definition 2.6. A *total order* is a partial order $P(S,\ \prec)$ which has an additional property called *totality* (or *comparability*, or *trichotomy*):

$$\forall a,\ b \in S,\ (a = b) \vee (a \prec b) \vee (b \prec a)$$

In other words, a total order is a partial order with no two concurrent (incomparable) events: all the events are totally ordered in one of $|S|!$ possible ways.

Definition 2.7. A *chain*, or a totally ordered subset $C \subseteq S$ of a partial order $P(S,\ \prec)$ is a set of events $C = \{a_1,\ a_2,\ ...,\ a_{|C|}\}$ such that it contains no two concurrent events: $a_k \prec a_{k+1},\ 1 \le k < |C|$. It is denoted as $a_1 \prec a_2 \prec ... \prec a_{|C|}$.

Partial order $P(S,\ \prec)$ from Example 2.2 is not total: it contains two concurrent events $a \parallel b$. Subsets $\{a,\ c,\ d\} \subset S$ and $\{b,\ c,\ d\} \subset S$, however, are totally ordered, so $a \prec c \prec d$ and $b \prec c \prec d$ are chains.

2.4 Hasse diagrams

A partial order $P(S,\ \prec)$ normally contains a lot of transitive dependencies, for instance, a chain of n events $a_1 \prec a_2 \prec ... \prec a_n,\ a_k \in S,\ k = 1...n$ implies $\binom{n-1}{2} = \frac{(n-1)(n-2)}{2} = O(n^2)$ transitive relationships $a_j \prec\!\!\prec a_k,\ 1 \le j < k - 1 < n$ ($a_1 \prec\!\!\prec a_3,\ a_1 \prec\!\!\prec a_4,\ a_2 \prec\!\!\prec a_4$ etc). So, there can be a lot more transitive dependencies than non-transitive, essential ones.

Hasse diagram [1, 4] is a graphical representation of a partial order based on *transitive reduction* [2]: it depicts only non-transitive dependencies between the events thus keeping the diagram as simple as possible. One can always reconstruct all the reduced transitive dependencies performing *transitive closure* of a Hasse diagram. Transitive reduction and closure are formally introduced in terms of graphs in Section 3. There is also a convention of arranging the events in a Hasse diagram in such a way that all the arrows point only downward or upward thus forming event levels which sometimes help understanding the depicted partial order.

Example 2.3. Hasse diagram of the strict partial order from Example 2.2 is shown in Figure 2. Note the difference from diagram in Figure 1: all the unnecessary (reflexive and transitive) dependencies are eliminated resulting in a clear and intuitively understandable form.

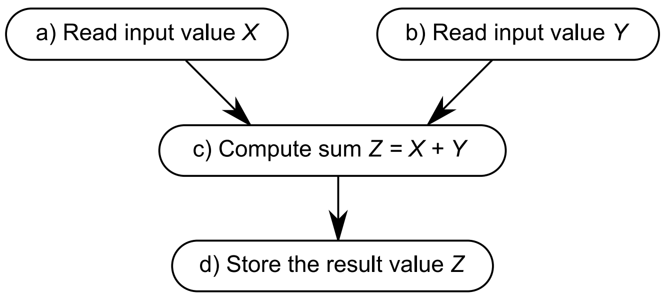


Figure 2: Hasse diagram of strict partial order

3 Directed acyclic graphs

Definition 3.1. A *directed graph* is a tuple $G(V,\ E)$ where V is a set of *vertices* (or *nodes*) and $E \subseteq V \times V$ is the set of ordered pairs of vertices, called *arcs* [2, 4].

A directed graph is typically depicted as a set of labelled circles ○ (standing for vertices) and a set of arrows $\longrightarrow$ between the circles (standing for arcs). Figure 3(a) shows an example of a directed graph $G(V,\ E)$ containing $|V| = 7$ vertices and $|E| = 6$ arcs.

A sequence of $n \geq 2$ vertices $(x_1,\ x_2,\ ...,\ x_n),\ x_k \in V,\ k = 1...n$ such that $(x_{k-1},\ x_k) \in E,\ k = 2...n$ is called a *path* from x_1 (start vertex) to x_n (end vertex) and is denoted as $\langle x_1,\ x_n\rangle$. The fact that graph G contains path $\langle x,\ y\rangle$ is denoted as $\langle x,\ y\rangle \in G$. For instance, graph G in Figure 3(a) contains paths $(c,\ d,\ f,\ g) = \langle c,\ g\rangle \in G$ and $(a,\ b) = \langle a,\ b\rangle \in G$ but does not contain path $(d,\ f,\ e) = \langle d,\ e\rangle \notin G$ because $(f,\ e) \notin E$.

A *cycle* is a path $\langle x,\ y\rangle$ whose start and end vertices coincide: $x = y$.

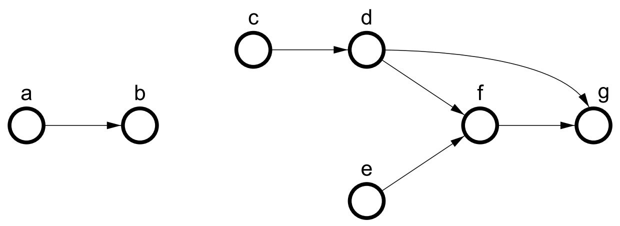
Definition 3.2. *Directed acyclic graph* (*DAG*) is a directed graph $G(V,\ E)$ that does not contain any cycles: $\forall x \in V,\ \langle x,\ x\rangle \notin G$. All the graphs in Figure 3 are DAGs.

Definition 3.3. The *transitive closure* of a graph $G(V,\ E)$ is graph $G^*(V,\ E^*)$ such that:

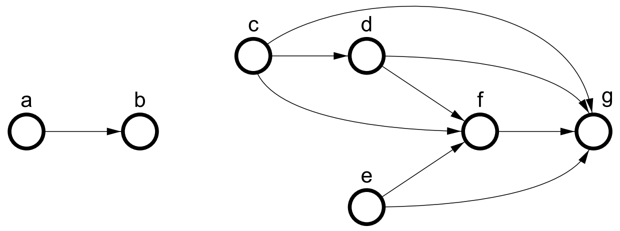
$$\forall x,\ y \in V,\ \langle x,\ y\rangle \in G \Leftrightarrow (x,\ y) \in G^*$$

In other words graph G^* contains arc $(x,\ y) \in E^*$ for every two vertices $x,\ y \in V$ that are connected with a path $\langle x,\ y\rangle \in G$ in the original graph (and vice versa). The transitive closure of graph from Figure 3(a) is shown in Figure 3(b).

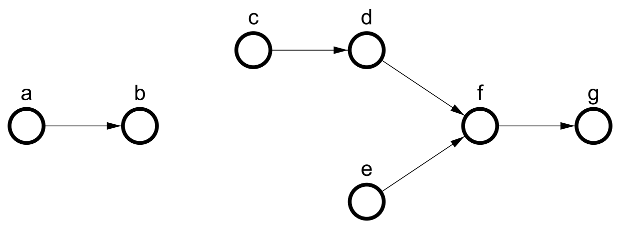
An arc $(x,\ y) \in E$ of a graph $G(V,\ E)$ is called *transitive* iff $\exists z \in V\backslash\{x,\ y\},\ \langle x,\ z\rangle \in G \wedge \langle z,\ y\rangle \in G$ i.e. there is an indirect path from x to y in the graph. Arcs $\{(c,\ f),\ (c,\ g),\ (d,\ g),\ (e,\ g)\}$ in Figure 3(b) are transitive.



(a) Directed acyclic graph $G(V,\ E)$



(b) Transitive closure $G^*(V,\ E^*)$



(c) Transitive reduction $G'(V,\ E')$

Figure 3: Directed acyclic graph, its transitive closure and transitive reduction

Definition 3.4. The *transitive reduction* of a graph $G(V,\ E)$ is the smallest (with respect to the number of arcs) graph $G'(V,\ E')$ such that:

$$\forall x,\ y \in V,\ \langle x,\ y\rangle \in G \Leftrightarrow \langle x,\ y\rangle \in G'$$

So, transitive reduction preserves all the paths in a graph but minimises the number of arcs: all the transitive arcs are reduced. Figure 3(c) shows the transitive reduction of graph from Figure 3(a): the transitive arc $(d,\ g) \in E$ has been removed.

3.1 DAGs and partial orders correspondence

There is a strong correspondence between partial orders and DAGs: every partial order is a DAG, and the transitive closure of a DAG is both a partial order and a DAG itself. The graph in Figure 3(b) directly matches a partial order relation E^* over the set of vertices V while the graph in Figure 3(a) does not because it violates the transitivity condition. For instance, it contains arcs $(e,\ f) \in E$ and $(f,\ g) \in E$ while the corresponding transitive arc is not present: $(e,\ g) \notin E$.

This correspondence between partial orders and DAGs provides an intuitive way of partial order specification. A DAG $G(V,\ E)$ defines a corresponding partial order $P(V,\ E^*)$. Note that there can be more than one DAG with the same corresponding partial order. For example, all the DAGs in Figure 3 have the same transitive closure and therefore they define the same partial order. The graph in Figure 3(c) is the simplest, however, and is preferable in most cases. This is equivalent to the approach used in Hasse diagrams (see Subsection 2.4).

Example 3.1. Figure 4 shows the four possible DAG specifications of partial order from Example 2.2. The leftmost graph is the simplest but all of them are valid. Their transitive closure is the same and is equal to the rightmost graph.

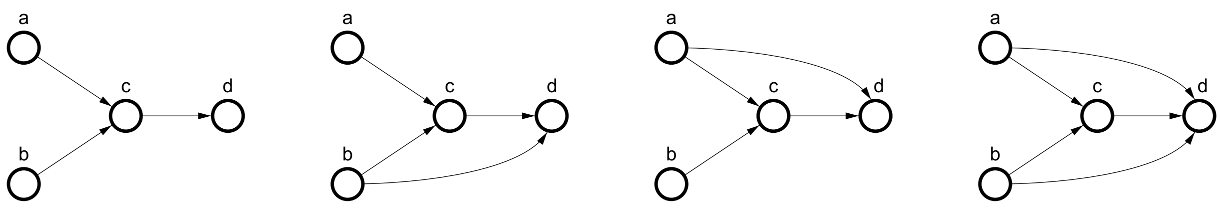


Figure 4: Possible specifications of a strict partial order using directed acyclic graphs

4 Conditional Partial Order Graphs

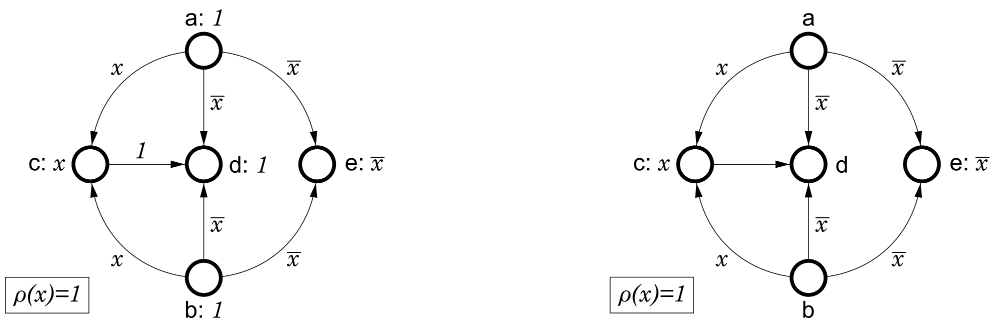
This section defines conditional partial order graphs formally and introduces algebra over them.

Definition 4.1. *Conditional partial order graph* (further called *CPOG* or *graph* for short) is a quintuple $H(V,\ E,\ X,\ \rho,\ \phi)$ where:

- V is a finite set of *vertices* which correspond to the events in the modelled system i.e. V defines the system's *event domain.*
- $E \subseteq V \times V$ is a set of ordered pairs of vertices, or *arcs*, representing the dependencies between the events.
- *Control vector* X is a finite set of Boolean variables (also called *control variables* or *signals*).
- $\rho \in \mathcal{F}(X)$ is a *restriction function,* where $\mathcal{F}(X)$ is the set of all Boolean functions over the control variables in X. ρ defines the *operational domain* of the graph: control vector X is allowed to have only those values $(x_1,\ x_2,\ ...,\ x_{|X|}) \in \{0,\ 1\}^{|X|}$ which satisfy the restriction function: $\rho(x_1,\ x_2,\ ...,\ x_{|X|}) = 1$. Graph is called *singular* iff its operational domain is empty i.e. function ρ is a contradiction: $\rho = 0$.
- Function $\phi :\ (V \cup E) \rightarrow \mathcal{F}(X)$ assigns a Boolean *condition* $\phi(z) \in \mathcal{F}(X)$ to every vertex and arc $z \in V \cup E$ in the graph. Let us also define $\phi(z) = 0$ for $z \notin V \cup E$ in order to simplify some of the further computations.

Conditional partial order graphs are represented graphically by drawing a labelled circle ○ for every vertex $v \in V$, and drawing a labelled arrow $\longrightarrow$ for every arc $e \in E$. Label of a vertex $v \in V$ consists of the vertex name, semicolon and the vertex condition $\phi(v)$, while every arc $e \in E$ is labelled with the corresponding arc condition $\phi(e)$. The restriction function ρ is depicted in a box next to the graph; control vector X can therefore be observed as the parameters of ρ.

Example 4.1. Figure 5(a) shows an example of a graph containing $|V| = 5$ vertices and $|E| = 7$ arcs. The restriction function is $\rho(x) = 1$, and the control vector consists of a single variable $X = \{x\}$. Vertices $\{a,\ b,\ d\}$ have constant $\phi = 1$ conditions and are called *unconditional*, while vertices $\{c,\ e\}$ are *conditional* and have conditions $\phi(c) = x$ and $\phi(e) = \overline{x}$ respectively. Arcs also fall into two classes: *unconditional* (arc $(c,\ d)$) and *conditional* (all the rest). As CPOGs tend to have many unconditional vertices and arcs it is reasonable to use a simplified notation in which conditions equal to 1 are not depicted in the graph. This is demonstrated in Figure 5(b).



(a) Full notation (b) Simplified notation

Figure 5: Graphical representation of conditional partial order graphs

The purpose of the vertex and arc conditions is to ‘switch off’ some of the vertices and arcs in the graph according to the control variables. This makes CPOGs capable of specifying multiple DAGs, and consequently multiple partial orders (due to the DAGs and partial orders correspondence, which was demonstrated in Subsection 3.1). Figure 6 shows an example of a graph and its two *projections* (cf. formal Definition 4.2). 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 control variable x with Boolean 1. Hence, vertex e disappears (denoted as a dashed circle ◌), because its condition evaluates to 0: $\phi(e) = \overline{x} = \overline{1} = 0$. Arcs $\{(a,\ d),\ (a,\ e),\ (b,\ d),\ (b,\ e)\}$ disappear for the same reason (denoted as dashed arrows ⇢). The rightmost projection is obtained in the same way with the only difference that the control variable x is set to 0. Note also that although the condition of arc $(c,\ d)$ evaluates to 1 (in fact it is constant 1) the arc is still excluded from the resultant graph because one of the vertices it connects (vertex c) is excluded and obviously an arc cannot appear in a graph without one of its vertices. The restriction function of the graph does not affect anything in this particular case because it evaluates to 1 for both possible control vector assignments ($x = 1$ and $x = 0$): $\rho(0) = \rho(1) = 1$. Its role will be explained in details later.

The concept of system specification with a set of partial orders contained within a single graph is clarified by the following example.

Example 4.2. Consider a processing unit that has the accumulator register A and the general purpose register B, and performs two different operations: addition and exchange of two variables stored in memory. 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) Add a value (a constant or a value from a register) to accumulator A.

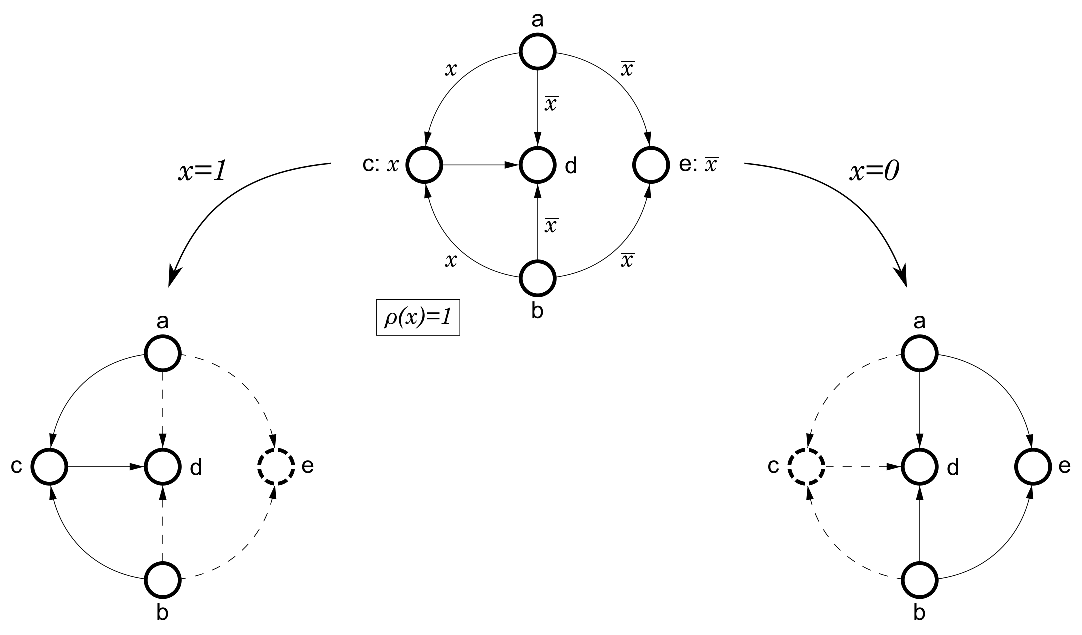


Figure 6: Multiple DAGs contained in a single CPOG

d) Save register A into memory.

e) Save register B into memory.

Table 1 contains the event descriptions of the two operations. Addition consists of loading of the two operands from memory (concurrent events a and b), their addition (event c), and saving the result (event d). This is reflected in the table with the corresponding partial order and DAG of this scenario (cf. also partial order in Example 2.2). The operation of exchange consists of loading of the operands (concurrent events a and b), and saving them into swapped memory locations (concurrent events d and e). Note, that in order to start saving one of the registers it is necessary to wait until both of them have been already loaded to avoid overwriting one of the values.

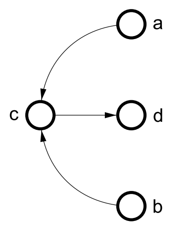
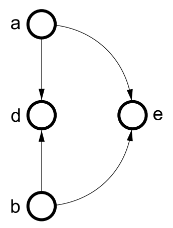
Operation		Addition	Exchange
Events description		a) Load A b) Load B c) Add B to A d) Save A	a) Load A b) Load B d) Save A e) Save B
Partial order	S	$\{a,\ b,\ c,\ d\}$	$\{a,\ b,\ d,\ e\}$
	$\prec$	$\{a \prec c,\ b \prec c,\ a \prec d,\ b \prec d,\ c \prec d\}$	$\{a \prec d,\ b \prec d,\ a \prec e,\ b \prec e\}$
DAG specification			

Table 1: Two behavioural scenarios specified as two CPOG projections

Now, one can observe that the two DAGs in Table 1 appear to be the two projections from Figure 6. Thus the both operations of the presented processing unit can be specified with the single graph.

The rest of the section contains the formal definitions of projections and algebra over CPOGs.

4.1 Projections

Definition 4.2. A *projection* of graph $H(V,\ E,\ X,\ \rho,\ \phi)$ under constraint $x = \alpha$ (where $x \in X$, $\alpha \in \{0,\ 1\}$) is denoted as $H|_{x=\alpha}$ and is equal to graph $H'(V,\ E,\ X \setminus \{x\},\ \rho|_{x=\alpha},\ \phi|_{x=\alpha})$ where notations $\rho|_{x=\alpha}$ and $\phi|_{x=\alpha}$ mean that variable x is substituted with constant Boolean value α in ρ and all functions $\phi(z),\ z \in V \cup E$, which implies that $\rho|_{x=\alpha}$ and $\phi|_{x=\alpha}(z)$ belong to $\mathcal{F}(X\setminus\{x\})$.

Projection is a *commutative operation* i.e. $(H|_{x=\alpha})|_{y=\beta} = (H|_{y=\beta})|_{x=\alpha}$ so the following short notation can be used without any ambiguity: $H|_{x=\alpha,\ y=\beta}$.

Definition 4.3. A *complete projection* of graph H is such a projection that all the variables in X are constrained to constants. It is denoted as $H|_\psi$ where $\psi :\ X \rightarrow \{0,\ 1\}$ is an *assignment function* (or *encoding*) that assigns a Boolean value to every variable in X. Complete projection is a graph whose restriction function and vertex/arc conditions are only Boolean constants $\rho|_\psi$ and $\phi|_\psi$ (either 0 or 1), and control signals set is empty: $X = \emptyset$.

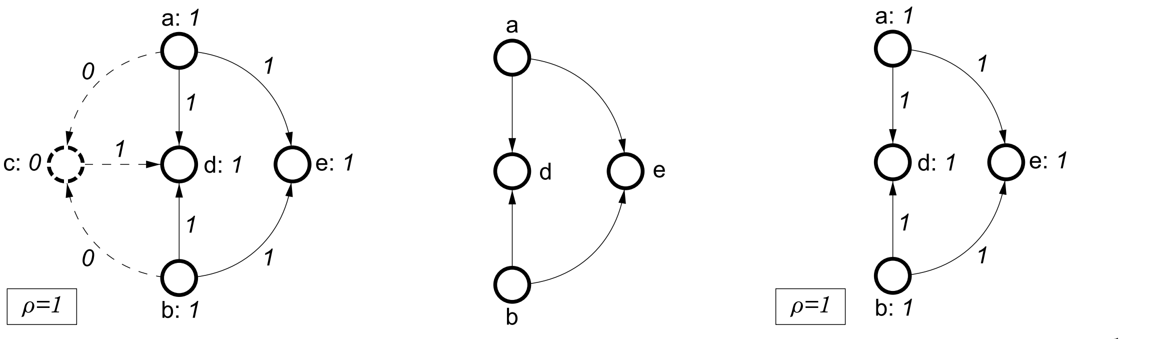
A (complete) projection is called *singular* iff the resultant graph is singular.

Definition 4.4. Given a non-singular complete projection $H(V,\ E,\ \emptyset,\ 1,\ \phi)$ operation $G = \mathbf{dg}(H)$ generates directed graph $G(V_G,\ E_G)$ such that

$$\begin{cases} V_G = \{v \in V \mid \phi(v) = 1\} \\ E_G = \{e = (a,\ b) \in E \mid \phi(a)\phi(b)\phi(e) = 1\} \end{cases}$$

In other words G includes only those vertices and arcs whose conditions in H are constant 1. Note, that exclusion of a vertex also leads to exclusion of all its adjacent arcs. Brackets around the operation argument can sometimes be omitted for clarity: $G = \mathbf{dg}\ H$.

The inverse operation is $H' = \mathbf{dg}^{-1}(G)$. Here $H'(V,\ E,\ X,\ \rho,\ \phi)$ is defined in terms of $G(V_G,\ E_G)$ as follows: $V = V_G$, $E = E_G$, $X = \emptyset$, $\rho = 1$ and $\phi(z) = 1,\ z \in V \cup E$. Note, that $\mathbf{dg}^{-1}$ is a *right inverse* operation i.e. $\mathbf{dg}(\mathbf{dg}^{-1}\ G) = G$ but $\mathbf{dg}^{-1}(\mathbf{dg}\ H)$ is not necessarily equal to H. This is demonstrated in Figure 7 which shows an example of complete projection H (Figure 7(a)), its conversion into directed graph $G = \mathbf{dg}\ H$ (Figure 7(b)), and complete projection $H' = \mathbf{dg}^{-1}\ G$ (Figure 7(c)). One can see, that $H \neq H'$ but both $\mathbf{dg}\ H$ and $\mathbf{dg}\ H'$ are the same and equal to G.



(a) Complete projection H (b) Directed graph $G = \mathbf{dg}\ H$ (c) Complete projection $H' = \mathbf{dg}^{-1}\ G$

Figure 7: Operation **dg** and its inverse

Definition 4.5. A complete projection $H|_\psi$ is called *valid* iff it is not singular and its corresponding directed graph $\mathbf{dg}\ H|_\psi$ is acyclic. Hence, an assignment function ψ is called *valid* with respect to graph H iff complete projection $H|_\psi$ is valid.

Definition 4.6. Graph $H(V,\ E,\ X,\ \rho,\ \phi)$ is *well-formed* iff its every non-singular complete projection $H|_\psi$ is valid. In other words, every complete projection $H|_\psi$ which is allowed by the restriction function $(\rho|_\psi = 1)$ must produce an acyclic directed graph $\mathbf{dg}\ H|_\psi$.

Let the set of all CPOGs be denoted as $\mathcal{C}$, and the set of all well-formed CPOGs – as $\mathcal{W}$.

Verification of well-formedness of a graph is a computationally expensive task so its use should be kept to a minimum by employing the 'safe' operations from CPOG algebra which are closed over the set of well-formed graphs $\mathcal{W}$ (see Subsections 4.2 through 4.5). In fact, it is possible to synthesise and optimise graphs without any verification of intermediate results using only the 'safe by construction' techniques. However, verification may still be required for the custom graph design/optimisation.

Definition 4.7. Given a DAG $G(V_G,\ E_G)$ operation $P = \mathbf{po}(G)$ generates partial order $P(S,\ \prec)$ such that $S = V_G$ and $\prec = E_G^*$ where $G^*(V_G,\ E_G^*)$ is the transitive closure of G.

The inverse operation is $\mathbf{po}^{-1}$: $G = \mathbf{po}^{-1}(P)$. The obtained $G(V_G,\ E_G)$ contains all the transitive arcs from $P(S,\ \prec)$: $V_G = S$, $E_G = \prec$. As was shown in Subsection 3.1, a partial order has more than one DAG specification, therefore $\mathbf{po}^{-1}$ is also a right inverse operation: $\mathbf{po}(\mathbf{po}^{-1}\ P) = P$ but $\mathbf{po}^{-1}(\mathbf{po}\ G) = G^*$ and $G \neq G^*$ in general.

Using operations $\mathbf{dg}$ and $\mathbf{po}$ it is possible to write equations operating over CPOGs, DAGs and partial orders. For example, the partial order defined by the rightmost projection of graph H in Figure 6 can be denoted as $\mathbf{po}(\mathbf{dg}\ H|_{x=0})$:

$$\mathbf{po}(\mathbf{dg}\ H|_{x=0}) = \begin{cases} S = \{a,\ b,\ d,\ e\} \\ \prec = \{a \preceq d,\ b \preceq d,\ a \preceq e,\ b \preceq e\} \end{cases}$$

Definition 4.8. The set of all partial orders defined by a well-formed graph $H(V,\ E,\ X,\ \rho,\ \phi)$ is denoted as $\mathcal{P}(H)$ and is formally defined as:

$$\mathcal{P}(H) = \{P = \mathbf{po}(\mathbf{dg}\ H|_\psi),\ \rho|_\psi = 1\}$$

For example, the set of all partial orders defined by graph in Figure 6 is

$$\mathcal{P}(H) = \{P_1,\ P_2\} = \{\mathbf{po}(\mathbf{dg}\ H|_{x=0}),\ \mathbf{po}(\mathbf{dg}\ H|_{x=1})\}$$

where P_1 and P_2 are shown in Table 1.

Note, that there is no restriction on the number of encodings for a particular partial order within a graph: there can be more than one assignment function yielding the same partial order $P = \mathbf{po}(\mathbf{dg}\ H|_{\psi_1}) = \mathbf{po}(\mathbf{dg}\ H|_{\psi_2}),\ \psi_1 \neq \psi_2$.

Definition 4.9. Two well-formed graphs H_1 and H_2 are said to be *in conflict* with respect to their restriction functions ρ_1 and ρ_2 iff $\rho_1\rho_2 \neq 0$. A conflict implies the existence of an encoding ψ such that both the restriction functions are satisfied: $\rho_1|_\psi = \rho_2|_\psi = 1$. This leads to an ambiguity in some cases (for instance, in case of graph addition introduced in Subsection 4.3), when two graphs describe different behaviour under the same encoding ψ. Depending on whether the two graphs actually specify the same or different scenarios under ψ the conflict can be either true or false.

A conflict is *true* if the scenarios encoded with ψ are different:

$$\mathbf{po}(\mathbf{dg}\ H_1|_\psi) \neq \mathbf{po}(\mathbf{dg}\ H_2|_\psi)$$

Similarly, a conflict is *false* if the scenarios encoded with ψ are in fact the same:

$$\mathbf{po}(\mathbf{dg}\ H_1|_\psi) = \mathbf{po}(\mathbf{dg}\ H_2|_\psi)$$

4.2 Equivalence

Definition 4.8 provides the background for a natural *equivalence relation* [4] $\sim$ over the set of well-formed graphs $\mathcal{W}$.

Definition 4.10. Graphs $H_1 \in \mathcal{W}$ and $H_2 \in \mathcal{W}$ are *equivalent* (denoted as $H_1 \sim H_2$) iff they define the same set of partial orders:

$$(H_1 \sim H_2) \overset{\text{df}}{=} \mathcal{P}(H_1) = \mathcal{P}(H_2)$$

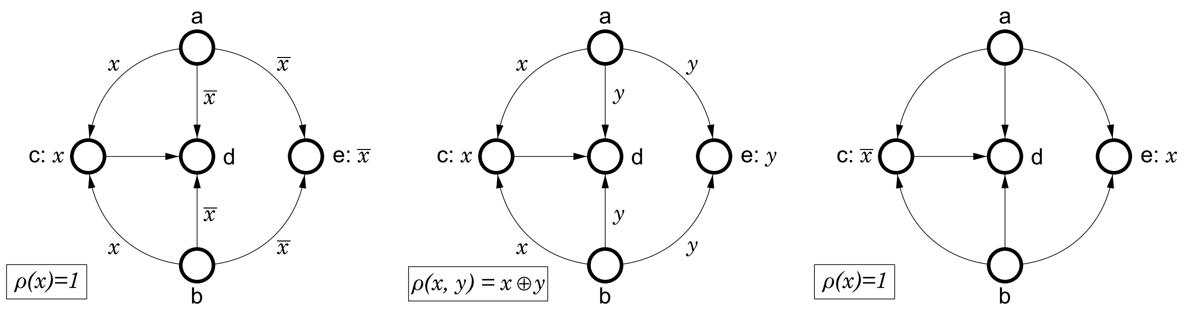
Pair $(\mathcal{W},\ \sim)$ satisfies all the required properties of an equivalence relation [4]:

- *Reflexivity*: $\forall H \in \mathcal{W},\ (H \sim H)$
- *Symmetry*: $\forall H_1,\ H_2 \in \mathcal{W},\ (H_1 \sim H_2) \Rightarrow (H_2 \sim H_1)$
- *Transitivity*: $\forall H_1,\ H_2,\ H_3 \in \mathcal{W},\ (H_1 \sim H_2) \wedge (H_2 \sim H_3) \Rightarrow (H_1 \sim H_3)$

Example 4.3. Figure 8 shows three equivalent graphs $H_a \sim H_b \sim H_c$. Graph H_a in Figure 8(a) is taken from Example 4.1. Figure 8(b) shows graph H_b with the modified control set. It contains two control variables $X = \{x,\ y\}$ which are restricted in the *one hot encoding* manner: only encodings $(0,\ 1)$ and $(1,\ 0)$ are allowed with the restriction function $\rho(x,\ y) = x \oplus y$. Graph H_c in Figure 8(c) does not contain any arc conditions (which are in fact redundant) and it also has inverted encodings compared to H_a.

In spite of the seeming difference between the three graphs, they are equivalent as they define the same set of two partial orders $\mathcal{P}(H_a) = \mathcal{P}(H_b) = \mathcal{P}(H_c) = \{P_1,\ P_2\}$:

$$\begin{cases} P_1 = \mathbf{po}(\mathbf{dg}\ H_a|_{x=0}) = \mathbf{po}(\mathbf{dg}\ H_b|_{x=0,\ y=1}) = \mathbf{po}(\mathbf{dg}\ H_c|_{x=1}) \\ P_2 = \mathbf{po}(\mathbf{dg}\ H_a|_{x=1}) = \mathbf{po}(\mathbf{dg}\ H_b|_{x=1,\ y=0}) = \mathbf{po}(\mathbf{dg}\ H_c|_{x=0}) \end{cases}$$



(a) Example graph (b) Graph with two control variables (c) No redundant conditional arcs

Figure 8: Equivalent graphs

It is useful to introduce a measure of complexity of graphs in order to be able to compare them within the same equivalence class. For instance, graph H_c in Figure 8 has the simpler description in comparison with graphs H_a and H_b and is preferred in most cases.

Definition 4.11. The *complexity* (or *size*) $C(H)$ of graph $H(V,\ E,\ X,\ \rho,\ \phi)$ is measured in the number of literals contained in the restriction function ρ and conditions $\phi(z),\ z \in V \cup E$:

$$C(H) \overset{\text{df}}{=} C(\rho) + \sum_{v \in V} C(\phi(v)) + \sum_{e \in E} C(\phi(e))$$

where $C(f),\ f \in \mathcal{F}(X)$ denotes the literal count of a Boolean function f (see [9]).

Looking at graphs in Figure 8 one can see that $C(H_a) = 0 + 2 + 6 = 8$, $C(H_b) = 2 + 2 + 6 = 10$, and $C(H_c) = 0 + 2 + 0 = 2$. So, graph H_c can be called *optimal* in this context. Methods for graphs size optimisation are addressed in [5].

4.3 Addition

Definition 4.12. The result of *addition* of two graphs $H_1(V_1,\ E_1,\ X_1,\ \rho_1,\ \phi_1)$ and $H_2(V_2,\ E_2,\ X_2,\ \rho_2,\ \phi_2)$ is graph $H(V_1 \cup V_2,\ E_1 \cup E_2,\ X_1 \cup X_2,\ \rho_1 + \rho_2,\ \phi)$ where the vertex/arc conditions ϕ are defined as

$$\forall z \in V_1 \cup V_2 \cup E_1 \cup E_2,\ \phi(z) \overset{\text{df}}{=} \rho_1\overline{\rho_2}\phi_1(z) + \overline{\rho_1}\rho_2\phi_2(z)$$

Addition is denoted using the standard notation $H = H_1 + H_2$.

Theorem 4.1. *Pair $(\mathcal{W},\ +)$ is a commutative semigroup [1] i.e. set of well-formed graphs $\mathcal{W}$ is closed under addition $+$, which is an associative and commutative operation.*

Proof. 1) Closure: $(H_1 \in \mathcal{W}) \wedge (H_2 \in \mathcal{W}) \Rightarrow (H_1 + H_2 \in \mathcal{W})$.

Let $H = H_1 + H_2$. According to Definition 4.6, graph H is well-formed iff its every non-singular complete projection $H|_\psi$ is valid.

Consider a non-singular complete projection $H|_\psi$. Non-singularity implies $\rho|_\psi = \rho_1|_\psi + \rho_2|_\psi = 1$ which is possible in one of the following three cases:

- $\rho_1|_\psi = \rho_2|_\psi = 1$ (H_1 and H_2 are in conflict with respect to encoding ψ). In this case, all the vertex and arc conditions $\phi(z)$ evaluate to zero: $\forall z,\ \phi(z)|_\psi = (\rho_1\overline{\rho_2}\phi_1(z) + \overline{\rho_1}\rho_2\phi_2(z))|_\psi = 1 \cdot \overline{1} \cdot \phi_1(z)|_\psi + \overline{1} \cdot 1 \cdot \phi_2(z)|_\psi = 0$. This projection generates an empty directed graph **dg** $H|_\psi$ which is obviously acyclic. Therefore, $H|_\psi$ is valid.

- $\rho_1|_\psi = 1$ and $\rho_2|_\psi = 0$. Here it is possible to show that **dg** $H|_\psi$ is equal to **dg** $H_1|_\psi$ and therefore $H|_\psi$ is valid due to the well-formedness of graph H_1 and validity of all its complete projections. For every $z \in V_1 \cup V_2 \cup E_1 \cup E_2$ condition $\phi(z)|_\psi$ in the complete projection $H|_\psi$ is equal to

  $$\phi(z)|_\psi = (\rho_1\overline{\rho_2}\phi_1(z) + \overline{\rho_1}\rho_2\phi_2(z))|_\psi = 1 \cdot \overline{0} \cdot \phi_1(z)|_\psi + \overline{1} \cdot 0 \cdot \phi_2(z)|_\psi = \phi_1(z)|_\psi$$

  So, **dg** $H|_\psi$ has the same set of vertices and arcs as **dg** $H_1|_\psi$, hence $H|_\psi$ is valid.

- $\rho_1|_\psi = 0$ and $\rho_2|_\psi = 1$. This case is symmetric to the previous one: $H|_\psi$ is valid because **dg** $H|_\psi =$ **dg** $H_2|_\psi$.

So, any non-singular complete projection $H|_\psi$ is valid, and thus $H = H_1 + H_2$ is well-formed.

2) Associativity: $\forall H_1,\ H_2,\ H_3 \in \mathcal{W},\ (H_1 + H_2) + H_3 = H_1 + (H_2 + H_3)$.

Follows from associativity of set union ($(V_1 \cup V_2) \cup V_3 = V_1 \cup (V_2 \cup V_3)$ etc.) and Boolean disjunction ($(\rho_1 + \rho_2) + \rho_3 = \rho_1 + (\rho_2 + \rho_3)$). To prove associativity with respect to conditions ϕ, let us define ρ' and ϕ' to be the restriction functions and conditions of graph $H' = H_1 + H_2$: $\rho' = \rho_1 + \rho_2$ and $\phi' = \rho_1\overline{\rho_2}\phi_1 + \overline{\rho_1}\rho_2\phi_2$. In the same way, let ρ and ϕ denote the restriction function and conditions of the final graph $H = H' + H_3$. So, $\rho = \rho' + \rho_3 = \rho_1 + \rho_2 + \rho_3$ while ϕ is equal to

$$\phi = \rho'\overline{\rho_3}\phi' + \overline{\rho'}\rho_3\phi_3 = (\rho_1 + \rho_2)\overline{\rho_3}(\rho_1\overline{\rho_2}\phi_1 + \overline{\rho_1}\rho_2\phi_2) + \overline{(\rho_1 + \rho_2)}\rho_3\phi_3 =$$

$$= (\rho_1 + \rho_2)(\rho_1\overline{\rho_2}\,\overline{\rho_3}\phi_1 + \overline{\rho_1}\rho_2\overline{\rho_3}\phi_2) + (\overline{\rho_1}\,\overline{\rho_2})\rho_3\phi_3 = \rho_1\overline{\rho_2}\,\overline{\rho_3}\phi_1 + \overline{\rho_1}\rho_2\overline{\rho_3}\phi_2 + \overline{\rho_1}\,\overline{\rho_2}\rho_3\phi_3$$

The result remains the same if the order of addition of the three graphs is altered: $H' = H_2 + H_3$, $H = H_1 + H'$. So, independently of the order, function $\phi(z)$ for a particular z will eventually be equal to $\rho_1\overline{\rho_2}\,\overline{\rho_3}\phi_1(z) + \overline{\rho_1}\rho_2\overline{\rho_3}\phi_2(z) + \overline{\rho_1}\,\overline{\rho_2}\rho_3\phi_3(z)$. Observe the correct scaling of the orthogonal coefficients from $\{\rho_1\overline{\rho_2},\ \overline{\rho_1}\rho_2\}$ to $\{\rho_1\overline{\rho_2}\,\overline{\rho_3},\ \overline{\rho_1}\rho_2\overline{\rho_3},\ \overline{\rho_1}\,\overline{\rho_2}\rho_3\}$.

3) Commutativity: $H_1 + H_2 = H_2 + H_1$.

Follows from the commutativity of set union ($V_1 \cup V_2 = V_2 \cup V_1$ etc.) and Boolean disjunction ($\rho_1 + \rho_2 = \rho_2 + \rho_1$ etc.) operations. □

Corollary 1. *When adding more than two graphs the redundant brackets can be omitted without any ambiguity:* $H_1 + H_2 + H_3$.

Corollary 2. *The general equation for conditions* ϕ *in graph* $H(V,\ E,\ X,\ \rho,\ \phi)$ *in case of addition of* $n \geq 2$ *graphs* $H_k(V_k,\ E_k,\ X_k,\ \rho_k,\ \phi_k),\ 1 \leq k \leq n$ *is*

$$\phi = \bigvee_{1 \leq k \leq n} (\phi_k \rho_k \bigwedge_{\substack{1 \leq j \leq n \\ j \neq k}} \overline{\rho_j})$$

e.g. if $n = 3$ *the equation is* $\phi = \rho_1 \overline{\rho_2}\,\overline{\rho_3}\phi_1 + \overline{\rho_1}\rho_2\overline{\rho_3}\phi_2 + \overline{\rho_1}\,\overline{\rho_2}\rho_3\phi_3$.

In the same way as graphs H_1 and H_2 are considered to be specifications of certain behavioural scenarios over event domains V_1 and V_2, graph $H_1 + H_2$ is considered to be specification of the scenarios from both the graphs over the joint event domain $V = V_1 \cup V_2$. This is formally stated in the following theorem.

Theorem 4.2. *If* H_1 *and* H_2 *are well-formed graphs that are not in conflict then* $\mathcal{P}(H_1) \cup \mathcal{P}(H_2) = \mathcal{P}(H_1 + H_2)$.

Proof. Let $H = H_1 + H_2$. At first let us show that $\mathcal{P}(H_1) \cup \mathcal{P}(H_2) \subseteq \mathcal{P}(H)$. Consider a partial order $P \in \mathcal{P}(H_1)$ (the proof for the case when $P \in \mathcal{P}(H_2)$ is similar due to the symmetry between H_1 and H_2). By Definition 4.8, there must exist at least one possible valid assignment function ψ such that $\rho_1|_\psi = 1$ and $P = \mathbf{po}(\mathbf{dg}\ H_1|_\psi)$. It is possible to show, that $\mathbf{dg}\ H_1|_\psi = \mathbf{dg}\ H|_\psi$ (and thus H also defines P under the same assignment function):

1. The restriction function $\rho_2|_\psi$ of H_2 is not satisfied because H_1 and H_2 are not in conflict: $(\rho_1\rho_2)|_\psi = \rho_1|_\psi \cdot \rho_2|_\psi = 1 \cdot \rho_2|_\psi = \rho_2|_\psi = 0$.

2. The restriction function ρ of H is satisfied with ψ: $\rho|_\psi = (\rho_1 + \rho_2)|_\psi = 1 + 0 = 1$.

3. Vertex/arc conditions $\phi(z)$ for $\forall z \in V_1 \cup E_1$ in $H|_\psi$ evaluate to the same values as in $H_1|_\psi$: $\phi(z)|_\psi = (\rho_1\overline{\rho_2}\phi_1(z) + \overline{\rho_1}\rho_2\phi_2(z))|_\psi = 1 \cdot \overline{0} \cdot \phi_1(z)|_\psi + \overline{1} \cdot 0 \cdot \phi_2(z)|_\psi = \phi_1(z)|_\psi$.

4. Vertex/arc conditions $\phi(z)$ for $\forall z \notin V_1 \cup E_1$ in $H|_\psi$ evaluate to 0: $\phi(z)|_\psi = \phi_1(z)|_\psi = 0$ (by Definition 4.1 of ϕ).

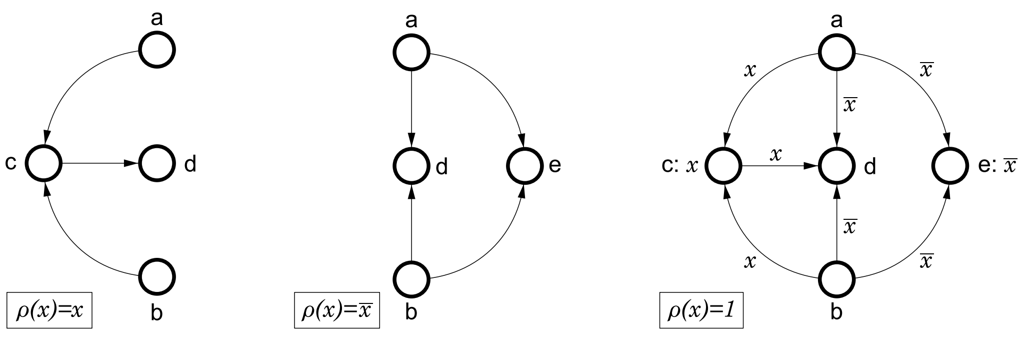
Thus sets of vertices and arcs of $\mathbf{dg}\ H|_\psi$ are the same as those of $\mathbf{dg}\ H_1|_\psi$. Consequently, $P = \mathbf{po}(\mathbf{dg}\ H_1|_\psi) = \mathbf{po}(\mathbf{dg}\ H|_\psi)$ and therefore $P \in \mathcal{P}(H)$.

Now, let us prove the reverse statement: $\mathcal{P}(H) \subseteq \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$. Consider a partial order $P \in \mathcal{P}(H)$. There must exist at least one possible valid assignment function ψ such that $P = \mathbf{po}(\mathbf{dg}\ H|_\psi)$. The restriction function $\rho = \rho_1 + \rho_2$ must be satisfied which means that either ρ_1 or ρ_2 is satisfied but not both of them. Let it be ρ_1: $\rho_1|_\psi = 1$ and $\rho_2|_\psi = 0$ (the other case is again symmetric). This leads to the same conclusion as in the first part of the proof (see points (3) and (4)): $\mathbf{dg}\ H_1|_\psi = \mathbf{dg}\ H|_\psi$. Therefore $P \in \mathcal{P}(H_1) \subseteq \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$. This completes the proof. □

Consider an example of addition in Figure 9. Each of graphs H_1 and H_2 specify a single scenario (cf. Table 1 for the details of the scenarios). The graphs are not in conflict ($\rho_1\rho_2 = x\overline{x} = 0$), the result of their addition $H_1 + H_2$ is shown in Figure 9(c). It contains both of the scenarios (as was demonstrated in Figure 6).

4.4 Scalar multiplication

Definition 4.13. Graph $H(V,\ E,\ X,\ \rho,\ \phi)$ can be *multiplied* by a Boolean function $f \in \mathcal{F}(Y)$ (which in our context can be called *scalar*). The resultant graph is $H'(V,\ E,\ X \cup Y,\ f\rho,\ \phi)$. The standard notation will be used for scalar multiplication: $H' = fH$.



(a) H_1 (b) H_2 (c) $H_1 + H_2$

Figure 9: Graph addition

Theorem 4.3. *For every Boolean function f and well-formed graph H, graph $H' = fH$ is also well-formed and $\mathcal{P}(H') \subseteq \mathcal{P}(H)$.*

Proof. Every encoding ψ which is valid with respect to H is either singular with respect to H' (when $f|_\psi = 0$) or also valid (when $f|_\psi = 1$, and conditions $\phi(z)$ in H' are the same as in H). In the latter case the partial order $P = \mathbf{po}(\mathbf{dg}\ H|_\psi)$ defined by ψ remains the same in H': $P = \mathbf{po}(\mathbf{dg}\ H'|_\psi)$. Thus function f only 'filters out' some of the partial orders defined by H by setting an additional constraint to the restriction function ρ, and no new partial orders are introduced. □

Corollary 3. *Multiplication by $f = 1$ does not change a graph: $1 \cdot H = H$ and $\mathcal{P}(1 \cdot H) = \mathcal{P}(H)$.*

Corollary 4. *Multiplication by $f = 0$ produces a singular graph: $\mathcal{P}(0 \cdot H) = \emptyset$.*

Definition 4.14. A *linear combination* of $n \geq 1$ graphs H_1, H_2, ..., H_n and scalars f_1, f_2, ..., f_n is

$$\sum_{1 \leq k \leq n} f_k H_k = f_1 H_1 + f_2 H_2 + ... + f_n H_n$$

Any linear combination of well-formed graphs is also well-formed due to the closure of addition and scalar multiplication operations over well-formed graphs (Theorems 4.1 and 4.3).

4.5 Conflict resolution

The operation of addition introduced in Subsection 4.3 produces a conservative result in case of a conflict in the added graphs. In particular, if there is a false conflict between graphs H_1 and H_2 for a particular encoding ψ the sum $H_1 + H_2$ does not contain the conflicting partial order $P = \mathbf{po}(\mathbf{dg}\ H_1|_\psi) = \mathbf{po}(\mathbf{dg}\ H_2|_\psi)$ at all, so $\mathcal{P}(H_1) \cup \mathcal{P}(H_2) \neq \mathcal{P}(H_1 + H_2)$ (cf. Theorem 4.2).

In order to be able to add graphs with true and false conflicts preserving the conflicting partial orders in the sum, the following concept of *asymmetric addition* is introduced.

Definition 4.15. The result of *asymmetric addition* of two graphs $H_1(V_1, E_1, X_1, \rho_1, \phi_1)$ and $H_2(V_2, E_2, X_2, \rho_2, \phi_2)$ is linear combination $H_1 \vec{+} H_2 \overset{\text{df}}{=} H_1 + \overline{\rho_1} H_2$. Asymmetric addition is a *left-associative* operation i.e. it is conventionally evaluated from left to right: $H_1 \vec{+} H_2 \vec{+} H_3 \overset{\text{df}}{=} (H_1 \vec{+} H_2) \vec{+} H_3$.

Asymmetric addition is closed over well-formed graphs $((H_1 \in \mathcal{W}) \wedge (H_2 \in \mathcal{W}) \Rightarrow (H_1 \vec{+} H_2 \in \mathcal{W}))$ but because of the asymmetry it is neither commutative $(H_1 \vec{+} H_2 \neq H_2 \vec{+} H_1)$ nor associative $((H_1 \vec{+} H_2) \vec{+} H_3 \neq H_1 \vec{+} (H_2 \vec{+} H_3))$ unlike normal addition.

It is possible to generalise the linear combination for asymmetric addition of more than two graphs. Let ρ' be the the restriction function of graph $(H_1 \vec{+} H_2)$: $\rho' = \rho_1 + \overline{\rho_1} \rho_2 = \rho_1 + \rho_2$. This leads to $(H_1 \vec{+} H_2) \vec{+} H_3 = (H_1 + \overline{\rho_1} H_2) \vec{+} H_3 = H_1 + \overline{\rho_1} H_2 + \overline{\rho'} H_3 = H_1 + \overline{\rho_1} H_2 + \overline{\rho_1}\,\overline{\rho_2} H_3$. The generalised linear combination for asymmetric addition of $n \geq 2$ graphs $H_k(V_k,\ E_k,\ X_k,\ \rho_k,\ \phi_k),\ 1 \leq k \leq n$ is

$$H_1 \vec{+} H_2 \vec{+} ... \vec{+} H_n = \sum_{1 \leq k \leq n} \Big(\bigwedge_{1 \leq j < k} \overline{\rho_j} \Big) H_k$$

Theorem 4.4. *If H_1 and H_2 are well-formed graphs that are not in a true conflict then $\mathcal{P}(H_1) \cup \mathcal{P}(H_2) = \mathcal{P}(H_1 \vec{+} H_2)$.*

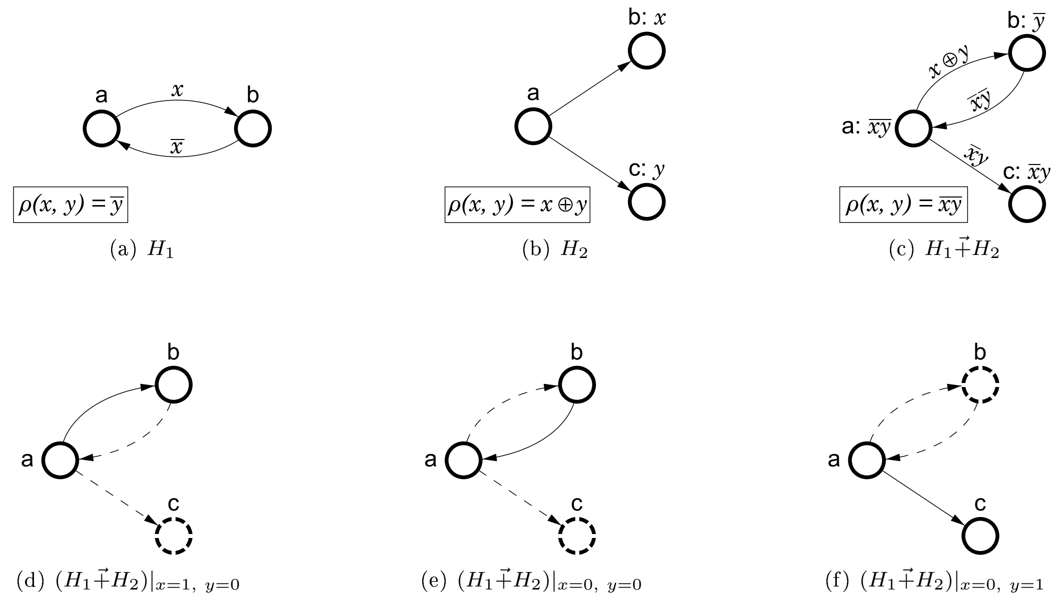
Proof. Let $H = H_1 \vec{+} H_2 = H_1 + \overline{\rho_1} H_2$. At first, notice that graphs H_1 and $\overline{\rho_1} H_2$ are not in conflict: $\rho_1(\overline{\rho_1}\rho_2) = 0$. According to Theorems 4.2 and 4.3, $\mathcal{P}(H) = \mathcal{P}(H_1 + \overline{\rho_1} H_2) = \mathcal{P}(H_1) \cup \mathcal{P}(\overline{\rho_1} H_2) \subseteq \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$.

Now, let us prove the reverse statement $\mathcal{P}(H_1) \cup \mathcal{P}(H_2) \subseteq \mathcal{P}(H)$. Any partial order $P \in \mathcal{P}(H_1)$ must belong to $\mathcal{P}(H) = \mathcal{P}(H_1 + \overline{\rho_1} H_2)$ (by Theorem 4.2). Consider a partial order $P \in \mathcal{P}(H_2)$ which has encoding ψ: $P = \mathbf{po}(\mathbf{dg}\ H_2|_\psi)$. There can be two cases with respect to the value of $\rho_1|_\psi$:

- $\rho_1|_\psi = 0$: $\mathcal{P}(\overline{\rho_1} H_2) = \mathcal{P}(1 \cdot H_2) = \mathcal{P}(H_2)$ (due to Corollary 3). So, $P \in \mathcal{P}(H_2)$ also belongs to $\mathcal{P}(\overline{\rho_1} H_2)$ and thus $P \in \mathcal{P}(H)$.
- $\rho_1|_\psi = 1$, which means that ψ is a conflicting encoding. If the conflict is false, then $P = \mathbf{po}(\mathbf{dg}\ H_1|_\psi)$ and as was already shown, any partial order from graph H_1 is included into $\mathcal{P}(H)$.

So, both $\mathcal{P}(H_1) \subseteq \mathcal{P}(H)$ and $\mathcal{P}(H_2) \subseteq \mathcal{P}(H)$ hold. Together with $\mathcal{P}(H) \subseteq \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$ this proves that $\mathcal{P}(H_1 \vec{+} H_2) = \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$. □

Corollary 5. *If well-formed graphs H_1 and H_2 have a true conflicting encoding ψ i.e. $\mathbf{po}(\mathbf{dg}\ H_1|_\psi) \neq \mathbf{po}(\mathbf{dg}\ H_2|_\psi)$ then asymmetric sum $H_1 \vec{+} H_2$ includes $\mathbf{po}(\mathbf{dg}\ H_1|_\psi)$ but not $\mathbf{po}(\mathbf{dg}\ H_2|_\psi)$.*

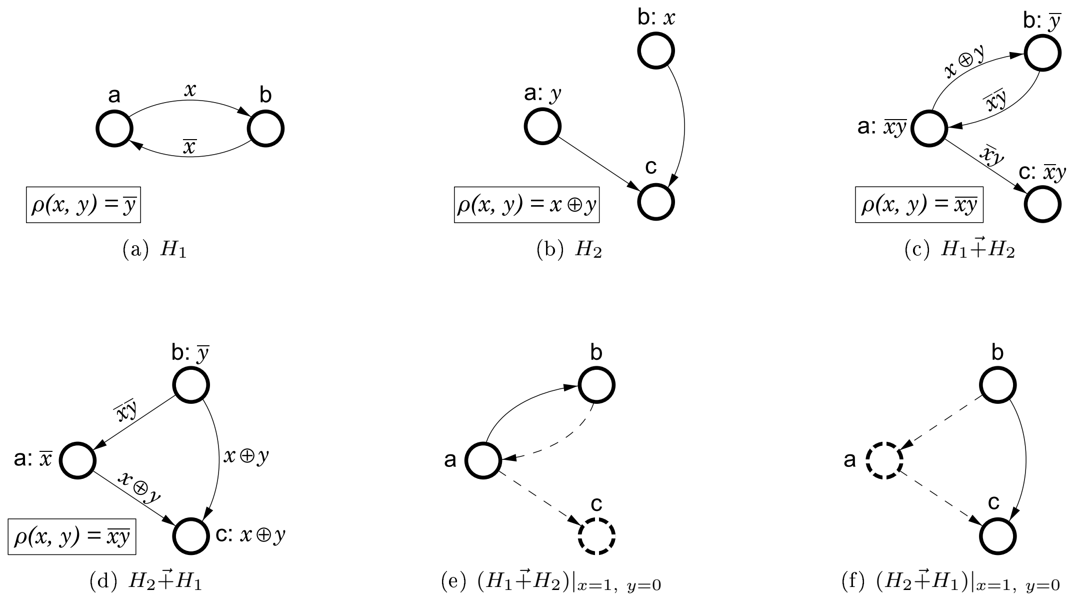


(a) H_1 (b) H_2 (c) $H_1 \vec{+} H_2$

(d) $(H_1 \vec{+} H_2)|_{x=1,\ y=0}$ (e) $(H_1 \vec{+} H_2)|_{x=0,\ y=0}$ (f) $(H_1 \vec{+} H_2)|_{x=0,\ y=1}$

Figure 10: Asymmetric addition: false conflict

Example 4.4. Consider an example of asymmetric addition of two graphs with a false conflict shown in Figure 10. Graph H_1 (Figure 10(a)) defines two simple partial orders $P_1 = \{a \prec b\} = \mathbf{po}(\mathbf{dg}\ H_1|_{x=1,\ y=0})$ and $P_2 = \{b \prec a\} = \mathbf{po}(\mathbf{dg}\ H_1|_{x=0,\ y=0})$, while graph H_2 (Figure 10(b)) defines $P_1 = \{a \prec b\} = \mathbf{po}(\mathbf{dg}\ H_2|_{x=1,\ y=0})$ and $P_3 = \{a \prec c\} = \mathbf{po}(\mathbf{dg}\ H_2|_{x=0,\ y=1})$. One can see that $\psi = (1,\ 0)$ is a conflicting encoding, but the conflict is false, because the corresponding partial orders are equal: $\mathbf{po}(\mathbf{dg}\ H_1|_{x=1,\ y=0}) = \mathbf{po}(\mathbf{dg}\ H_2|_{x=1,\ y=0}) = P_1$. Asymmetric sum $H_1 \vec{+} H_2$ shown in Figure 10(c) contains all the three partial orders: $\mathcal{P}(H_1 \vec{+} H_2) = \{P_1,\ P_2,\ P_3\} = \mathcal{P}(H_1) \cup \mathcal{P}(H_2)$. The corresponding projections are demonstrated in Figures 10(d), (e), (f).



a
x
b
$\overline{x}$
$\rho(x, y) = \overline{y}$
(a) H_1
b: x
a: y
c
$\rho(x, y) = x \oplus y$
(b) H_2
b: $\overline{y}$
$x \oplus y$
$\overline{x}\overline{y}$
a: $\overline{x}\overline{y}$
$\overline{x}y$
c: $\overline{x}y$
$\rho(x, y) = \overline{x}\overline{y}$
(c) $H_1 \vec{+} H_2$
b: $\overline{y}$
$\overline{x}y$
a: $\overline{x}$
$x \oplus y$
$x \oplus y$
c: $x \oplus y$
$\rho(x, y) = \overline{x}\overline{y}$
(d) $H_2 \vec{+} H_1$
b
a
c
(e) $(H_1 \vec{+} H_2)|_{x=1,\ y=0}$
b
a
c
(f) $(H_2 \vec{+} H_1)|_{x=1,\ y=0}$

Figure 11: Asymmetric addition: true conflict

Example 4.5. Asymmetric addition of graphs with a true conflict is demostrated in Figure 11. Graph H_1 (Figure 11(a)) defines partial orders $P_1 = \{a \prec b\} = \textbf{po}(\textbf{dg}\ H_1|_{x=1,\ y=0})$ and $P_2 = \{b \prec a\} = \textbf{po}(\textbf{dg}\ H_1|_{x=0,\ y=0})$, while graph H_2 (Figure 11(b)) defines $P_3 = \{b \prec c\} = \textbf{po}(\textbf{dg}\ H_2|_{x=1,\ y=0})$ and $P_4 = \{a \prec c\} = \textbf{po}(\textbf{dg}\ H_2|_{x=0,\ y=1})$. Conflict under $\psi = (1,\ 0)$ is true, because the corresponding partial orders are different: $P_1 = \textbf{po}(\textbf{dg}\ H_1|_{x=1,\ y=0}) \neq \textbf{po}(\textbf{dg}\ H_2|_{x=1,\ y=0}) = P_3$. Two asymmetric sums $H_1 \vec{+} H_2$ and $H_2 \vec{+} H_1$ are shown in Figures 11(c) and (d). The difference between them is due to the different conflict resolution choice: the former graph keeps partial order $P_1 = \{a \prec b\}$ while the latter keeps $P_3 = \{b \prec c\}$. This fact is demonstrated in Figures 11(e) and (f) which show the complete projections of these graphs under the conflicting encoding $(x,\ y) = (1,\ 0)$. So, the result of asymmetric sum depends significantly on the order of arguments: $\mathcal{P}(H_1 \vec{+} H_2) = \{P_1,\ P_2,\ P_4\}$, while $\mathcal{P}(H_2 \vec{+} H_1) = \{P_2,\ P_3,\ P_4\}$.

5 Conclusions

The paper presents a set of relations and operations over well-formed CPOGs: equivalence and conflict relations; addition, scalar multiplication and asymmetric addition operations. A method for flexible conflict resolution based on non-commutativity of asymmetric addition is also introduced. This provides the necessary set of tools for 'safe by construction' synthesis and optimisation.

The future work includes the incorporation of the introduced techniques into the existing CPOG software toolkit.

Acknowledgement

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

References

[1] G. Birkhoff. *Lattice Theory.* Third Edition, American Mathematical Society, Providence, RI, 1967.

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

[3] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. *Logic synthesis of asynchronous controllers and interfaces.* Advanced Microelectronics. Springer-Verlag, 2002.

[4] Art Lew. *Computer Science: A Mathematical Introduction.* Prentice-Hall, 1985.

[5] 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.

[6] 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.

[7] Steven Nowick. *Automatic Synthesis of Burst-Mode Asynchronous Controllers.* PhD thesis, Stanford University, 1993.

[8] Jens Sparsø and Steve Furber. *Principles of Asynchronous Circuit Design: A Systems Perspective.* Kluwer Academic Publishers, 2001.

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