How to Mix Concurrency and Choice and Not Explode*

Andrey Mokhov

Newcastle University, United Kingdom
andrey.mokhov@ncl.ac.uk

Abstract. This paper makes two observations. The first one may be surprising but is probably useless. The second one may someday help solve a real problem but is somewhat tedious. The reader is encouraged to take note of the former and make use of the latter.
The first observation is that *concurrency* is wrongly blamed for the dreadful state explosion problem. In fact, *choice* is a more explosive substance that should be handled with great care. Avoid mixing concurrency and choice; but if you must, read on.
The second observation is that the old Divide and Conquer strategy can be used to effectively deal with the mix of concurrency and choice. To conquer concurrency, slash it into fine diamonds. To conquer choice, wield the laws of thought. As a concrete demonstration, this paper presents a compact encoding of the Concurrent Kleene Algebra using Conditional Partial Order Graphs, where the explosion is avoided by dividing concurrency from choice and using partial orders and Boolean algebra, respectively, for their compact representation.

1 Introduction: Concurrency and Choice

Concurrent systems, which comprise multiple components that can operate and interact simultaneously, are notoriously difficult to design, control and reason about. The complexity of concurrent systems grows exponentially with the number of constituent components, and one way of coping with the complexity is to employ formal models for describing the behaviour of concurrent systems.

State graphs have been used for formal description of behaviour for many decades. However, state graphs of concurrent systems are not compact enough due to the *state space explosion problem* [1], which is illustrated in Fig. 1: as the number of concurrent events in a system increases by one, the state graph describing possible orders of their occurrence doubles in size: the state graph of a system with n concurrent events will therefore have 2^n states.

More surprisingly, state space explosion may also occur in fully sequential systems if they contain a lot of choices. In fact, as illustrated by Fig. 2, the explosion due to choice is not bounded by $O(2^n)$, but can reach $\Theta(n2^n)$ in some cases. The state graphs in Fig. 2 model systems that, given an input choice of one of 2^{n-1} possible behaviours, generate the corresponding *birmutations* on a set of n events $S = \{e_1, e_2, \cdots, e_n\}$, as defined below.

* This paper is dedicated to Prof Maciej Koutny on the occasion of his 60th birthday.

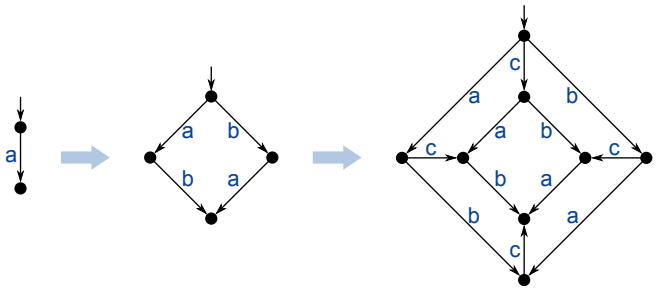

Fig. 1. State space explosion due to concurrency: 2^n states for n events.

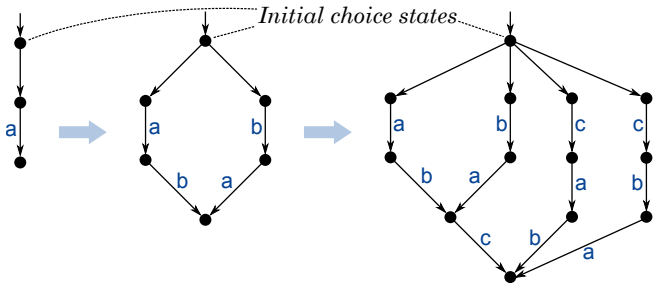

Fig. 2. State space explosion due to choice: $(n+3)2^{n-2}+1$ states for n events.

Definition 1 (Birmutations). *The set of* birmutations B_n *comprises* 2^{n-1} *event sequences that are chosen out of* $n!$ *permutations on* n *events. The chosen subset is recursively defined as follows:*

- *There is one birmutation on the singleton set* $S = \{e_1\}$*: the sequence* e_1*. That is,* $B_1 = \{e_1\}$*.*
- *When* $n > 1$*,* B_n *is obtained by inserting* e_n *either at the beginning or at the end of all birmutations* B_{n-1}*. That is,* $B_n = (e_n \circ B_{n-1}) \cup (B_{n-1} \circ e_n)$*, where* $\circ$ *denotes concatenation of an event to all sequences in a given set.*

Birmutations $B_{1\ldots4}$ are given below in the lexicographic order. We use $\{a, b, c, d\}$ to denote events instead of e_i for clarity.

- $B_1 = \{a\}$.
- $B_2 = \{ab, ba\}$.
- $B_3 = \{abc, bac, cab, cba\}$.
- $B_4 = \{abcd, bacd, cabd, cbad, dabc, dbac, dcab, dcba\}$.

A simple state graph representation of a system generating birmutations B_n would explicitly enumerate all 2^{n-1} sequences, resulting in a state graph of size $1 + (n+1)2^{n-1}$, where the leading 1 corresponds to the initial choice state, and $n+1$ states are used to model the generation of a sequence of n events.

This representation is clearly not the most compact one, because many of the 2^{n-1} sequences have common suffices which can be merged, thereby reducing the size of the state graph. For example, sequences *abcd* and *bacd* have the same suffix *cd* and can therefore be merged after the prefixes *ab* and *ba*. However, it turns out that by merging common suffices one cannot achieve any asymptotic improvement: the resulting state graph will still contain $\Theta(n2^n)$ states.

More precisely, let T_n denote the smallest size of the state graph describing birmutations B_n. Then T_n satisfies the following recurrence:

$$T_n = 2T_{n-1} + 2^{n-2} - 1.$$

Note that sequences from the set $L = B_{n-1} \circ e_n$ have no common suffices with those from the set $R = e_n \circ B_{n-1}$, and their state graphs are therefore disjoint, apart from the initial state – hence the term $2T_{n-1} - 1$. All sequences from the set L have the common suffix e_n, which is merged; all sequences from the set R have common prefix e_n, which cannot be merged, resulting in the term 2^{n-2}. The base case $T_1 = 3$, as well as the cases $T_2 = 6$ and $T_3 = 13$ are illustrated in Fig. 2. By solving the recurrence we obtain $T_n = (n+3)2^{n-2} + 1 = \Theta(n2^n)$.

As state graphs grow, humans (and at some point machines too) lose the ability to comprehend them, which motivates computer scientists to search for more compact models, such as *partial orders* [2], *event structures* [3], *Petri nets* [4], *structured occurrence nets* [5], numerous *process algebras* [6], *conditional partial order graphs* [7][8] and many others.

To deal with concurrency, a common approach is to dissect state graphs into *diamonds*, i.e. sets of independent events, and represent these diamonds compactly using partial orders. To deal with choices, one can employ *Boolean algebra* [9] to compactly describe the conditions for generating particular events. In this paper we demonstrate how these two approaches can be combined to compactly describe systems with a mix of concurrency and choice.

2 Concurrent Kleene Algebra

Concurrent Kleene Algebra (CKA) was introduced by Hoare *et al.* [10] as a unifying theory axiomatising the fundamental concepts of choice, sequential and concurrent composition, and iteration. Many models of CKA have been studied to date, e.g., see [11]. In this paper we show that the *Conditional Partial Order Graph* (CPOG) formalism [7][8] is a compact model of CKA.

In this section we briefly recap basic CKA definitions [10]. We start by introducing three common constants:

- *Bottom* $\bot$ is a contradictory specification, which permits no behaviour and can be equated to the predicate *false*.
- *Skip* 1 describes doing nothing. Note, that this is different from $\bot$; indeed, specification 1 is *implementable* (by doing nothing), while $\bot$ is impossible to implement (by its definition).

– *Top* $\top$ is the opposite of bottom in the sense that it permits any behaviour; it can be equated to the predicate *true*. Any behaviour, including doing nothing, is an admittable implementation of $\top$.

The algebra provides a rich collection of composition operators, the most important of which are listed below.

– *Sequential composition* $(p; q)$ describes the execution of both p and q, where p finishes before q starts. Sequential composition is associative, has 1 as unit and $\bot$ as zero.
– *Concurrent composition* $(p|q)$ describes the execution of both p and q, where p and q can start and finish together (but not required to do so). During the execution p and q are allowed to communicate with each other and with the environment. This operator is both associative and commutative, has 1 as unit and $\bot$ as zero.
– *Choice* $(p \cup q)$ describes the execution of either p or q. Choice is associative, commutative and idempotent, has $\bot$ as unit and $\top$ as zero. The operator is also distributive by both sequential and concurrent composition:

$$p; (q \cup r) = (p; q) \cup (p; r) \quad p|(q \cup r) = (p|q) \cup (p|r).$$

The notion of *refinement*, which is central for formal design and verification methods, is defined as a reflexive and transitive relation $p \Rightarrow q$, which holds iff p's behaviour is included in that of q. This can be equivalently expressed using the choice operator:

$$p \Rightarrow q \quad \text{iff} \quad p \cup q = q.$$

The refinement ordering has $\bot$ as the minimum (empty) behaviour and $\top$ is the maximum (any) behaviour and $\bot \Rightarrow \top$ (compare this to logical implication, where $\mathit{false} \Rightarrow \mathit{true}$).

An important law combining the above concepts is the *exchange law*, which is the most general form of so-called *concurrency reduction* that is often used when implementing a concurrent system using interleaving:

$$(p|q); (p'|q') \;\Rightarrow\; (p; p')|(q; q').$$

An interesting consequence of the exchange law is

$$p; q \;\cup\; q; p \;\Rightarrow\; p|q,$$

that is, one possible way to implement a concurrent composition is by combining choice with sequential composition.

In this work we focus on the set of operations and laws defined above. We refer the reader to [10][11], where a much broader exposition can be found.

3 Sets of Maximal Partial Orders

In this section we demonstrate that a set of maximal partial orders is a model of CKA, which is our first step towards compact models introduced in §4.

3.1 CKA without choice

Consider a partial order $P = (E, \prec)$, where $E \subseteq \mathcal{E}$, $\prec \subseteq E \times E$, and $\mathcal{E}$ is a universe of *events* that can occur in all possible behaviours. We can define some elements of CKA as follows:

- Skip 1 is the empty partial order:

$$1 \stackrel{df}{=} (\emptyset, \emptyset).$$

- Sequential composition of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1; P_2 \stackrel{df}{=} (E_1 \cup E_2, \prec_1 \cup \prec_2 \cup E_1 \times E_2).$$

  In words, assuming $E_1 \cap E_2 = \emptyset$ we schedule all events of P_1 to occur before all events of P_2.
- Concurrent composition of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1 | P_2 \stackrel{df}{=} (E_1 \cup E_2, \prec_1 \cup \prec_2).$$

  In words, assuming $E_1 \cap E_2 = \emptyset$ events of P_1 occur concurrently to events of P_2. (It is sometimes useful to consider the case when $E_1 \cap E_2 \neq \emptyset$, where the above formulation synchronises common events, thus making communication between P_1 and P_2 possible.)
- Refinement $P_1 \Rightarrow P_2$ of $P_1 = (E_1, \prec_1)$ and $P_2 = (E_2, \prec_2)$ is

$$P_1 \Rightarrow P_2 \stackrel{df}{=} E_1 = E_2 \wedge \prec_2 \subseteq \prec_1,$$

  which means that P_1 permits less concurrency than P_2.

Theorem 1. *Both* ; *and* | *as defined above are associative and have* 1 *as unit; furthermore,* | *is commutative, and the exchange law holds.*

Proof. Follows from Theorem 3.5 of [11]: (p,s,c) is replaced by $(E,\prec,(E\times E)\setminus \prec)$.

Similarly to the *trace model* [11], the *partial order model* of CKA is not powerful enough for handling the choice operator. The next subsection shows that *sets of partial orders* are sufficiently powerful.

3.2 Modelling choice

Consider a set of partial orders $S = \{P_1, P_2, ...\}$, where all constituent partial orders $P_k = (E_k, \prec_k)$ are defined on the universe of events $\mathcal{E}$, that is $E_k \subseteq \mathcal{E}$, and the set S is *downward closed*, i.e., if $P \in S$ and $P' \Rightarrow P$ then $P' \in S$. This construction is inspired by [11]. The downward closure operation will henceforth be denoted as *dc*.

We can now lift previously defined operations on partial orders to sets of partial orders, as well as introduce the missing elements of CKA, namely $\bot$, $\top$ and the choice operator:

- Bottom $\bot$ is the empty set of partial orders:

$$\bot \stackrel{df}{=} \emptyset.$$

- Skip 1 is the singleton set containing the empty partial order:

$$1 \stackrel{df}{=} \{(\emptyset, \emptyset)\}.$$

- Top $\top$ is the set containing all possible partial orders that can be defined in universe $\mathcal{E}$.
- Sequential composition of S_1 and S_2 is

$$S_1; S_2 \stackrel{df}{=} dc\left(\{\ P_1; P_2 \mid P_1 \in S_1,\ P_2 \in S_2\}\right),$$

  where $P_1; P_2$ is sequential composition of partial orders defined previously.
- Concurrent composition of S_1 and S_2 is

$$S_1 | S_2 \stackrel{df}{=} dc\left(\{\ P_1 | P_2 \mid P_1 \in S_1,\ P_2 \in S_2\}\right),$$

  where $P_1 | P_2$ is concurrent composition of partial orders defined previously.
- Choice between S_1 and S_2 is simply $S_1 \cup S_2$, where $\cup$ is the usual set union.
- Refinement can now be defined via choice as in Section 2:

$$S_1 \Rightarrow S_2 \stackrel{df}{=} S_1 \cup S_2 = S_2,$$

  which can be further simplified to the set inclusion relation:

$$S_1 \Rightarrow S_2 \stackrel{df}{=} S_1 \subseteq S_2.$$

Theorem 2. *Sets of downward closed partial orders is a model for CKA.*

Proof. (Sketch.) Here we only prove the distributivity properties of $\cup$ and the exchange law.

(1) We prove that choice $\cup$ is distributive by ; by using the definition of ; splitting set $q \cup r$ into sets q and r, and using the properties of downward closure, as shown in Fig. 3. The proof of choice distributivity by | is analogous.

(2) The exchange law is proved thanks to its *linearity*, which allows lifting the law from partial orders to downward closed sets of partial orders as explained in §3.2 and Appendix B of [11]. The rest of the laws can be verified analogously.

3.3 Reduction to maximal partial orders

As explained in the previous subsection, sets of downward closed partial orders is a model of CKA. An explicit representation of such sets, however, is very inefficient and cannot be directly used for verification or synthesis; indeed, 6.6 trillion different partial orders can be defined on just 10 events!

$$
\begin{array}{rcl}
p;(q \cup r) & = & \text{(by definition of ;)} \\
dc(\{\ a;b \mid a \in p,\ b \in q \cup r\ \}) & = & \text{(splitting set } q \cup r \text{ into } q \text{ and } r) \\
dc(\{\ a;b \mid a \in p,\ b \in q\ \} \ \cup\ \{\ a;b \mid a \in p,\ b \in r\ \}) & = & \text{(downward closure)} \\
dc(\{\ a;b \mid a \in p,\ b \in q\ \}) \ \cup\ dc(\{\ a;b \mid a \in p,\ b \in r\ \}) & & \text{(by definition of ;)} \\
(p;q) \cup (p;r). & &
\end{array}
$$

Fig. 3. Proof sketch for Theorem 2.

The first step towards compact CKA models is to notice that it is sufficient to keep only *maximal partial orders* and drop the requirement of downward closure. A partial order P is *maximal* with respect to set S, if there is no partial order $P' \in S$ such that $P \neq P'$ and $P \Rightarrow P'$. Indeed, keeping only maximal partial orders does not lead to any loss of information, because all omitted partial orders can be restored by downward closure of the remaining maximal ones.

The above is a significant improvement; however, the number of maximal partial orders is still exponential. The next section discusses two approaches that can be used for compact representation of sets of partial orders.

4 Conditional Partial Order Graphs

A CPOG is a directed graph (V, E), whose *vertices* V and *arcs* $E \subseteq V \times V$ are labelled with Boolean functions, or *conditions*, $\phi : V \cup E \to (\{0,1\}^X \to \{0,1\})$, where $\{0,1\}^X \to \{0,1\}$ is a Boolean function on a set of Boolean *variables* X.

Fig. 4 (the top left box) shows an example of a CPOG H containing 4 vertices $V = \{a, b, c, d\}$, 6 arcs and 2 variables $X = \{x, y\}$. Vertex d is labelled with condition $x + y$ (i.e. 'x OR y'), arcs (b, c) and (c, b) are labelled with conditions x and y, respectively. All other vertices and arcs have trivial conditions 1 (trivial conditions are not shown for clarity); we call such vertices and arcs *unconditional*.

There are $2^{|X|}$ possible assignments of variables X, called *codes*. Each code induces a subgraph of the CPOG, whereby all the vertices and arcs, whose conditions evaluate to 0 are removed. For example, by assigning $x = y = 0$ one obtains graph H_{00} shown in the bottom right box in Fig. 4; vertex d and arcs (b, c) and (c, b) have been removed from the graph, because their conditions are equal to 0 when $x = y = 0$. Different codes can produce different graphs, therefore a CPOG with $|X|$ variables can potentially specify a *family* of $2^{|X|}$ graphs. Fig. 4 shows two other members of the family specified by CPOG H: H_{01} and H_{10}, corresponding to codes 01 and 10, respectively, which differ only in the direction of the arc between vertices b and c. Codes will be denoted in a bold font, e.g. $\mathbf{x} = 01$, to distinguish them from vertices and variables.

It is often useful to focus only on a subset $C \subseteq \{0,1\}^X$ of codes, which are meaningful in some sense. For example, code 11 applied to CPOG H in Fig. 4 produces a graph with a loop between vertices b and c, which is undesirable if arcs are interpreted as causality. A Boolean *restriction function* $\rho : \{0,1\}^X \to \{0,1\}$ can be used to compactly specify the set $C = \{\mathbf{x} \mid \rho(\mathbf{x}) = 1\}$ and its complement

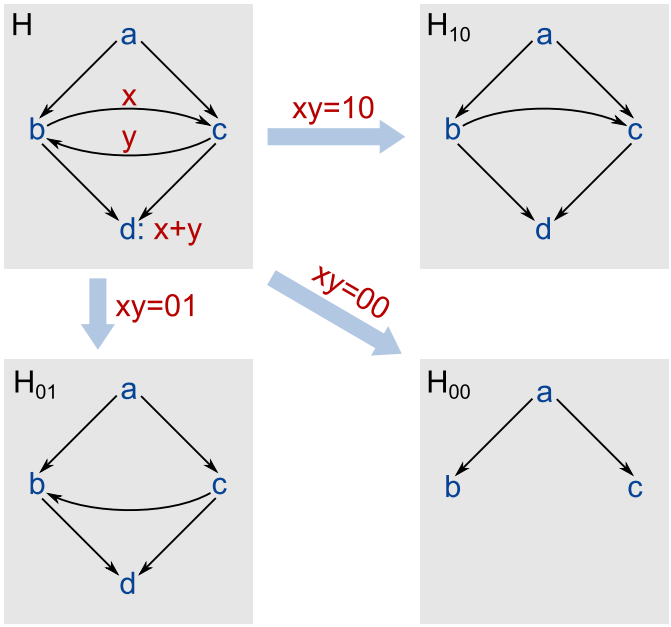

Fig. 4. A CPOG and the associated family of graphs

$DC = \{\mathbf{x} \mid \rho(\mathbf{x}) = 0\}$, which are often referred to as the *care* and *don't care* sets [12]. By setting $\rho = \overline{xy}$ one can disallow the code $\mathbf{x} = 11$ as $\rho(11) = 0$, thereby restricting the family of graphs specified by CPOG H to three members only, which are all shown in Fig. 4.

The *size* $|H|$ of a CPOG $H = (V, E, X, \phi, \rho)$ is defined as:

$$|H| = |V| + |E| + |X| + \left| \bigcup_{z \in V \cup E} \phi(z) \cup \rho \right|,$$

where $|\{f_1, f_2, \ldots, f_n\}|$ stands for the size of the smallest circuit [13] that computes all Boolean functions in the set $\{f_1, f_2, \ldots, f_n\}$.

A CPOG $H = (V, E, X, \phi, \rho)$ is *well-formed* if every allowed code $\mathbf{x}$ produces an acyclic graph $H_{\mathbf{x}}$. By computing the transitive closure $H^*_{\mathbf{x}}$ one can obtain a *strict partial order*, an irreflexive and transitive relation on the set of *events* corresponding to vertices of $H_{\mathbf{x}}$.

We can therefore interpret a well-formed CPOG as a specification of a family of partial orders. We use the term *family* instead of the more general term *set* to emphasise the fact that partial orders are *encoded*, that is each partial order $H^*_{\mathbf{x}}$ is paired with the corresponding code $\mathbf{x}$. For example, the CPOG shown in Fig. 4 specifies the family comprising the partial order H^*_{00}, where event a precedes concurrent events b and c, and two total orders H^*_{01} and H^*_{10} corresponding to sequences *acbd* and *abcd*, respectively.

It has been demonstrated in [14][15] that CPOGs are a compact model for representing families of partial orders. In particular, they can be exponentially more compact than *labelled event structures* [3] and *Petri net unfoldings* [16]. Furthermore, for some applications CPOGs provide more comprehensible models than other widely used formalisms, such as *finite state machines* and *Petri nets*, as has been shown in [7] and [17].

4.1 CPOGs as a compact model for CKA

CPOGs are capable of representing arbitrary sets of partial orders compactly and can therefore be considered a compact CKA model. Below we establish a correspondence between CPOG and CKA constructs. Bear in mind that a CPOG is a quintuple $H = (V, E, X, \rho, \phi)$.

- Bottom $\bot$ is the empty set of partial orders, which can be obtained by setting the restriction function ρ to *false*:

$$\bot \stackrel{df}{=} (\emptyset, \emptyset, \emptyset, 0, \emptyset).$$

- Skip 1 is the singleton set containing the empty partial order, which should be permitted by setting $\rho = true$:

$$1 \stackrel{df}{=} (\emptyset, \emptyset, \emptyset, 1, \emptyset).$$

- Top $\top$ is the set containing all partial orders that can be defined in universe $\mathcal{E}$:

$$\top \stackrel{df}{=} (\mathcal{E}, \mathcal{E} \times \mathcal{E}, X, 1, \phi),$$

  where X contains a variable v_z for each vertex and arc z in the graph, and $\phi(z) = v_z$. In words, top $\top$ is described by a most general CPOG that has different single-literal conditions on every graph element, which allows to obtain any possible subgraph from it by setting variables v_z accordingly. This definition demonstrate compactness of CPOGs: the size of the definition is quadratic in $\mathcal{E}$, yet it describes an exponential number of different partial orders that can be defined on the events in $\mathcal{E}$.
- Sequential composition of CPOGs $H_k = (V_k, E_k, X_k, \rho_k, \phi_k)$ for $k \in \{1, 2\}$ is

$$H_1; H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2 \cup E_1 \times E_2, X_1 \cup X_2, \rho_1 \wedge \rho_2, \phi^{seq}),$$

  where ϕ^{seq} can be defined as follows:

$$\phi^{seq}(v) = \begin{cases} \phi_1(v) & \text{if } v \in V_1, \\ \phi_2(v) & \text{if } v \in V_2. \end{cases}$$

$$\phi^{seq}(u \to v) = \begin{cases} \phi_1(u \to v) & \text{if } u \to v \in E_1, \\ \phi_2(u \to v) & \text{if } u \to v \in E_2, \\ 1 & \text{if } u \in V_1, v \in V_2, \\ 0 & \text{otherwise.} \end{cases}$$

  We assume that $V_1 \cap V_2 = \emptyset$ as before, and $X_1 \cap X_2 = \emptyset$ (the latter may be relaxed in case we want to allow H_1 and H_2 to synchronise on their *choices*).

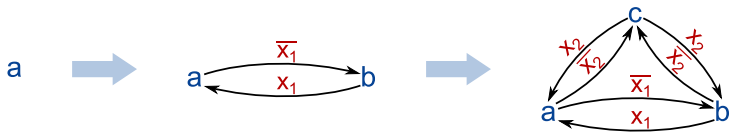

Fig. 5. CPOG models for birmutations B_1, B_2 and B_3.

– Concurrent composition of CPOGs is

$$H_1|H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2, X_1 \cup X_2, \rho_1 \wedge \rho_2, \phi^{par}),$$

where ϕ^{par} can be defined as follows:

$$\phi^{par}(v) = \begin{cases} \phi_1(v) & \text{if } v \in V_1, \\ \phi_2(v) & \text{if } v \in V_2. \end{cases}$$

$$\phi^{par}(u \to v) = \begin{cases} \phi_1(u \to v) & \text{if } u \to v \in E_1, \\ \phi_2(u \to v) & \text{if } u \to v \in E_2, \\ 0 & \text{otherwise.} \end{cases}$$

We assume that $V_1 \cap V_2 = \emptyset$ as before (unless we want to allow H_1 and H_2 to synchronise on their *events*), and $X_1 \cap X_2 = \emptyset$ (unless we want to allow H_1 and H_2 to synchronise on their *choices*).

– Choice between $H_1 = (V_1, E_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, E_2, X_2, \rho_2, \phi_2)$ is

$$H_1 \cup H_2 \stackrel{df}{=} (V_1 \cup V_2, E_1 \cup E_2, X_1 \cup X_2, \rho_1 \oplus \rho_2, \phi^{par}),$$

where $X_1 \cap X_2 = \emptyset$ (this requirement can be relaxed if $\rho_1 \wedge \rho_2 = 0$, that is H_1 and H_2 do not share any codes).

– Refinement $H_1 \Rightarrow H_2$ for $H_1 = (V_1, E_1, X_1, \rho_1, \phi_1)$ and $H_2 = (V_2, E_2, X_2, \rho_2, \phi_2)$ can be defined as

$$H_1 \Rightarrow H_2 \stackrel{df}{=} \begin{array}{c} \bigwedge_{v \in V_1} \rho_1\phi_1(v) \Rightarrow \rho_2\phi_2(v) \\ \wedge \\ \bigwedge_{u \to v \in E_2} \rho_2\phi_2(u \to v) \Rightarrow \rho_1\phi_1(u \to v). \end{array}$$

In words, if an event v appears in H_1 it must also appear in H_2, and if there is a dependency constraint $u \to v$ in H_2 then it must also exist in H_1.

The above completes the correspondence between CPOGs and CKA. Importantly, the defined constructs are very compact: they are either linear (parallel composition, choice, refinement) or quadratic (sequential composition, due to quadratic explosion of newly added arcs $E_1 \times E_2$; see [17], where this issue is resolved by using *dummy* vertices), despite potentially operating on exponentially large families of partial orders.

Fig. 5 shows compact CPOG models for birmutations: B_n is described by a CPOG with n vertices, $n^2 - n$ arcs, and $n - 1$ variables $X = \{x_1, x_2, \cdots, x_{n-1}\}$, such that each birmutation corresponds to one possible code. For example, the code of the birmutation $cab \in B_3$ is $(x_1, x_2) = 01$. The construction is a direct translation of the recursive case $B_n = (e_n \circ B_{n-1}) \cup (B_{n-1} \circ e_n)$ from Definition 1.

5 Final Remarks

The paper has demonstrated that choice can lead to larger state space explosion compared to concurrency, and showed how to avoid the state explosion by combining partial orders with Boolean algebra. As a concrete example, the paper described how CPOGs can be used as compact CKA models with potential applications in verification and synthesis of CKA specifications. Partial automation has already been implemented and reported in [14][17]. The future work includes validation of the approach on real-life case studies.

Algebra of Parameterised Graphs [17] provided an algebraic characterisation for CPOGs; the algebra satisfies the CKA laws defined in Section 2, thereby providing an alternative demonstration that CPOGs are a model for CKA.

CPOGs have been used in a number of applications, in particular for compact representation of processor instruction sets [7], synthesis of on-chip communication controllers [18], and in process mining [19], where they have shown superior performance compared to conventional approaches due to their compactness. We conjecture that CPOGs may be a generally useful CKA model that can lead to efficient verification and synthesis algorithms.

Acknowledgement

This work was conducted during a 6-month research visit to Microsoft Research Cambridge that was funded by Newcastle University, EPSRC (grant reference EP/K503885/1), and Microsoft Research. The author would like to thank Tony Hoare for his comments on an earlier draft of this paper.

References

1. A. Valmari. The state explosion problem. In *Lectures on Petri nets I: Basic models*, pages 429–528. Springer, 1998.
2. A. Mazurkiewicz. Trace theory. In *Petri nets: applications and relationships to other models of concurrency*, pages 278–324. Springer, 1987.
3. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13:85–108, 1981.
4. J. Esparza. Decidability and complexity of petri net problem – an introduction. In *Lectures on Petri Nets I: Basic Models*, pages 374–428. Springer, 1998.
5. Maciej Koutny and Brian Randell. Structured occurrence nets: A formalism for aiding system failure prevention and analysis techniques. *Fundamenta Informaticae*, 97(1-2):41–91, 2009.
6. R. Milner. *A calculus of communicating systems*, volume 92. Springer Verlag Berlin, 1980.
7. A. Mokhov. *Conditional Partial Order Graphs.* PhD thesis, Newcastle University, 2009.
8. A. Mokhov and A. Yakovlev. Conditional partial order graphs: Model, synthesis, and application. *IEEE Trans. Computers*, 59(11):1480–1493, 2010.
9. George Boole. *An investigation of the laws of thought: on which are founded the mathematical theories of logic and probabilities.* Dover Publications, 1854.

10. T. Hoare, B. Möller, G. Struth, and I. Wehrman. Concurrent Kleene Algebra and its Foundations. *The Journal of Logic and Algebraic Programming*, 80(6):266–296, 2011.
11. T. Hoare, S. van Staden, B. Möller, G. Struth, J. Villard, H. Zhu, and P. O'Hearn. Developments in concurrent kleene algebra. pages 1–18, 2014.
12. Giovanni de Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill Higher Education, 1994.
13. Ingo Wegener. *The Complexity of Boolean Functions*. Johann Wolfgang Goethe-Universitat, 1987.
14. H. Ponce de León and A. Mokhov. Building bridges between sets of partial orders. In *International Conference on Language and Automata Theory and Applications (LATA)*, 2015.
15. Hernán Ponce de León and Andrey Mokhov. Compact and efficiently verifiable models for concurrent systems. *Formal Methods in System Design*, pages 1–25.
16. K. McMillan. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*, pages 164–177. Springer, 1993.
17. A. Mokhov and V. Khomenko. Algebra of Parameterised Graphs. *ACM Transactions on Embedded Computing*, 13(4s), 2014.
18. Crescenzo D'Alessandro, Andrey Mokhov, Alex Bystrov, and Alex Yakovlev. Delay/phase regeneration circuits. In *13th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), 2007.*, pages 105–116. IEEE, 2007.
19. Andrey Mokhov, Josep Carmona, and Jonathan Beaumont. Mining conditional partial order graphs from event logs. In *Transactions on Petri Nets and Other Models of Concurrency XI*, pages 114–136. Springer, 2016.