Hindawi Publishing Corporation
Journal of Applied Mathematics
Volume 2014, Article ID 352152, 20 pages
http://dx.doi.org/10.1155/2014/352152

Research Article

Unified Mathematical Framework for Slicing and Symmetry Reduction over Event Structures

Xinyan Gao,[1] Yingcai Ding,[1] Wenbo Liu,[1] Kaidi Zheng,[1] Siyu Huang,[1] Ning Zhou,[2,3] and Dakui Li[1]

[1] *G&S Labs, School of Software, Dalian University of Technology, Dalian 116620, China*
[2] *School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China*
[3] *School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China*

Correspondence should be addressed to Dakui Li; ldk@dlut.edu.cn

Received 28 January 2014; Accepted 27 April 2014; Published 12 June 2014

Academic Editor: Xiaoyu Song

Nonclassical slicing and symmetry reduction can act as efficient structural abstract methods for pruning state space when dealing with verification problems. In this paper, we mainly address theoretical and algorithmic aspects for nonclassical slicing and symmetry reduction over prime event structures. We propose sliced and symmetric quotient reduction models of event structures and present their corresponding algorithms. To construct the underlying foundation of the proposed methodologies, we introduce strong and weak conflict concepts and a pair of mutually inverse operators and extend permutation group based symmetry notion of event structures. We have established a unified mathematical framework for slicing and symmetry reduction, and further investigated the translation, isomorphism, and equivalence relationship and other related basic facts from a theoretical point of view. The framework may provide useful guidance and theoretical exploration for overcoming verification challenges. This paper also demonstrates their practical applications by two cases.

1. Introduction

Generally, to detect whether a finite execution trace of a distributed program satisfies a given predicate, namely, predicate detection (a kind of verification problems), is a fundamental problem in asynchronous distributed systems. It has applications in many domains such as testing, debugging, and monitoring of distributed programs and it is also a powerful runtime verification method.

Unfortunately, predicate detection is NP complete [1] and suffers from the excessive size of the state space and the state explosion problem—the number of possible global states of the program increases exponentially owing to simple combination.

To deal with this problem, several useful reduction techniques have been suggested in succession for reducing the state space in recent years, such as partial order reduction and symmetric reduction methods [2–4].

On the one hand, the basic observation is that many distributed or concurrent systems exhibit a certain degree of symmetry, for example, a system composed of identical or isomorphic components whose identities are interchangeable from a verification point of view. This kind of structural symmetry in the system is also reflected in the full state space of the system. The main idea behind the symmetry reduction method is to figure out this symmetry and obtain a condensed state space which is typically much smaller than the full state space, but from which the same kind of properties of the system can be derived without unfolding the condensed state space to the full state space. Thus, it can be used to verify any property of the original model.

On the other hand, a slice of a system with respect to a criterion is a subsystem that only contains all the states of the original system that satisfy this specification. The advantage of this technique lies in the fact that the detection is performed only on the small part of the global state space which is of interest. In many cases, the slice is exponentially smaller than the original. In order to tangle predicate detection problem, nonclassical slicing technique, named computation slicing, as an abstraction mechanism, inspired by the classical program slicing of Weiser [5, 6], was first proposed by Garg and Mittal [7].

For the majority predicate classes, the computation slicing algorithm has polynomial-time complexity and gains exponential reduction of state spaces. Computation slicing has been proved to be an efficient technique for pruning state space of predicate detection in distributed computation. Moreover, it has also been successfully applied to solve the problems of temporal properties verification in transaction level hardware descriptions such as PCI local bus protocol and the MSI (modified shared invalid) cache coherence protocol [6] in SoC (system on chip) systems and so forth.

Due to the restriction of partial order execution trace model [5, 8], this approach has some limitations. Firstly, it is a runtime checking method and only checks a single partial execution trace once. It is not easy to obtain 100% path coverage even though this detection is performed multiple times. Thus, it is not suitable for exhaustive analysis by reasoning about all possible execution of the system model. Secondly, its underlying model is partially ordered set and it is not expressive enough to handle these models with explicit choice structures or conflicts. Because all the runtime traces do not contain any conflict information, it is not convenient to analyze the system under construction statically.

In this paper, we extend the notion of computation slicing from partial order traces to prime event structures with conflict. We propose a more general event structure slicing notion and a complete mathematical theoretical framework for computing the event structure slices.

The main idea is that a prime event structure can be viewed as such a system model consisting of several conflict-free substructures. These substructures themselves are in mutual weak conflict. Any of such conflict-free event substructures of a prime event structure acts as a partial order execution trace which can be sliced by traditional computation slicing algorithm. Based on this idea, we propose a partition approach to decompose a prime event structure into a group of conflict-free substructures equivalently. Each of these substructures can be sliced with respect to a given slicing criterion by the existing slicing algorithm and we can get a set of the sliced substructures. We have proved that these sliced results can be composed together and yield a new prime event structure by a so-called weak choice composition operation. We have shown that the newly generated prime event structure is the slicing result of the original prime event structure. Meanwhile, based on above partition, we can detect structural symmetry property and make symmetric reduction on each substructure of the original system. In additional, we also investigate the relationship between the symmetric reduction model and the original one.

The main contribution of our work can be summarized as follows. We introduced the slicing notion into the area ofevent structure and extended nonclassical computation slicing with conflict. We also proposed a unified mathematical framework as a common theory basis for event structure slicing and symmetry reduction. We also made a comparison between our event structure slicing and the traditional computation slicing and demonstrated the mathematical aspects of this framework.

The rest of this paper is structured as follows. Related work is discussed in Section 2. Section 3 introduces the notion of event structure and other basic definitions. Section 4 describes two core operators over event strictures. Slicing reduction derived from computation slicing will be discussed in Section 5. Symmetry reduction theory based on permutation group is reported in Section 6. The overall mathematical framework for event structure slicing and symmetry reduction will be provided in Section 7. In the last section, we make a short summary of our work.

2. Related Work

Regarding the slicing technique, the work in [5, 6] proposed classical program slice idea firstly by Weiser. Given a program and a set of variables, a program slice consists of all statements in the program that may affect the value of the variables in the set at some given point.

During years after the program slice notion was proposed, a lot of work based on this notion had been performed. For example, in 1992, the notion of a slice has been also extended to distributed programs [9]. In 2000, the notion of a nonclassical computation slice, which is very similar to the concept of a program slice, has been proposed. In work [7, 10], computation slice over partial order traces was firstly investigated by Garg and Mittal, de Bakker et al. This computation slice notion is based on partial order traces model, which is a special case of event structure without conflict.

Event structure, as an true concurrency model [11–16], can be taken as an extension of partial order model. In concurrency theory, event structures constitute a major branch of concurrent models. These were initially developed as a link between Petri nets and Scott domain theory [17] and have since been extensively applied as a semantic model for process algebras, for example [18].

All the previous work [7, 8, 19, 20] does not consider the case with conflict. Compared with them, our work is aimed to extend this slicing notion to the area of event structure.

On the other hand, as for symmetry reduction, the use of symmetry to reduce state space has been investigated widely by researchers. Technically speaking, symmetry in event structures [3, 4] is similar to symmetry in model checking [2, 21, 22]. In work [23], a category of event structures with symmetry was introduced and its categorical properties were investigated, while our work is relevant to the structural reduction via symmetry property over event structure model.

In our previous work [24], we have extended this technique to event structure area. In this paper, we will further investigate the common basis for both slicing and symmetry reduction over event structures and provide a unified framework.

3. Event Structure and Basic Definitions

In this section, we will introduce the notion of prime event structure [11, 17, 25, 26] and the basic definitions we use throughout the paper. The prime event structure is firstly defined and other related key notions are introduced. Moreover, we focus on finite prime event structures only.

Definition 1 (prime event structure). A *prime event structure* (over an alphabet $\mathscr{A}$, a set of actions) is a 4-tuple structure $(E, \preceq, \sharp, l)$ with

(i) E, a finite set of events;

(ii) $\preceq \subseteq E \times E$, a partial order, the causality relation, satisfying the principle of finite causes: for all $e \in E$: $\{e' \in E \mid e' \preceq e\}$ is finite and the inverse of $\preceq$ is denoted by $\preceq^{-1}$;

(iii) $\sharp \subseteq E \times E$, the (irreflexive and symmetric) conflict relation, satisfying the principle of conflict inheritance: $\forall d, e, f \in E : d \preceq e \wedge d \sharp f \Rightarrow e \sharp f$;

(iv) $l : E \rightarrow \mathscr{A}$, the action-labelling function.

A prime event structure (for short, an event structure) represents a system in the following way: the action names are activities which the system may perform, an event labelled $a \in \mathscr{A}$ stands for a particular occurrence of an action, $e_a \preceq e_b$ indicates that a cannot occur before b has, and $e_c \sharp e_d$ indicates that actions c and d can never occur together in one run.

The conflict inheritance property states that if an event e is in conflict with some event f, then it is in conflict with all causal successors of f.

From the causality relation, it is not difficult to derive a notion of *causal independence:*

$$e co d \Longleftrightarrow \neg \left(e = d \vee e \preceq d \vee e \preceq^{-1} d \vee e \sharp d\right). \tag{1}$$

Let $\mathbb{E}$ denote the domain of prime event structures labelled over $\mathscr{A}$ and $\emptyset = (\emptyset, \emptyset, \emptyset, \emptyset)$ stand for the empty event structure. Generally, the components of an event structure $\mathscr{E}$ will be denoted by $E_{\mathscr{E}}$, $\preceq_{\mathscr{E}}$, $\sharp_{\mathscr{E}}$, and $l_{\mathscr{E}}$, respectively. More specifically, $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}})$. If clear from the context, the index will be omitted; that is, $\mathscr{E} = (E, \preceq, \sharp, l)$ is also a valid form.

Additionally, for $X \subseteq E_{\mathscr{E}}$, the restriction of $\mathscr{E}$ to X can be defined as $\mathscr{E}|_X = (X, \preceq_{\mathscr{E}} \cap (X \times X), \sharp_{\mathscr{E}} \cap (X \times X), l_{\mathscr{E}}|_X)$. Let Succ($e$) denote all causal successors of an event e; that is, $\text{Succ}(e) = \{a \in E_{\mathscr{E}} \mid e \preceq_{\mathscr{E}} a, e \in E_{\mathscr{E}}\}$.

Definition 2 (event substructure). Let $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$ and $\mathscr{E}' = (E_{\mathscr{E}'}, \preceq_{\mathscr{E}'}, \sharp_{\mathscr{E}'}, l_{\mathscr{E}'}) \in \mathbb{E}$ be event structures; $\mathscr{E}'$ is called a *substructure* of $\mathscr{E}$ (denoted by $\mathscr{E}' \lhd \mathscr{E}$) if and only if

(i) $E_{\mathscr{E}'} \subseteq E_{\mathscr{E}}$;

(ii) for all $e, e' \in E_{\mathscr{E}}$, $e \sharp_{\mathscr{E}'} e' \Leftrightarrow e, e' \in E_{\mathscr{E}'} \wedge e \sharp_{\mathscr{E}} e'$;

(iii) for all $e, e' \in E_{\mathscr{E}}$, $e \preceq_{\mathscr{E}'} e' \Leftrightarrow e, e' \in E_{\mathscr{E}'} \wedge e \preceq_{\mathscr{E}} e'$.

Definition 3 (conflict-free event structure). An event structure $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$ is called *conflict-free event structure* (denoted by $cfES$, for short) if and only if its conflict relation is empty; that is, $\sharp_{\mathscr{E}} = \emptyset$.

Let $\mathbb{F}$ denote the domain of conflict-free prime event structures.

In order to characterize the conflict relationship between two conflict-free event structures (or substructures of a prime event structure), we introduce the following basic definitions: strong conflict, weak conflict, and weak conflict event structure set (for short, weak conflict set).

Definition 4 (strong conflict). Let $\mathscr{F}_1 = (E_{\mathscr{F}_1}, \preceq_{\mathscr{F}_1}, \emptyset, l_{\mathscr{F}_1}) \in \mathbb{F}$ and $\mathscr{F}_2 = (E_{\mathscr{F}_2}, \preceq_{\mathscr{F}_2}, \emptyset, l_{\mathscr{F}_2}) \in \mathbb{F}$. The conflict relation between E_1 ($E_1 \subseteq E_{\mathscr{F}_1}$ and $E_1 \neq \emptyset$) and E_2 ($E_2 \subseteq E_{\mathscr{F}_2}$ and $E_2 \neq \emptyset$) is called *strong conflict* if and only if for all $e \in E_1, f \in E_2 : e \sharp f$, denoted by $E_1 \sharp^s E_2$. $\mathscr{F}_1$ and $\mathscr{F}_2$ are called *strong conflict* if and only if their event sets are in mutually strong conflict, that is, for all $\mathscr{F}_1, \mathscr{F}_2 \in \mathbb{F}$: $\mathscr{F}_1 \sharp^s \mathscr{F}_2 \Leftrightarrow E_{\mathscr{F}_1} \sharp^s E_{\mathscr{F}_2}$, denoted by $\mathscr{F}_1 \sharp^s \mathscr{F}_2$.

More generally, for any $\mathscr{E}_1 \in \mathbb{E}$ and $\mathscr{E}_2 \in \mathbb{E}$, the relation between nonempty E_1' ($\emptyset \neq E_1' \subseteq E_{\mathscr{E}_1}$) and E_2' ($\emptyset \neq E_2' \subseteq E_{\mathscr{E}_2}$) is called *extended strong conflict* if and only if for all $e \in E_1', f \in E_2' : e \sharp f$, denoted by $E_1' \sharp^{xs} E_2'$. That is, each of E_1' is in conflict with each of E_2' and the existence of conflict relation in E_1' or E_2' is allowed.

Definition 5 (weak conflict). Let $\mathscr{F}_1 = (E_{\mathscr{F}_1}, \preceq_{\mathscr{F}_1}, \emptyset, l_{\mathscr{F}_1}) \in \mathbb{F}$ and $\mathscr{F}_2 = (E_{\mathscr{F}_2}, \preceq_{\mathscr{F}_2}, \emptyset, l_{\mathscr{F}_2}) \in \mathbb{F}$. The conflict relation between event sets E_1 ($E_1 \subseteq E_{\mathscr{F}_1}$ and $E_1 \neq \emptyset$) and E_2 ($E_2 \subseteq E_{\mathscr{F}_2}$ and $E_2 \neq \emptyset$) is called *weak conflict* if and only if $\exists e \in E_1, \exists f \in E_2 : e \sharp f$, denoted by $E_1 \sharp^w E_2$. The conflict-free event structures, $\mathscr{F}_1$ and $\mathscr{F}_2$, are called *weak conflict* if and only if their event sets are in weak conflict; that is, for all $\mathscr{F}_1, \mathscr{F}_2 \in \mathbb{F} : \mathscr{F}_1 \sharp^w \mathscr{F}_2 \Leftrightarrow E_{\mathscr{F}_1} \sharp^w E_{\mathscr{F}_2}$, denoted by $\mathscr{F}_1 \sharp^w \mathscr{F}_2$.

Stated in words, it is not that each event of $E_{\mathscr{F}_1}$ is in conflict with each event of $E_{\mathscr{F}_2}$, but there exists at least one conflicting event pair between $E_{\mathscr{F}_1}$ and $E_{\mathscr{F}_2}$.

Basically, according to the previous definitions, strong conflict relation is a special case of weak conflict relation.

Definition 6 (weak conflict set). Let $\mathscr{WF}_n^{cfw} = \{\mathscr{F}_i \in \mathbb{F} \mid i, n \in \mathbf{N}, \mathbf{1} \leq \mathbf{i} \leq \mathbf{n}\}$ over event set $E_{\mathscr{WF}}$, $\mathscr{WF}_n^{cfw}$ is called a *weak conflict set* if and only if $E_{\mathscr{WF}} = \bigcup_{i=1}^{n} E_{\mathscr{F}_i}$ and for all $\mathscr{F}_i, \mathscr{F}_j \in \mathscr{WF}_n^{cfw} : i \neq j \Rightarrow \mathscr{F}_i \sharp^w \mathscr{F}_j$ ($i, j \in N, 1 \leq i \leq n$).

For convenience, let $wfsetF(\mathscr{WF}_n^{cfw}) = \{E_{\mathscr{F}_i} \mid i \in \mathbf{N}, 1 \leq i \leq n\}$ denote the family of weak conflict event sets.

Definition 7 (maximal conflict-free event substructure). Let $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$ be an event structure; any event subset $E' \subseteq E_{\mathscr{E}}$ is called a *maximal conflict-free event subset* (for short, mcfset) of $E_{\mathscr{E}}$ if and only if it satisfies the following:

(1) for all $e, f \in E' : \neg(e \sharp f)$;

(2) for all $c \in (E_{\mathscr{E}} - E') : \exists d \in E' \Rightarrow c \sharp d$.

Its corresponding substructure $\mathscr{E}'$ is called *maximal conflict-free event substructure* of $\mathscr{E}$; that is, $\mathscr{E}' = (E', \preceq_{\mathscr{E}} \cap (E' \times E'), \emptyset, l_{E'})$.

4. Operators over Event Structure

In this section, a pair of mutually inverse operators, cfp (conflict-free partition) and wcc (weak conflict composition), will be introduced and discussed. For any prime event structure $\mathscr{E}$, partition and composition operation over it can be associated via its family of configurations.

4.1. Maximal Conflict-Free Partition. In fact, a prime event structure can be viewed as a system consisting of several substructures, which are conflict-free themselves. Such a conflict-free event substructure of a prime event structure represents a specific possible partial order execution trace via branching or nondeterministic choices. For any prime event structure, it is a certainty that we can get its maximal conflict-free substructures by some kind of conflict-free partition operation according to the characteristics of its conflict relation.

First of all, we give the definition of maximal conflict pattern for an event structure. The notion of maximal conflict pattern can make great contributions to accelerate the process of partition by avoiding unnecessary partition steps. We then provide the key partition algorithm for a prime event structure.

Definition 8 (maximal conflict pattern). Let $\mathscr{E} = (E_{\mathscr{E}}, \leq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$; for any $A \subseteq E_{\mathscr{E}}$ and $B \subseteq E_{\mathscr{E}}$, $A\sharp^{xs}B$ is called a *maximal conflict pattern*if and only if for all $A' \subseteq E_{\mathscr{E}}$, for all $B' \subseteq E_{\mathscr{E}} : (A \subset A' \land B \subset B') \Rightarrow \neg(A'\sharp^{xs}B')$.

For any prime event structure, we can get these maximal conflict patterns by the following two steps:

(1) casual successors expanding;

(2) conflict pairs merging.

Firstly, due to the conflict inheritance property, we know that if event e is in conflict with event f then their casual successors are also in mutual conflict; that is, $\forall c \in \text{Succ}(e), \forall d \in \text{Succ}(f) : e\sharp f \Rightarrow c\sharp d$.

Let $\{e\}_\sharp = \{e\} \cup \text{Succ}(e)$ and $\{f\}_\sharp = \{f\} \cup \text{Succ}(f)$; we have that $\{e\}_\sharp$ and $\{f\}_\sharp$ are in strong conflict if $e\sharp_{\mathscr{E}} f$; namely, $\forall e, f \in E_{\mathscr{E}} : e\sharp_{\mathscr{E}} f \Rightarrow \{e\}_\sharp \sharp^s \{f\}_\sharp$.

For example, for a prime event structure $\mathscr{E}$, if $c\sharp_{\mathscr{E}} d$ and casual relations are $c \preceq_{\mathscr{E}} e_1$, $c \preceq_{\mathscr{E}} e_2$, $d \preceq_{\mathscr{E}} f_1$, $f_1 \preceq_{\mathscr{E}} f_2$, and $d \preceq_{\mathscr{E}} f_3$, we then have $\{c, e_1, e_2,\}\sharp^s\{d, f_1, f_2, f_3\}$.

We also have that any nonempty subset of $\{e\}_\sharp$ and any nonempty subset of $\{e\}_\sharp$ are also in strong conflict.

Consider a prime event structure $\mathscr{E} \in \mathbb{E}$ whose conflict relation has l ($l \in \mathbf{N}$) conflict pairs. Expand each conflict relation with its successors according to the conflict inheritance property and we can get full conflict relation pairs: $\{e_1\}_\sharp \sharp^s \{f_1\}_\sharp, \ldots, \{e_l\}_\sharp \sharp^s \{f_l\}_\sharp$; here, $e_i \sharp_{\mathscr{E}} f_i$, $e_i \in E_{\mathscr{E}}$, $f_i \in E_{\mathscr{E}} (i \in \mathbf{N}, 1 \leq i \leq l)$.

Secondly, for such a group of full conflict relation pairs obtained by the above steps, there may exist common elements among some pairs that can be merged together and form a maximal conflict pattern. For example, events e_1, e_2, e_3, and e_4 are mutually in conflict; we have six immediate conflict relation pairs: $\{e_1\}\sharp^s\{e_2\}$, $\{e_1\}\sharp^s\{e_3\}$, $\{e_1\}\sharp^s\{e_4\}$, $\{e_2\}\sharp^s\{e_3\}$, $\{e_2\}\sharp^s\{e_4\}$, and $\{e_3\}\sharp^s\{e_4\}$.

From the principle of permutation, we then have three maximal conflict patterns by merging conflict pairs: $\{e_1\}\sharp^{xs}\{e_2, e_3, e_4\}$, $\{e_2\}\sharp^{xs}\{e_3, e_4\}$, and $\{e_3\}\sharp^{xs}\{e_4\}$. Equivalently, $\{e_4\}\sharp^{xs}\{e_1, e_2, e_3\}$, $\{e_3\}\sharp^{xs}\{e_1, e_2\}$, and $\{e_2\}\sharp^{xs}\{e_1\}$ are also the valid maximal conflict patterns.

Assume that there are $m \in \mathbf{N}$ maximal conflict patterns after expanding and merging which are $\mathscr{I}^1_{\sharp_{\mathscr{E}}} : A_1\sharp^{xs}B_1, \ldots, \mathscr{I}^m_{\sharp_{\mathscr{E}}} : A_m\sharp^{xs}B_m$, respectively.

Here, $\mathscr{I}^i_{\sharp_{\mathscr{E}}} : A_i\sharp^{xs}B_i$ denotes the ith pattern, and $\forall e \in A_i \subseteq E_{\mathscr{E}}, \forall f \in B_i \subseteq E_{\mathscr{E}} : e\sharp_{\mathscr{E}} f (i \in \mathbf{N}, 1 \leq i \leq m)$.

If $A_i\sharp^{xs}B_i$, then any nonempty subset of A_i and any nonempty subset of B_i are also in extended strong conflict.

For any event set D, let $\widehat{D}$ denote any nonempty subset of D $(\emptyset \neq \widehat{D} \subseteq D)$; correspondingly, $\widehat{\mathscr{I}^i_{\sharp_{\mathscr{E}}}} : \widehat{A_i}\sharp^{xs}\widehat{B_i}$ denotes a conflict subpattern of $\mathscr{I}^i_{\sharp_{\mathscr{E}}}$ $(i \in \mathbf{N}, 1 \leq i \leq m)$.

Formally, $\mathscr{I}' : A'\sharp^{xs}B'$ is called a *conflict subpattern* of $\mathscr{I} : A\sharp^{xs}B$ if and only if $(\emptyset \neq A' \subseteq A) \land (\emptyset \neq B' \subseteq B)$, denoted by $\mathscr{I}' \subseteq_{\text{pattern}} \mathscr{I}$ (or $\mathscr{I} \supseteq_{\text{pattern}} \mathscr{I}'$). Otherwise, $\mathscr{I}' \nsubseteq_{\text{pattern}} \mathscr{I}$ (or $\mathscr{I} \nsupseteq_{\text{pattern}} \mathscr{I}'$).

Let $P_{\mathscr{E}} = \{\mathscr{I}^i_{\sharp_{\mathscr{E}}} : A_i\sharp^{xs}B_i \mid i \in \mathbf{N}, 1 \leq i \leq m\}$ denote the maximal conflict pattern set of an event structure $\mathscr{E}$.

For any prime event structure, it is a certainty that we can get its maximal conflict-free substructures by some kind of conflict-free partition operation according to its conflict relation characteristics: *maximal conflict patterns*. Thus, we have the following theorem for partition.

Theorem 9. *For any prime event structure, its maximal conflict-free partition exists and the partition result is unique.*

Proof. (1) Existence. The proof is constructive.

If there is no conflict in $E_{\mathscr{E}}$, then $E_{\mathscr{E}}$ itself is the maximal conflict-free event subset of $\mathscr{E}$. Otherwise, for any nonempty event subset E $(E \subseteq E_{\mathscr{E}} \land E \neq \emptyset)$, and there exists such maximal conflict pattern $\mathscr{I}^i_{\sharp_{\mathscr{E}}} : A_i\sharp^{xs}B_i$ $(i \in \mathbf{N}, 1 \leq i \leq m)$ that $(\widehat{A_i} \cup \widehat{B_i}) \subseteq E$.

In order to make a subset E' of E $(E' \subseteq E)$ become conflict-free with respect to the conflict relation: $A_i\sharp^{xs}B_i$, that is, eliminate this conflict relation from its all subsets, we have known that if $\widehat{A_i} \subseteq E'$ (or $\widehat{B_i} \subseteq E'$), then there should be $B_i \nsubseteq E'$(or $A_i \nsubseteq E'$); otherwise, the $\mathscr{I}^i_{\sharp_{\mathscr{E}}}$ conflict pattern will still exist in its subsets.

By greedy policy, let A'_i and B'_i be both maximal inclusion subsets with respect to E calculated by $A'_i = \text{maxincl}(A_i, E)$ and $B'_i = \text{maxincl}(B_i, E)$, respectively. This means the current event set E will be partitioned into two parts by this maximal conflict subpattern: one part is $(E - A'_i)$, and the other is $(E - B'_i)$. Certainly, there exists no $\mathscr{I}^i_{\sharp_{\mathscr{E}}}$ conflict relation between $(E - A'_i)$ and $(E - B'_i)$ any more. If there does not exist any conflict in $(E - B'_i)$ (or $(E - A'_i)$), then $(E - B'_i)$ (or $(E - A'_i)$) is one conflict-free event subset of $E_{\mathscr{E}}$.

Otherwise, apply the next maximal conflict pattern $\mathcal{I}^{i+1}_{\sharp_{\mathscr{E}}}$ $(i \in \mathbf{N}, 1 \le i \le m)$ to all the previously obtained event subsets in the same manner. This partition process is continued until no conflict exists.

As we know, if each pattern of the maximal conflict patterns set has been applied just once by the above manner, then any consequent subset will be conflict-free and the partition process will stop. Meanwhile, there are 2^m conflict-free subsets at most.

Because intersection of x_i and y_i $(x, y \in \{A, B\},\ i \ne j, 1 \le i, j \le m)$ can be nonempty, thus the partition tree is not yet a full binary tree and set inclusion among these solution nodes is allowed. If some subsets are included by others, then they will be removed until every result subset cannot be included by others. It is not difficult to verify that every consequent subset is maximal and conflict-free. Exploiting these expanded fully conflict patterns to partition the event set $E_{\mathscr{E}}$ step by step, we will eventually get all maximal conflict-free event subsets. That is, there exists a practical algorithm to implement the partition operation. Without loss of generality, let $\oslash_A$ denote such partition for the time being.

(2) Uniqueness. Assume we have $mcfsetN_A(\mathscr{E})$ distinct maximal conflict-free event subsets in total by partition $\oslash_A$. These subsets form a set of *mcfsets*, denoted by $\mathbf{R}_A = \{A_1, A_2, \ldots, A_n \mid n = mcfsetN_A(\mathscr{E})\}$.

We might as well assume there is another partition $\oslash_B$ that generates the result set $\mathbf{R}_B = \{B_1, B_2, \ldots, B_m \mid m = mcfsetN_B(\mathscr{E})\}$ which is also a set of *mcfsets*.

Consider any element of $\mathbf{R}_B$; let B_i $(1 \le i \le m, i \in \mathbf{N})$ denote it. The relationship between an element A_j $(1 \le j, k \le mcfsetN_A(\mathscr{E}), j, k \in \mathbf{N})$ in $\mathbf{R}_A$ and B_i satisfies the following.

(1) $\exists A_j \in \mathbf{R}_A : A_j \subset B_i \subseteq E_{\mathscr{E}}$.

Since $\forall A_k \in \mathbf{R}_A,\ k \ne j : A_j \sharp^w A_k$, then $\forall A_k \in \mathbf{R}_A,\ k \ne j : B_i \sharp^w A_k$. We have known that $A_j \in \mathbf{R}_A$ is maximal, and now event subset B_i is also a subset of $E_{\mathscr{E}}$ and is in weak conflict with other event subsets except A_i. Moreover, B_i includes A_i. This case leads to a contradiction.

(2) $\exists A_j \in \mathbf{R}_A : B_i \subset A_j \subseteq E_{\mathscr{E}}$.

The proof is similar to the above case (1). This case also leads to a contradiction.

(3) $\forall A_j \in \mathbf{R}_A : A_j \ne B_i$.

(3.1) $\forall A_j \in \mathbf{R}_A : A_j \sharp^w B_i$.

Since B_i is also a subset of $E_{\mathscr{E}}$, thus, $\mathbf{R}'_A = \{A_1, A_2, \ldots, A_n, B_i \mid n = mcfsetN_A(\mathscr{E})\}$ is a valid set of *mcfsets*. There are $n + 1$ subsets in this partition $\oslash_A$. This is in contradiction with that there are n $(n = mcfsetN_A(\mathscr{E}))$ subsets in $\mathbf{R}_A$.

(3.2) $\exists A_j \in \mathbf{R}_A : \neg(A_j \sharp^w B_i)$.

Since $\forall A_k \in \mathbf{R}_A,\ k \ne j : A_j \sharp^w A_k$, then $\forall A_k \in \mathbf{R}_A,\ k \ne j : (A_j \cup B_i) \sharp^w A_k$; that is, $\mathbf{R}''_A = \{A_1, A_2, \ldots, \{A_j \cup B_i\}, \ldots, A_n, \mid n = mcfsetN_A(\mathscr{E})\}$ is a valid set of *mcfsets*. A_j $(A_j \in \mathbf{R}_A)$ is maximal; moreover, $A_j \subset (A_j \cup B_i) \in \mathbf{R}''_A$ is maximal too. This leads to a contradiction.

(3.3) $\forall A_j \in \mathbf{R}_A : \neg(A_j \sharp^w B_i)$.

The proof is similar to the above case (3.2). This case also leads to a contradiction.

Therefore, we are forced to have only $\exists A_j \in \mathbf{R}_A : A_j = B_i$; that is, any element in $\mathbf{R}_B$ is also an element in $\mathbf{R}_A$; we get $\mathbf{R}_B \subseteq \mathbf{R}_A$; in the same manner, we will get $\mathbf{R}_A \subseteq \mathbf{R}_B$. Thus, we have $\mathbf{R}_B = \mathbf{R}_A$.

This establishes the uniqueness and also implies the partition result is independent of partition order or conflict pattern.

Therefore, we have the conclusion.

Assume $\mathscr{E} \in \mathbb{E}$ has $mcfsetN(\mathscr{E}) \in \mathbf{N}$ *mcfESs* in total. Here, let $N_{\mathscr{E}} = mcfsetN(\mathscr{E})$ ($N_{\mathscr{E}}$, for short) denote total amount of *mcfESs*, $\mathscr{E}^{\max}_i$ $(i \in N, 1 \le i \le N_{\mathscr{E}})$ denote the ith maximal conflict-free event substructure, and $E_{\mathscr{E}^{\max}_i}$ denote the event set of $\mathscr{E}^{\max}_i$.

Then the result set can be represented as $\mathscr{B}^{mcfES}(\mathscr{E}) = \{\mathscr{E}^{\max}_i \mid i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}\}$.

In fact, every $\mathscr{E}^{\max}_i$ $(i \in, 1 \le i \le mcfsetN(\mathscr{E}))$ of the original prime event structure represents a specific possible execution choice in a system run. We might as well let *cfp* denote such an operator. Then, we have the following definition of this partition operator.

Definition 10 (conflict-free partition). An operator *cfp* is called *conflict-free partition* operator for $\mathscr{E} \in \mathbb{E}$ if and only if $\mathscr{B}^{mcfES}(\mathscr{E}) = cfp(\mathscr{E})$.

According to our previous discussion, we have C-like pseudocode descriptions: Algorithm 1 for *cfp*.

4.2. Family of Configurations.

In general, the behavior of an event structure is described by its configurations which are sets of events with certain properties. In other words, a configuration is a set of events that have happened during a specific run of the event structure.

We will review the basic definition of configuration in the following section. More detailed information can be found in [26].

Definition 11 (configuration). Let X be a subset of $X \subseteq E_{\mathscr{E}}$ of a prime event structure $\mathscr{E} \in \mathbb{E}$; then X is called a *configuration* of $\mathscr{E}$ if and only if

(1) X is left-closed if and only if $\forall c, d \in E, d \in X \wedge c \le d \Rightarrow c \in X$.

(2) X is conflict-free if and only if $\forall e, f \in X : \neg(e \sharp_{\mathscr{E}} f)$.

A configuration can also be viewed as a global state where all events in the configuration have occurred. The configuration of the event structure should be conflict-free because conflicting events can never happen in a system run. In addition, all casual predecessors of an event in

```
Input: a prime event structure: $\mathscr{E}$;
Output: the set of $mcfESs$: $\mathscr{B}^{mcfES}(\mathscr{E})$;
BEGIN
(1)  $C = \emptyset; int\ i = 1; int\ n = 0; R = \emptyset$;
(2)  $SQ = EnterQueue(E_{\mathscr{E}}, 1)$;
     /* Initialize the Queue */
(3)  if $(IsConflictFree(E_{\mathscr{E}}))${
(5)      $C = \cup\{E\}$;
(6)      Goto BUILDES;
(7)  } /* end if; */
     /* Expand and merge each conflict pair and build maximal conflict patterns:
     $\{\mathscr{I}^i_{\#_{\mathscr{E}}} : A_i \#^{xs} B_i \mid i \in, 1 \le i \le m\}$; */
(8)  for $(i = 1; i \le m; i{+}{+})$ { /* do */
(9)      Select a partition pattern: $\mathscr{I}^i_{\#_{\mathscr{E}}} : A_i \#^{xs} B_i$;
(10) /* Current level is $i$, apply $\mathscr{I}^i_{\#_{\mathscr{E}}}$, otherwise, skip while loop to the next one: $i{+}{+}$; */
(11)     while $(!EmptyQueue(SQ)$ &&
          $i == VarQueue(SQ, \textbf{LEVEL}))$ {
     /* Get the head of the Queue: event set */
(12)        $E = OutQueue(SQ, \textbf{EVENT})$;
(13)        $A'_i = maxincl(A_i, E)$;
(14)        $B'_i = maxincl(B_i, E)$;
(15)        $E_1 = (E - A'_i)$;
(16)        $E_2 = (E - B'_i)$;
     /* $E_1$ is a conflict-free subset; */
(17)        if $(IsConflictFree(E_1))$ {
(18)            $C = C \cup \{E_1\}$;
     /* Remove these elements included in others; */
(19)            $RemoveIncludedElement(C)$;
(20)        }else{
     /* Continue next partition by the conflict pattern: $\mathscr{I}^{i+1}_{\#_{\mathscr{E}}}$; */
(21)            $SQ = EnterQueue(E_1, i + 1)$;
(22)        } /* end if */
(23)        if $(IsConflictFree(E_2))$ {
(24)            $C = C \cup \{E_2\}$;
(25)            $RemoveIncludedElement(C)$;
(26)        }else{
(27)            $SQ = EnterQueue(E_2, i + 1)$;
(28)        } /* end if */
(29)     } /* end while */
(30) } /* end for */
(31) BUILDES: /* Build the sub-structure: $\mathscr{F} = (E_{\mathscr{F}}, \preceq_{\mathscr{F}}, \#_{\mathscr{F}}, l_{\mathscr{F}})$ */
(32) while $(\neg IsEmpty(C))$ {
(33)     Select a conflict-free event subset $A_{!\#}$ from $C$;
(34)     $E_{\mathscr{F}} = A_{!\#}$;
(35)     $\#_{\mathscr{F}} = \emptyset$;
(36)     $\preceq_{\mathscr{F}} = \preceq_{\mathscr{E}} \cap (A_{!\#} \times A_{!\#})$;
(37)     $l_{\mathscr{F}} = l|_{A_{!\#}}$;
(38)     $C = C - \{A_{!\#}\}$;
(39)     $R = R \cup \mathscr{F}; n{+}{+}$;
(40) } /* end while */
(41) $mcfsetN(\mathscr{E}) = n$;
(42) return $R$;
END;
```

Algorithm 1: Conflict-free partition: *cfp*.

a configuration should be contained in this configuration too; that is, configuration should be downwards closed; otherwise this event could not have happened at all.

That is, a subset X is a (finite) configuration of $\mathscr{E}$ if and only if it is finite, left-closed, and conflict-free.

The semantics of a prime event structure is defined as the family of its configurations ordered by set inclusion. Let $ConfF(\mathscr{E})$ denote the family of all configurations of event structure $\mathscr{E}$, which forms an ordered set (called prime algebraic coherent partial order; see [16]) by inclusion; that is, $(ConfF(\mathscr{E}), \subseteq)$ is partial order.

Definition 12. A configuration $X \in \text{Conf}F(\mathscr{E})$ is called *complete* or (successfully) *terminated* if and only if $\forall d \in E : d \notin X \Rightarrow \exists e \in X : e \sharp d$. A configuration $X \in \text{Conf}F(\mathscr{E})$ is called *maximal* if and only if $\forall Y \in \text{Conf}F(\mathscr{E}) : X \not\subset Y$.

For any prime event structure $\mathscr{E}$, a configuration of $\mathscr{E}$ is maximal if and only if it is complete. Obviously, for any maximal configuration of a prime event structure, there exists a corresponding maximal conflict-free substructure set. An empty or initial configuration, denoted by $\emptyset_{\text{ConfF}} \in ConfF(\mathscr{E})$, represents the initial state in which there is no event happened.

In general, initial configuration and complete configuration are also called trivial configurations, while others are called nontrivial configurations.

Similarly, we have the following configuration definition for conflict-free event structure.

Definition 13 (configuration of $cfES$). Let $\mathscr{F} = (E_{\mathscr{F}}, \preceq_{\mathscr{F}}, \emptyset, l_{\mathscr{F}})$ be a $cfES$ and let Z be a subset of $E_{\mathscr{F}}$ ($Z \subseteq E_{\mathscr{F}}$); then Z is called a configuration of $\mathscr{F}$ if and only if Z is left-closed; that is, $\forall c,\ d \in E_{\mathscr{F}},\ d \in Z \wedge c \preceq d \Rightarrow c \in Z$.

Since $\mathscr{F} \in \mathbb{F}$, its event subset is evidently conflict-free.

Let $cf\text{Conf}F(\mathscr{F})$ denote the family of all configurations of conflict-free event structure $\mathscr{F}$. Clearly, when $\mathscr{F}$ is the ith $mcfES$: $\mathscr{E}_i^{\max}$ ($i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}$) of prime event structure $\mathscr{E} \in \mathbb{E}$, its family of all configurations is denoted by $mcf\text{Conf}F(\mathscr{E}_i^{\max})$.

Definition 14 (subfamily of configurations). Let $\mathscr{F} \in$ be a $cfES$ and let Ω ($\emptyset \ne \Omega \subseteq E_{\mathscr{F}}$) be a nonempty event subset; a subfamily of configurations of $\mathscr{F}$ with respect to event subset Ω is the family of configurations of its event substructure $\mathscr{F}|_{\Omega} = (\Omega, \preceq_{\mathscr{F}} \cap (\Omega \times \Omega), \emptyset, l_{\Omega})$ restricted by event subset Ω; that is, $cf\text{subConf}F(\mathscr{F}, \Omega) \triangleq cf\text{Conf}F((\Omega, \preceq_{\mathscr{F}} \cap (\Omega \times \Omega), \emptyset, l_{\Omega}))$.

Clearly, for any $mcfES\mathscr{E}_i^{\max}$ ($i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}$) of event structure $\mathscr{E} \in \mathbb{E}$, its subfamily of configurations with respect to event subset Ω ($\emptyset \ne \Omega \subseteq E_{\mathscr{E},i}^{\max}$) is denoted by $mcf\text{subConf}F(\mathscr{E}_i^{\max}, \Omega) \triangleq mcf\text{Conf}F((\Omega, \preceq_{\mathscr{E}_i^{\max}} \cap (\Omega \times \Omega), \emptyset, l_{\Omega}))$ for convenience.

Lemma 15. *The relation between the family of configurations of a prime event structure and that of its mcfESs can be described by ConfF*$(\mathscr{E}) = \bigcup_{1\le i\le N_{\mathscr{E}}} (mcf\,ConfF(\mathscr{E}_i^{\max})$.

Proof. To prove the result of this lemma, we will show that

$$\bigcup_{1\le i\le N_{\mathscr{E}}} (mcf\text{Conf}F(\mathscr{E}_i^{\max})) \subseteq \text{Conf}F(\mathscr{E}),$$
$$\text{Conf}F(\mathscr{E}) \subseteq \bigcup_{1\le i\le N_{\mathscr{E}}} (mcf\text{Conf}F(\mathscr{E}_i^{\max})) \tag{2}$$

both hold.

(1) "$\subseteq$". For any configuration $X \in \text{Conf}F(\mathscr{E})$, since X is a configuration, by definition, X should be conflict-free. Thus X should be the subset of one of the maximal configurations. Otherwise, if X is greater than any maximal configuration, then X must contain mutual conflicting events; that is impossible.

Therefore, we have that there must exist a maximal configuration which contains X. Such a maximal configuration corresponds to a maximal conflict-free event subset: $E_{\mathscr{E}_i^{\max}} \in$ ($i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}$); that is, X must be the element of $mcf\text{Conf}F(\mathscr{E}_i^{\max})$; that is, $X \in mcf\text{Conf}F(\mathscr{E}_i^{\max})$. We have $\text{Conf}F(\mathscr{E}) \subseteq \bigcup_{1\le i\le N_{\mathscr{E}}} (mcf\text{Conf}F(\mathscr{E}_i^{\max}))$.

(2) "$\supseteq$". For any configuration $X \in mcf\text{Conf}F(\mathscr{E}_i^{\max})$ ($i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}$), of course, $X \in \bigcup_{1\le i\le N_{\mathscr{E}}} (mcf\text{Conf}F(\mathscr{E}_i^{\max}))$; this implies $X \subseteq E_{\mathscr{E}_i^{\max}}$ and $E_{\mathscr{E}_i^{\max}} \subseteq E_{\mathscr{E}}$; therefore, we get $X \subseteq E_{\mathscr{E}}$. Since X is a configuration, it is also a configuration of $\mathscr{E}$; that is, $X \in ConfF(\mathscr{E})$.

We have $\cup_{1\le i\le N_{\mathscr{E}}} (mcf\text{Conf}F(\mathscr{E}_i^{\max})) \subseteq \text{Conf}F(\mathscr{E})$.

Therefore, from (1) and (2), we have the result.

4.3. Domains of Configurations.

4.3. Domains of Configurations. In this section, we will discuss the concept of domain from the point of view that computation states are taken as such subsets and progress in a computation is measured by the occurrence of more events.

Firstly, we will recall some related conceptions regarding domain [16, 27]. Then, some important facts will be discussed.

Definition 16 (least upper bound). Let $\mathbf{D} = (D, \sqsubseteq)$ be a partial order; an element $d \in D$ is called *least upper bound* of subset X ($X \subseteq D$), denoted by $d = \sqcup X$, if and only if $(\forall x \in X : x \sqsubseteq d) \wedge (\forall d' \in D : (\forall x \in X : x \sqsubseteq d') \Rightarrow d \sqsubseteq d')$.

Definition 17 (coherent). Let $\mathbf{D} = (D, \sqsubseteq)$ be a partial order; two elements $x, y \in D$ are called *consistent* (denoted by $x \uparrow y$) if and only if $\exists z \in D : x \sqsubseteq z \wedge y \sqsubseteq z$; a subset $X \subseteq D$ is *pairwise consistent* if and only if any two of its element have an upper bound in D; that is, $\forall x, y \in X : x \uparrow y$; $(D, \sqsubseteq)$ is called *coherent* if and only if every pairwise consistent subset X ($X \subseteq D$) has a least upper bound $\sqcup X$.

The *consistency* relation of x and y is denoted by $x \uparrow y$; conversely, *inconsistency* is denoted by $x \not\uparrow y$.

Definition 18 (complete prime). A partial order $\mathbf{D} = (D, \sqsubseteq)$; an element is a *complete prime* if and only if for every finite subset $X \subseteq D$, if $\sqcup X$ exists and $p \subseteq \sqcup X$ then there exists an $x \in X$ such that $p \sqsubseteq x$ (i.e., $p \sqsubseteq \sqcup X \Rightarrow \exists x \in X . p \sqsubseteq x$).

Let $P(D)$ denote the set of complete prime of $(D, \sqsubseteq)$.

Definition 19 (prime algebraic). A partial order $\mathbf{D} = (D, \sqsubseteq)$ is called *finitary* if and only if $\forall p, d \in P(D) : \{d \in D \mid d \sqsubseteq p\}$ is finite. $(D, \sqsubseteq)$ is called *prime algebraic* if and only if $P(D)$ is countable and $\forall d \in D : d = \sqcup\{p \in P(D) \mid p \sqsubseteq d\}$.

Namely, $\mathbf{D}$ is called *prime algebraic* if and only if, for every element $d \in D$, $\sqcup \mathbf{D}_d$ exists (define $\sqcup \mathbf{D}_d = \{p \sqsubseteq d \mid p \text{ is a complete prime}\}$), and $d = \sqcup \mathbf{D}_d$.

Definition 20 (domain). A coherent, prime algebraic, and finitary partial order is called a *Scott domain* (or simply a *domain*).

Definition 21. Let $\mathbf{D} = (D, \sqsubseteq)$ be a *coherent, finitary prime algebraic domain.* Define $\mathscr{P}_r[D] = (P(D), \preceq, \sharp)$, where $P(D)$ consists of the complete primes of $\mathbf{D}$:

(1) $\forall p, p' \in P(D) : p \preceq p' \Leftrightarrow p \sqsubseteq p'$;

(2) $\forall p, p' \in P(D) : p \sharp p' \Leftrightarrow p \not\uparrow p'$.

Definition 22. Let $\mathbf{D} = (D, \sqsubseteq)$ be a *prime algebraic complete lattice.* Define $\mathscr{P}_r[D] = (P(D), \preceq)$, where $P(D)$ consists of the complete primes of $\mathbf{D}$, $\forall p, p' \in P(D) : p \preceq p' \Leftrightarrow p \sqsubseteq p'$.

Theorem 23. *Let $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$; then $(ConfF(\mathscr{E}), \subseteq)$ is a finitary coherent prime algebraic domain; the complete primes are the set $\{a \in E_{\mathscr{E}} \mid a \preceq e, e \in E_{\mathscr{E}}\}$ (see [25]).*

Theorem 24. *Let $(D, \sqsubseteq)$ be a finitary coherent prime algebraic domain. Then, $\mathscr{P}_r[D] = (P(D), \preceq, \sharp)$ is a prime event structure, with $\varphi : (D, \sqsubseteq) \cong (ConfF(\mathscr{P}_r[D]), \subseteq)$ giving an isomorphism of partial orders where $\varphi(d) = \{p \sqsubseteq d \mid p \text{ is a complete prime}\}$ with inverse $\lambda : ConfF(\mathscr{P}_r[D]) \to (D, \sqsubseteq)$ given by $\lambda(x) = \sqcup x$ (see [16]).*

Evidently, event structures and coherent, finitary prime algebraic domains are equivalent; one can be used to represent the other.

The following theorem describes the important property of family of configurations of a prime event structure.

Theorem 25. *For any nonempty mcfES: $\mathscr{E}_i^{\max}$ ($i \in \mathbf{N}, 1 \leq i \leq N_{\mathscr{E}}$) of event structure $\mathscr{E} \in \mathbb{E}$, its family of configurations $(mcfConfF(\mathscr{E}_i^{\max}), \subseteq)$ is prime algebraic complete lattice. Its complete primes are those elements of the form $\{a \in E_{\mathscr{E}_i^{\max}} \mid a \preceq e, e \in E_{\mathscr{E}_i^{\max}}\}$.*

Proof. The proof is straightforward.

Thus prime event structure and finitary coherent prime algebraic domain are equivalent; this implies that there is a one-to-one correspondence between a prime event structure and its family of configurations; one can be used to represent the other.

4.4. Weak Choice Composition. Theorem 23 describes an important property between the domains of configurations of prime event structures and the prime event structures themselves.

We can obtain a full set of *mcfESs* from a prime event structure by applying *cfp* operator over it. Conversely, given a full set of *mcfESs* of an event structure, we can certainly recover the original event structure that generates this set of *mcfESs* by some kind of composition operation.

Further, for any weak conflict set, we give the constraint conditions, under which this weak conflict set can be composed together and form a prime event structure that can generate this set by conflict-free partition operation.

The following theorem discusses the constraint conditions for composition.

Theorem 26 (necessary and sufficient condition for composition). *For any weak conflict set $\mathscr{WF}_n^{cfw} = \{\mathscr{F}_i \in \mathbb{F} \mid i, n \in \mathbf{N}, 1 \leq i \leq n\}$, if it satisfies the following conditions: (1) and (2), then there exists a unique prime event structure $\mathscr{E} \in \mathbb{E}_{prime}$ that can generate this set by cfp partition operation; that is, $\mathscr{WF}_n^{cfw} = cfp(\mathscr{E})$.*

(1) $\forall \mathscr{F}_i, \mathscr{F}_j \in \mathscr{WF}_n^{cfw} : \Omega = (E_{\mathscr{F}_i} \cap E_{\mathscr{F}_j}) \wedge (\Omega \neq \emptyset) \Rightarrow$ *$cfsubConfF(\mathscr{F}_i, \Omega) = cfsubConfF(\mathscr{F}_j, \Omega)$ ($i, j \in N, i \neq j, 1 \leq i \leq n$).*

(2) *$(\bigcup_{i=1}^{n} cfConfF(\mathscr{F}_i), \subseteq)$ is a finitary coherent prime algebraic domain.*

Proof. On one hand, the intersection of event sets of any two *cfESs* is nonempty meaning that common events have happened from both event structures. By definition, if these events represent common global states in runs of a system described by the same prime event structure with multiple choices, they should behave identically. That is, their configurations with respect to the intersection of event set should be identical.

In addition, from Theorems 23 and 24, the family of configurations of a prime event structure ordered by set inclusion should be a finitary coherent prime algebraic domain.

Thus, we have the necessary condition for composition.

On the other hand, from Theorems 23 and 24, we have that there is a one-to-one correspondence between a prime event structure and its family of configurations. Given a valid family of configurations for prime event structure, then there should exist a corresponding prime event structure.

For any weak conflict set: $\mathscr{WF}_n^{cfw} = \{\mathscr{F}_i \in \mathbb{F} \mid i, n \in, 1 \leq i \leq n\}$, if all $cfConfF(\mathscr{F}_i)$ by joining can form a valid family of configurations for a prime event structure, that is, $\bigcup_{1 \leq i \leq n} cfConfF(\mathscr{F}_i)$ forms a ordered by set inclusion, then there should exist such a unique prime event structure $\mathscr{E}$ that $ConfF(\mathscr{E}) = \bigcup_{1 \leq i \leq n} cfConfF(\mathscr{F}_i)$.

Therefore, we get the necessary and sufficient condition for composition.

Obviously, the set $\mathscr{B}^{mcfES}(\mathscr{E})$ of a prime event structure satisfies the above condition. Clearly, this implies that there must exists a composition operation which can construct the target event structure $\mathscr{E}$ from a weak conflict set that satisfies the constraint conditions. We may as well let *wcc* denote the operator. Thus, we have the following definition.

Definition 27 (weak choice composition (*wcc* operator)). Let $\mathscr{WF}_n^{cfw}$ be a weak conflict set, which satisfies necessary and sufficient conditions for composition; an operator *wcc* is called *weak choice composition* operator if and only if the result event structure $\mathscr{R} = wcc(\mathscr{WF}_n^{cfw})$ and $\mathscr{R}$ satisfies the following:

(1) $\mathrm{Conf}F(R) = (\bigcup_{i=1}^{n} cf\mathrm{Conf}F(\mathscr{F}_i), \subseteq)$;

(2) $\mathscr{WF}_n^{cfw} = cfp(R)$.

The following theorem states that the operator *wcc* and *cfp* are mutually inverse for a prime event structure.

Theorem 28. *For any $\mathscr{E} \in \mathbb{E}$, $\mathscr{E} = wcc(cfp(\mathscr{E}))$ holds.*

Proof. The proof is straightforward.

Obviously, it is not difficult to derive an algorithm for weak conflict composition operator from Definition 27 and Theorem 28.

5. Slicing Reduction

In this section, we will discuss slicing reduction technique for partial order trace or prime event structure. Slicing is often taken as an effective abstract technique to combat the state explosion problem. A slicing algorithm for event structure with respect to predicates in a subset of temporal logic formulas is studied. Specially, we focus on statically analyzing rather than online detecting over event structure model.

First of all, we will retrospect the classical notion of computation slicing for partial order traces. Then, we will extend the idea from partial order traces to prime event structures with conflict relations. Additionally, all related definitions and theorems [18, 19, 28] for our theory will be discussed.

5.1. Partial Order Trace Slicing. Computation slicing was introduced in [7] as an abstraction technique for analyzing partial order traces of distributed programs or distributed computations.

Generally, for classical program slicing, programs are sliced with respect to a slicing criterion that is an interested point for analyzing. In static program slicing, for example, "a program line number" can be taken as a valid slicing criterion. Thus, in order to compute a slice, we need to firstly define the slicing criterion.

Intuitively, a slice of a trace with respect to a temporal logic specification or a predicate (slicing criterion) φ is a subtrace that contains all the states of the trace that satisfy φ. A slice contains all the states that satisfy φ such that it can be computed efficiently and is often much smaller than the original model.

We can use directed graphs to model partial order (execution) traces (POTs, for short) as well as slices. Thus, a notion named *graph ideal* (or *order ideal*) of directed graph [29] is introduced to specify partial order traces and slices pictorially. Formally, its definition is given as follows.

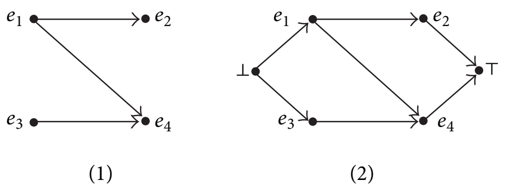


Figure 1: Partial order trace and its directed graph representation.

Definition 29 (order ideal). Given a poset $(X, \leq)$, ($\leq$ denotes an order relation) a subset S of X is an *order ideal* if it satisfies $\forall x, y : x, y \in X : (x \in S) \wedge (y \leq x) \Rightarrow (y \in S)$.

Definition 30 (graph ideal). Given a directed graph $G = (V, \Gamma)$, let $V(G)$ and $\Gamma(G)$ denote the set of vertices with event labels and directed edges, respectively. A subgraph H of G is a *graph ideal* if it satisfies $\forall u, v \in V(G), H \subseteq V(G), v \in H \wedge (u, v) \in \Gamma(G) \Rightarrow u \in H$.

It is more convenient to use directed graphs to represent partially ordered sets and prime event structures for slicing computation. It satisfies the following.

(1) For any event e and f of $\mathscr{E}$, if $e \preceq f$, then there is directed edge from the vertex v_e labelled with e to the vertex v_f labelled with f.

(2) For any event e and f of $\mathscr{E}$, if $e \sharp f$, then there is dash line between the vertex v_e labelled with e to the vertex v_f labelled with f.

For example, as shown in Figure 1, a partial order trace or a *mcfES* is demonstrated pictorially. The corresponding event structure for Figure 1 is as follows.

(i) $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}), (Act = \{a, b, c, d\})$.

(1) $E_{\mathscr{E}} = \{e_1, e_2, e_3, e_4\}$.
(2) $\preceq_{\mathscr{E}} = \{e_1 \preceq e_2, e_1 \preceq e_4, e_3 \preceq e_4\}$.
(3) $l(e_1) = a,\ l(e_2) = b,\ l(e_3) = c,\ l(e_4) = d$.
(4) $\sharp_{\mathscr{E}} = \emptyset$.

In addition, when attempting to construct the graph representation of *mcfES*, as Figure 1 shows, two specific vertexes $\top$ and $\perp$ will be added as initial state and terminal state corresponding to initial configuration and maximal configuration, respectively.

A subset of elements forms an order ideal if whenever an element is contained in the subset then all its preceding elements are also contained in the subset. Intuitively, order ideals or left-closed subsets can be graphically represented by graph ideals. Generally, independency relation will not be represented explicitly. It is not difficult to have that partial order trace is only a special case of prime event structure with no conflict relations. Here, graph ideal is a notion equivalent to the configuration of an event structure. Empty set and the set of all vertices are called trivial ideal. Similarly, initial configuration and complete configuration are also called trivial configurations.

Definition 31 (predicate on configuration). Intuitively, a logic formula or predicate is a Boolean-valued function defined on the set of configurations: $\varphi : \mathrm{Conf}F(\mathscr{E}) \to \{0,1\}$. It actually represents a subset of configurations in which the Boolean function evaluates to 1.

The predicate detection problem is to decide whether the initial configuration of an event structure satisfies a predicate. More formally, we have the following definition.

Definition 32 (predicate detection). For any prime event structure $\mathscr{E}$ and any predicate φ, predicate detection is to decide whether $\mathrm{Conf}F(\mathscr{E}), \{\perp\} \models \varphi$ holds or not.

Predicates are used to specify system behaviors and properties such as safety and liveness. Properties expressed by a CTL (computational tree logic, introduced in [30]) formula are beyond the scope of this paper. For evaluating the value of a predicate efficiently, various predicate classes [28] such as conjunctive, stable, observer-independent, linear, relational, and nontemporal regular [7] predicates have been defined.

Generally, predicate on configurations will act as the slicing criterion for POTs slicing.

Definition 33 (slice of *mcfES*(POTs)). A *slice* of a *mcfES*(POTs): $\mathscr{E}_i^{\max}$ $(i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}})$ of prime event structure $\mathscr{E}$ with respect to a formula φ, denoted by $mcfES\mathrm{slice}(\mathscr{E}_i^{\max}, \varphi)$, is such an event structure that satisfies the following.

(i) Its family of configurations contains all the configurations that satisfy φ.

(ii) Its family of configurations has the least number of configurations and still forms a sublattice.

This formal definition is derived from computation slice notion [7] given by Garg and Mittal. Meanwhile, existence and uniqueness of the *mcfES* slice have also been discussed; that is, the following theorem holds.

Theorem 34. *For any $\mathscr{E}_i^{\max}$ $(i \in N, 1 \le i \le N_{\mathscr{E}})$ of a prime event structure and any predicate φ, the slice of with respect to predicate φ, that is, $mcfESslice(\mathscr{E}_i^{\max}, \varphi)$ exists and is unique.*

Proof. The proof is straightforward; see [8, 20, 31].

In general, the family of configurations for a *mcfES* forms a distributed lattice, and its slice with respect to a predicate is a sublattice. Sometimes a slice may contain those configurations that do not satisfy the predicate for completing sublattice.

In the next section, we will discuss the slicing definition and model for prime event structure.

5.2. Sliced Model over Event Structure. Generally, predicate on configurations acts as the slicing criterion for prime event structure slicing. Temporal regular predicate, such as a regular subset of CTL called RCTL [7, 8, 29], which contains four temporal operators EF, AG, EG, and EX[j], and nontemporal regular predicates both can also be taken as the slicing criterions.

Compared with the definition of slice of *mcfES*, we have a similar case for prime event structure.

Definition 35 (slice of prime event structure). A slice of a prime event structure with respect to a formula φ, denoted by $SliceES(\mathscr{E}, \varphi)$, is such an event structure that satisfies the following.

(i) Its family of configurations contains all the configurations that satisfy φ.

(ii) Its family of configurations has the least number of configurations.

Generally, a slice may contain configurations that do not satisfy the given predicate. The slice of an event structure with respect to a predicate is called *lean* [32] if every configuration of the slice satisfies the predicate.

Theorem 36. *For any $\mathscr{E} \in \mathbb{E}$ and any predicate φ, $SliceES(\mathscr{E}, \varphi)$ exists and is unique, and $SliceES(\mathscr{E}, \varphi) = wcc(mcfESslice(cfp(\mathscr{E}), \varphi))$ holds.*

Proof. (1) Existence and Uniqueness. From Theorem 34, we have that, for any $\mathscr{E}_i^{\max}$ $(i \in N, 1 \le i \le N_{\mathscr{E}})$ of a prime event structure $\mathscr{E}$ and any predicate φ, its slice with respect to predicate φ exists and is unique.

For any *mcfES*, the family of configuration of $mcfES\mathrm{slice}(\mathscr{E}_i^{\max}, \varphi)$ is a distributed lattice and is unique.

Further, let $\bigcup_{\mathscr{E}}^{\mathbf{C}} = \bigcup_{1 \le i \le N_{\mathscr{E}}} (mcf\mathrm{Conf}(mcfESslice)((\mathscr{E}_i^{\max}, \varphi)))$; $\bigcup_{\mathscr{E}}^{\mathbf{C}}$ is also unique and $(\bigcup_{\mathscr{E}}^{\mathbf{C}}, \subseteq)$ is a finitary coherent prime algebraic domain.

Next, we show that the slicing operation will keep the second part of necessary and sufficient condition for composition.

For any $\mathscr{E}_i^{\max}$ and $\mathscr{E}_j^{\max}$ $(j \ne i)$, if $E_{\mathscr{E}_i^{\max}} \cap E_{\mathscr{E}_j^{\max}} = D$ and $D \ne \emptyset$, we then have that $cf\mathrm{subConf}F(\mathscr{E}_i^{\max}, D) = cf\mathrm{subConf}F(\mathscr{E}_j^{\max}, D)$; that is, for any nonempty event subset D' $(D' \subseteq D)$ and any predicate φ, if any configuration of D' satisfies φ, that is, D' is the common part of both slices of $\mathscr{E}_i^{\max}$ and $\mathscr{E}_j^{\max}$. We still get that $cf\mathrm{subConf}F(\mathscr{E}_i^{\max}, D') = cf\mathrm{subConf}F(\mathscr{E}_j^{\max}, D')$.

This means that, for any two *mcfESs*, if their intersection is nonempty, no matter which part of the intersection belongs to the slice, after slicing, the necessary and sufficient condition for composition will be still satisfied.

Thus, we get that $\bigcup_{\mathscr{E}}^{\mathbf{C}}$ is a valid family of configurations for prime event structures; there should exist such a unique prime event structure $R \in \mathbb{E}$ that satisfies $\mathrm{Conf}F(R) = \bigcup_{\mathscr{E}}^{\mathbf{C}}$. We can get R by applying *wcc* to the corresponding event structures of $\bigcup_{\mathscr{E}}^{\mathbf{C}}$.

Therefore, the existence and uniqueness for event structure slicing have been established. We will then prove that the prime event structure R is the ultimate result of slicing.

(2) Satisfactoriness and Minimality. On the one hand, for any configuration X of event structure $\mathscr{E}$ that makes predicate φ hold, that is, $X \in \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E}, \varphi))$, there must be a *mcfES*: $\mathscr{E}_i^{\max}$ so that $X \in \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E}, \varphi))$;

let $\mathbf{C}_i = mcf\mathrm{Conf}F(mcfES\mathrm{slice}(\mathscr{E}_j^{\max},\varphi))$, because $\mathbf{C}_i$ contains all the configurations of event structure $\mathscr{E}_j^{\max}$ that make predicate φ hold. We have that X must be contained by $\mathbf{C}_i$; that is, $X \in \mathbf{C}_i$.

We get $X \in \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi)) \Rightarrow X \in \bigcup_{\mathscr{E}}^{\mathbf{C}}$.

Further, we get $\mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi)) \subseteq \bigcup_{\mathscr{E}}^{\mathbf{C}}$.

On the other hand, for any configuration $X \in \bigcup_{\mathscr{E}}^{\mathbf{C}}$, we get $X \in \mathrm{Conf}F(\mathscr{E})$ and X can make predicate φ hold; then $X \in \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E}))$ must hold. Thus, we get $X \in \bigcup_{\mathscr{E}}^{\mathbf{C}} \Rightarrow X \in \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi))$.

That is, $\bigcup_{\mathscr{E}}^{\mathbf{C}} \subseteq \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi))$.

Therefore, we have $\bigcup_{\mathscr{E}}^{\mathbf{C}} = \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi))$.

Thus, we get that $\mathrm{Slice}ES(\mathscr{E},\varphi) = R$.

Moreover, by the definition of slice of maximal conflict-free event substructure, we have that, for any $\mathscr{E}_i^{\max}$ ($i \in \mathbf{N}, 1 \le i \le N_{\mathscr{E}}$), the corresponding $mcfES\mathrm{slice}(\mathscr{E}_i^{\max},\varphi)$ contains the least number of configurations that satisfy the given predicate φ; we then have that $\bigcup_{\mathscr{E}}^{\mathbf{C}} = \mathrm{Conf}F(\mathrm{Slice}ES(\mathscr{E},\varphi))$ also contains the least number of configurations satisfying this specification. Thus, satisfactoriness and minimality both hold.

Consequently, from both (1) and (2), we conclude that the theorem holds.

5.3. Slicing Reduction Algorithm. In this section, we will present an approach for event structure slice computing. The slicing algorithm for a prime event structure or its *mcfESs* with respect to regular predicates is based on the *Adding Edges Theorem* (see [8, 20, 31, 33]).

In fact, by the following theorem, these lattices will never be actually constructed in the slicing process for efficiency.

The configurations do not satisfy the predicate but still can be included to complete the sublattice.

Given a distributive lattice L generated by a graph G, every sublattice of L can be generated by a graph obtained by adding edges to G. The following theorem holds.

Theorem 37 (Adding Edges Theorem). *Let L' be any sublattice of a finite distributive lattice L generated by the directed graph G. Then, there exists a graph G' that can be obtained by adding edges to (removing vertices from) G that generates L'.*

For any prime event structure, we can get the slices of its *mcfESs* by applying the Adding Edges Theorem. These slices can be composed by *wcc* to form a new prime event structure which is the target slice of the original event structure. This approach is less general but results in more efficient detection algorithms for a special class of predicates. Note that we will never actually construct the lattice or family of configurations of the event structure due to efficiency.

Garg and Mittal have presented an efficient algorithm $\mathrm{slice}(G,\varphi)$ [8, 28] based on graphical representation G to compute the slice of POTs (or conflict-free event structures) with respect to a predicate φ. The algorithm adopts the principle of the Adding Edges Theorem and can produce a sliced graph representation. Especially, we have $G = \mathrm{slice}(G,\mathrm{true})$ for predicate $\varphi = \mathrm{true}$ itself.

We extend the idea and algorithm to more general models and provide an algorithm for slicing the *mcfESs* and the original prime event structure. Thus, we have Algorithm 2 to compute the slice of conflict-free event structure.

For a prime event structure with conflict relations, we have to apply *cfp* operator to get $mcfsetN(\mathscr{E})$ maximal conflict-free event substructures and each of them can be sliced by *mcfES*slice. Then, the set consisting of each sliced result can be composed together by *wcc* to construct a new event structure. This new event structure will be the sliced result.

Thus, we can derive Algorithm 3 to compute the slice of a prime event structure.

Because the set of the slices of *mcfESs* may no longer keep the weak conflict relation which exists in the original *mcfESs*.

Therefore, after $mcfES\mathrm{slice}(mcfES(\mathscr{E}),\varphi)$ operation is performed, the relation among these slices can be one of the following cases:

(1) strong conflict;

(2) conflict-free;

(3) weak conflict;

(4) hybrid of weak conflict and conflict-free;

(5) hybrid of weak conflict and strong conflict;

(6) hybrid of strong conflict, weak conflict, and conflict-free.

In case of (1), (3), and (5), the operation *wcc* can be performed directly. But in case of (2), (4), and (6), we have to add some events in order to make the result set of slices still be able to form a valid weak conflict set at the end of process.

For temporal predicates [8], such as EF, EG and AG can be computed by $\mathrm{slice}(G,EF(\varphi))$, $\mathrm{slice}(G,EG(\varphi))$, and $\mathrm{slice}(G,AG(\varphi))$, respectively. From the definition of a slice, we know that every configuration of a slice $\mathrm{slice}ES(\mathscr{E},\varphi)$ is also a configuration of $\mathscr{E}$.

Clearly, the following two corollaries hold.

Corollary 38. *(1) For any prime event structure $\mathscr{E} \in \mathbb{E}$, $ConfF(sliceES(\mathscr{E},EG(\varphi))) \subseteq ConfF(sliceES(\mathscr{E},\varphi)) \subseteq ConfF(\mathscr{E})$.*

Similarly for AG, the following holds.

(2) For any prime event structure $\mathscr{E} \in \mathbb{E}$, $ConfF(sliceES(\mathscr{E},AG(\varphi))) \subseteq ConfF(sliceES(\mathscr{E},\varphi)) \subseteq ConfF(\mathscr{E})$.

Corollary 39. *For any prime event structure $\mathscr{E} \in \mathbb{E}$, $ConfF(sliceES(\mathscr{E},EF(\varphi))) \subseteq ConfF(\mathscr{E})$.*

5.4. Case Study for Slicing Reduction. In this section, we will give an example to illustrate the prime event structure slice notion and its computing process.

Consider a prime event structure: $\mathscr{E} = (E_{\mathscr{E}}, \le_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}})$, as shown in Figure 2. The components are described as follows:

(1) event set: $E_{\mathscr{E}} = \{e_a, e_b, e_c, e_1, e_2, e_3, e_4\}$;

(2) conflict relation: $\#_{\mathscr{E}} = \{e_b \# e_1\}$;

```
Input:
(1) a conflict-free event structure: $mcfES(\mathscr{E})$,
(2) a regular predicate: $\varphi$
Output: the slice of $mcfES$: ${\mathscr{E}_i^{\max}}_{slice}$
BEGIN
(1) $K = \emptyset$;
(2)    generate graph representation $G$ for $\mathscr{E}_i^{\max}$;
(3)    computing slice: $K = slice(G, \varphi)$;
(4)    generate event structure ${\mathscr{E}_i^{\max}}_{slice}$ from graph representation $K$;
(5)    return ${\mathscr{E}_i^{\max}}_{slice}$;
END
```

Algorithm 2: Slicing algorithm: $mcfESslice(mcfES(\mathscr{E}), \varphi)$.

```
Input:
(1) an event structure: $\mathscr{E}$
(2) a regular predicate: $\varphi$
Output: the slice: $\mathscr{E}_{slice}$
BEGIN
(1)  $int\ n = 0, C = \emptyset$;
(2)  $\mathscr{R} = \emptyset$;
(3)  $\mathscr{B}^{mcfES}(\mathscr{E}) = cfp(\mathscr{E})$
(4)  $n = mcfsetN(\mathscr{E})$;
(5)  for ($int\ i = 0; i < n; i$++) {
(6)        get the $i$th $mcfES$: $\mathscr{E}_i^{\max}$ from $\mathscr{B}^{mcfES}$;
(7)        $C = C \cup mcfESslice(\mathscr{E}_i^{\max}, \varphi)$;
(8)  }
(9)  if ($C \neq \emptyset$){
(10)       $\mathscr{R} = wcc(C)$;
(11) }
(12) return $\mathscr{R}$;
END
```

Algorithm 3: Slicing algorithm: $sliceES(\mathscr{E}, \varphi)$.

(3) casual relation: $\preceq_{\mathscr{E}} = \{e_b \preceq e_a, e_c \preceq e_a, e_1 \preceq e_2, e_1 \preceq e_4, e_3 \preceq e_4, e_c \preceq e_3\}$;

(4) action labels: $l_{\mathscr{E}}(e_a) = a$, $l_{\mathscr{E}}(e_b) = b$, $l_{\mathscr{E}}(e_c) = c$,$l_{\mathscr{E}}(e_1) = a_1$, $l_{\mathscr{E}}(e_2) = a_2$, $l_{\mathscr{E}}(e_3) = a_3$, $l_{\mathscr{E}}(e_4) = a_4$;

(5) action functions:

(i) action(a_1) : $\{x = x + 1\}$;
(ii) action(a_2) : $\{x = x + 3\}$;
(iii) action(a_3) : $\{y = y + 3\}$;
(iv) action(a_4) : $\{y = y + 2\}$;
(v) action(a) : $\{z = z + 2\}$;
(vi) action(b) : $\{z = z + 1\}$;
(vii) action(c) : $\{y = y - 1\}$;
(viii) initValue : $\{x = 1; y = 1; z = 0\}$;

(6) slice criterion: $\{\varphi = -2 \leq (y - x + z) < 2\}$.

In this example, the system global states will be updated after an action function executes. Figure 2 depicts all the events conflict (for simplicity, only immediate conflict relation is shown) and casual relation. Figure 3 shows its corresponding family of configurations.

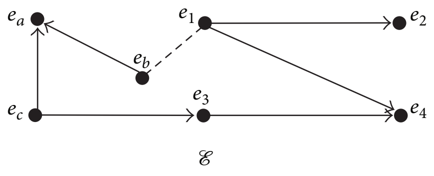


Figure 2: A Prime event structure $\mathscr{E}$.

There is one conflict relation between event b and e_1; due to the conflict inheritance property of prime event structures, we have $e_b \sharp e_1, e_b \sharp e_2, e_b \sharp e_4$ and $e_a \sharp e_1, e_a \sharp e_2, e_a \sharp e_4$; that is, each of $\{e_a, e_b\}$ is in conflict with each of $\{e_1, e_2, e_4\}$. Obviously, according to this conflict relation, apply cfp operation to the prime event structure $\mathscr{E}$ and we get that this event structure has only two $mcfESs$: they are $\mathscr{E}_1 = (\{e_a, e_b, e_c, e_3\}, \leq_{\mathscr{E}_1}, \emptyset, l_{\mathscr{E}_1})$ and $\mathscr{E}_2 = (\{e_1, e_2, e_3, e_4, e_c\}, \leq_{\mathscr{E}_2}, \emptyset, l_{\mathscr{E}_2})$ depicted by Figures

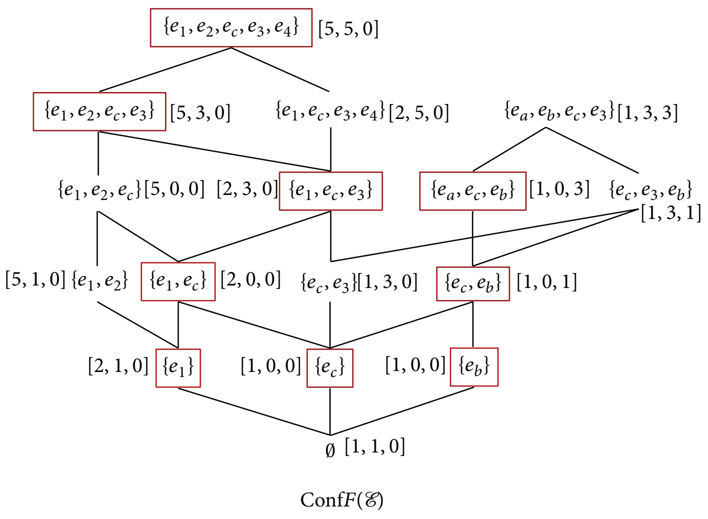

$\{e_1, e_2, e_c, e_3, e_4\}$ [5, 5, 0]
$\{e_1, e_2, e_c, e_3\}$ [5, 3, 0]
$\{e_1, e_c, e_3, e_4\}$ [2, 5, 0]
$\{e_a, e_b, e_c, e_3\}$ [1, 3, 3]
$\{e_1, e_2, e_c\}$ [5, 0, 0]
[2, 3, 0] $\{e_1, e_c, e_3\}$
$\{e_a, e_c, e_b\}$ [1, 0, 3]
$\{e_c, e_3, e_b\}$ [1, 3, 1]
[5, 1, 0] $\{e_1, e_2\}$
$\{e_1, e_c\}$ [2, 0, 0]
$\{e_c, e_3\}$ [1, 3, 0]
$\{e_c, e_b\}$ [1, 0, 1]
[2, 1, 0] $\{e_1\}$
[1, 0, 0] $\{e_c\}$
[1, 0, 0] $\{e_b\}$
$\emptyset$ [1, 1, 0]
Conf$F(\mathscr{E})$

FIGURE 3: Family of configurations of $\mathscr{E}$.

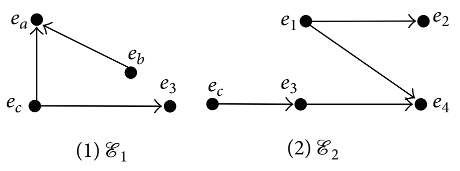


FIGURE 4: *mcfESs* of $\mathscr{E}$: $\mathscr{E}_1$ and $\mathscr{E}_2$.

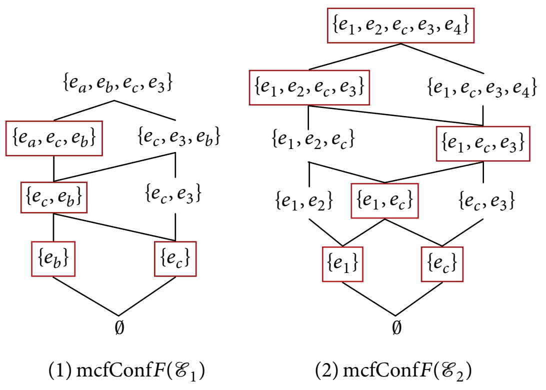


FIGURE 5: Families of configurations of *mcfESs*($\mathscr{E}_1$ and $\mathscr{E}_2$).

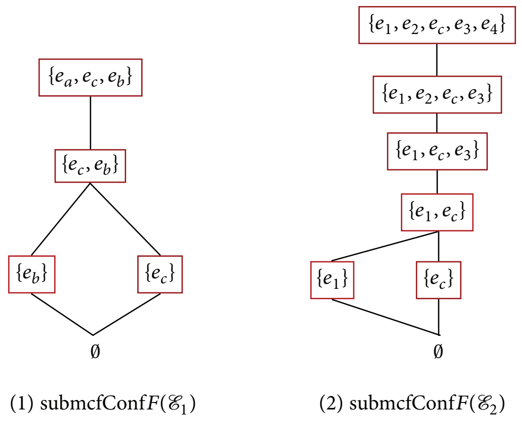


FIGURE 6: Subfamilies of configurations of the slices.

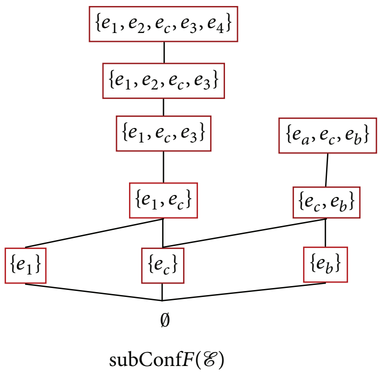


FIGURE 7: Subfamily of configurations of the slice.

4(1) and 4(2), respectively, and Figures 5(1) and 5(2) show their corresponding families of configurations.

The configurations that satisfy the predicate are labelled with frames. In fact, these configurations are only used to describe relationship between original event structure and its slice graphically; in general, they will never be actually constructed in the slicing algorithm for efficiency.

The families of configurations of the slices of $\mathscr{E}_1$ and $\mathscr{E}_2$ with respect to the predicate $\{\varphi = -2 \le (y - x + z) < 2\}$ are shown in Figures 6(1) and 6(2), respectively. It can be verified that both $mcf\text{Conf}(\mathscr{E}_1)$ and $mcf\text{Conf}(\mathscr{E}_2)$ form distributed lattices.

These configurations of the slices are exactly the ones that satisfy the given predicate in the family of configurations of the original event structure. Figure 7 shows the family of configurations constructed by applying $\cup$ operation to the families of configurations of all slices.

Finally, in Figure 8, the slice of $\mathscr{E}_1$ and the slice of $\mathscr{E}_2$ are combined into the slice of $\mathscr{E}$ by *wcc* operator, as expected.

To illustrate the benefit of predicate detection by using slicing reduction as shown in above example, consider the states in Figure 3 again.

Let $\{\varphi = -2 \leq (y - x + z) < 2\}$ be the predicate to be checked, and suppose we want to detect whether $EF(\varphi)$ holds or not; that is, there exists a global state that satisfies φ. Without slicing reduction applied, we are forced to examine all global states, 15 states in total as shown in Figure 3, to decide whether the traces satisfy the predicate.

Alternatively, we can compute the slice via slicing reduction technique with respect to the regular temporal predicate and use this slice for predicate detection.

For this purpose, firstly, we compute the slice with respect to φ and the slice is shown in Figure 7.

Finally, we check whether the initial state is the same as the initial state of the slice and decide whether the predicate is satisfied or not.

The slice contains only 9 states and has much fewer states than the original traces itself. Generally, it is exponentially smaller in many cases and this can result in substantial savings.

6. Symmetry Reduction

Finite state systems frequently exhibit symmetry which can be found in memories, caches, register files, bus protocols, and anything that has a lot of replicated structures. The use of symmetry to reduce state space has been investigated widely by researchers [2, 3, 15, 22, 23].

In this section, we will discuss symmetry properties over prime event structures. Symmetry in an event structure implies the existence of nontrivial permutation groups that preserve both the events labelling and all relations of causal dependence and independence that exist between events. We start by introducing some notions of group theory.

6.1. Automorphism Groups. We know that the set of all permutation on a set forms a permutation group under functional composition. A permutation group over a finite set X consists of bijections, $X \rightarrow X$, and their compositions as the binary operations.

Definition 40 (permutation group). Let X be a finite set; a *permutation* of X is a bijection from X to itself. Then, $S_X := \{f : X \rightarrow X \mid f \text{ is abijection}\}$; that is, the family of all the permutations of the set X, denoted by S_X, forms a group called the *symmetric group* on X. For any bijection $f \in S_X$ is called a *permutation.* Any subgroup of is called a permutation group on X.

Obviously, a symmetric group is a special permutation group. A permutation group over a set has good properties; specially, it can induce an equivalence relation. The equivalence classes of an equivalence relation on a set can form a partition of this set. Thus, for a set X, if there exists a permutation group on the set X, the permutation group can induce a partition of the set X. We can easily check this property.

In this paper, permutation groups are used to partition the set of events in an event structure so that we use equivalence classes (orbits) of events to investigate symmetry in this event structure.

Definition 41 (automorphism). Let $\mathscr{E} = (E, \preceq, \sharp, l) \in \mathbb{E}$ and let G be a permutation group on the event set E of $\mathscr{E}$. A permutation $f \in G$ is said to be an *automorphism* of E if and only if f satisfies the following conditions:

(1) $\forall e_1, e_2 \in E : e_1 \preceq e_2 \Rightarrow f(e_1) \preceq f(e_2)$, $e_1 \sharp e_2 \Rightarrow f(e_1) \sharp f(e_2)$;

(2) $l(e_1) = l(f(e_2))$.

Definition 42 (automorphism group). A permutation group G is called an *automorphism group* for the event structure $\mathscr{E}$ ($\mathscr{E} = (E, \preceq, \sharp, l) \in \mathbb{E}$) if and only if every permutation $f \in G$ is an automorphism of $\mathscr{E}$.

Notice that every $f \in G$ has an inverse, which is also an automorphism; our definition of an automorphism group can prove that $f \in G$ is an automorphism for an event structure $\mathscr{E}$ if and only if f satisfies the following condition: $\forall e_1, e_2 \in E$: $e_1 \preceq e_2 \Leftrightarrow f(e_1) \preceq f(e_2)$; $e_1 \sharp e_2 \Leftrightarrow f(e_1) \sharp f(e_2)$; and $l(e_1) = l(f(e_2))$.

6.2. Quotient Model of an Event Structure. The symmetric quotient model for an event structure is a structural reduced model.

Let G be a permutation group acting on the set E and $e \in E$; then the orbit of e is the set $\theta(e) = \{d \mid \exists g \in G : f(e) = d\}$. From each orbit $\theta(e)$ we pick a representative that is called $\mathrm{rep}(\theta(e))$. Intuitively, the quotient model can be obtained by collapsing all the events to orbits.

Definition 43 (symmetric quotient model). Let $\mathscr{E} = (E, \preceq, \sharp, l) \in \mathbb{E}$ and let G be an automorphism group on the event set E of the event structure $\mathscr{E}$. The *symmetric quotient model* $\mathscr{E}_G = (E_G, \preceq_G, \sharp_G, l_G)$ is defined as follows.

(1) The event set is $E_G = \{\theta(e) \mid e \in E\}$, the set of orbits of the events in E.

(2) The causality relation $\preceq_G$ is given by $\preceq_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in \preceq\}$ and the inverse of $\preceq_G$ is denoted by $\succeq_G$.

(3) The conflict relation $\sharp_G$ is given by $\sharp_G = E_G \times E_G \setminus (\prec_G \cup \succ_G \cup \ \mathrm{co}_G \cup \{(e'', e'') \mid e'' \in E_G\})$, where $\mathrm{co}_G = \{(\theta(e_1), \theta(e_2)) \mid \mathrm{co} = E \times E(\preceq \cup \succeq \cup \sharp \cup (e', e') \mid e' \in E), (e_1, e_2) \in E\}$.

(4) The labelling function l_G is given by $l_G(\theta(e)) = l(\mathrm{rep}(\theta(e)))$.

An automorphism group G of an event structure $\mathscr{E} = (E, \preceq, \sharp, l)$ is an invariance group for an action $a \in Act$ if and only if the following condition holds: $\forall f \in G, e \in E : l(\theta) = a \Leftrightarrow l(f(\theta)) = a$.

We then say that a is an invariant under G. Thus, if G is an invariance group for all actions in the action set Act of $\mathscr{E}$ and $\mathscr{E}_G$ is the symmetric quotient model for $\mathscr{E}$; we can directly have $\forall a \in \text{Act}, e \in E : l(\theta) \Leftrightarrow l_G(\theta(e)) = l(\text{rep}(\theta(e)) = a$.

From the above definition, we have that the symmetric quotient model is still a prime event structure and preserves all the causal dependence and independence relations of the original, but the conflict relations are reduced. Note that every two different events in an orbit are exactly in conflict with each other. The quotient model can preserve all the behavior of the original one.

6.3. Symmetry Reduction Algorithm. Based on previous discussion, we have Algorithm 4 for symmetry reduction. In this algorithm, replicated substructure will be removed and only one copy will be kept.

Firstly, cfp operator will be applied on a given event structure to get a set of conflict-free substructures.

Then, for any two elements of them, a checking procedure will be performed to remove the redundant one.

Finally, wcc will be applied on the substructure set to get the reduced model. Detailed pseudocode description is shown in Algorithm 4.

6.4. An Example for Symmetry Reduction. In this section, an example will be discussed to demonstrate the reduction process based on Algorithm 4.

The example of a prime event structure $\mathscr{E}$ with five events (e_1, e_2, e_3, e_4, and e_5) and its semantics in terms of families of configurations is given in Figures 9(1) and 9(2), respectively.

The action-labelling (action set: $Act = \{a, b, c, d\}$) function is defined as follows: $l(e_1) = a$, $l(e_2) = b$, $l(e_3) = c$, $l(e_4) = b$, and $l(e_5) = d$.

In this structure, we have $e_3 \preceq e_5$, $e_2 \sharp e_4$, $e_3 \sharp e_2$, and $e_3 \sharp e_4$. We also have $e_5 \sharp e_2$, $e_5 \sharp e_1$, and $e_5 \sharp e_4$(due to conflict inheritance).

Then event set is $E_G = \{e_1, e_2, e_3, e_4, e_5\}$ and we can construct a permutation group G on the set E_G where two permutations are as follows:

$$\begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 \\ e_1 & e_2 & e_3 & e_4 & e_5 \end{pmatrix} \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 \\ e_1 & e_4 & e_3 & e_2 & e_5 \end{pmatrix}. \quad (3)$$

Obviously, the group G is an invariance group for all actions in $Act = \{a, b, c, d\}$. We then have the orbits of events in being $\theta(e_2) = \theta(e_4) = \{e_2, e_4\}$, $\theta(e_1) = \{e_1\}$, $\theta(e_3) = \{e_3\}$, and $\theta(e_5) = \{e_5\}$. Thus, the symmetric quotient model can be described as shown in Figure 10.

We can get weak conflict set of the event structure $\mathscr{E}$ by cfp operator as follows: $cfp(\mathscr{E}) = \{\mathscr{E}_1, \mathscr{E}_2, \mathscr{E}_3\}$, where

$$\begin{aligned} \mathscr{E}_1 &= (E_1, \preceq_1, \sharp_1, l_1), \\ \mathscr{E}_2 &= (E_2, \preceq_2, \sharp_2, l_2), \\ \mathscr{E}_3 &= (E_3, \preceq_3, \sharp_3, l_3), \\ E_1 &= \{e_1, e_2\}, \\ E_2 &= \{e_1, e_4\}, \\ E_3 &= \{e_1, e_3, e_5\}. \end{aligned} \quad (4)$$

Thus, we have $E_1 \sharp^w E_2$, $E_1 \sharp^w E_3$, and $E_3 \sharp^w E_2$ (or, $\mathscr{E}_1 \sharp^w \mathscr{E}_2$, $\mathscr{E}_1 \sharp^w \mathscr{E}_3$, and $\mathscr{E}_3 \sharp^w \mathscr{E}_2$).

By definition, we have that $\mathscr{E}_1$ and $\mathscr{E}_2$ are symmetric. According to the symmetry reduction algorithm, we will remove replicated events but keep the common or representation ones. The substructure $\mathscr{E}_2$ is removed. The resulted substructure set consists of two elements: $\mathscr{E}_1$ and $\mathscr{E}_3$, because $\mathscr{E}_1$ and $\mathscr{E}_3$ are weak conflict sets of $\mathscr{E}$ and wcc operator can be applied on them to construct a new event structure: $\mathscr{E}' = wcc(\mathscr{E}_1, \mathscr{E}_3)$. Thus, the symmetric reduced event structure can be described by Figure 11.

To illustrate the advantage for properties checking by using symmetry reduction as shown in the above example, consider the states in Figure 9, 9 states in total.

Suppose we want to check whether a property $EF(p)$ holds or not; that is, there exists a global state that satisfies φ. Without symmetry reduction applied, we have to examine all global states, 9 states in total. But with symmetry reduction, as shown in Figure 11 or Figure 10 to decide whether the property holds, only 7 states should be checked and a considerable saving can be achieved.

7. Mathematical Framework

In this section, we will provide a unified mathematical framework for slicing and symmetry reduction based on cfp and wcc operators.

Firstly, we will review some basic definitions in this section. Here, we refer the reader to [14] for details. Next, we will introduce the single action transitions [3] for event structures. Finally, we will discuss the related theories for slicing and symmetric reduction and establish the unified framework.

7.1. Basic Definitions

Definition 44. For any event structure $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$, define $\mathscr{L}[\mathscr{E}] = (\text{Conf}F(\mathscr{E}), \subseteq)$. Especially, for any conflict-free event structure $\mathscr{F} = (E_{\mathscr{F}}, \preceq_{\mathscr{F}}, \emptyset, l_{\mathscr{F}}) \in \mathbb{F}$, define $\mathscr{L}[\mathscr{F}] = (\text{Conf}F(\mathscr{F}), \subseteq)$.

In fact, $\mathscr{L}[\mathscr{F}]$ is the partial order of left-closed and conflict-free subsets of $E_{\mathscr{E}}$ ordered by set inclusion.

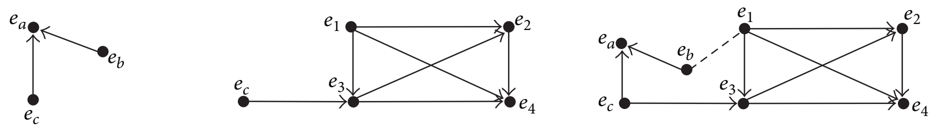
$\mathscr{L}[\mathscr{E}]$ is the partial order of left-closed subsets of $E_{\mathscr{F}}$ ordered by set inclusion.

Definition 45 (single action transition). Let $\mathscr{E} \in \mathbb{E}$. A transition $X \xrightarrow{a}_{\mathscr{E}} X'$ is called a *single action transition* if and only if $a \in Act, X, X' \in \text{Conf}F(\mathscr{E}), X \subseteq X'$ and there exists an event e such that $X' - X = e$ with $l_{\mathscr{E}}(e) = a$.

Here, $X \xrightarrow{a}_{\mathscr{E}} X'$ indicates that in the event structure the state represented by the configuration X may evolve into a state represented by the configuration by performing

```
Input: a prime event structure $\mathscr{E}$;
Output: the reduced model of the event structure: $\mathscr{E}_{sr}$;
BEGIN
(1)  $C = \emptyset; i = 1; n = 0; B = \emptyset; \mathscr{B} = \emptyset$;
     /* Step One: partition via $cfp$ operator, we will get all maximal conflict-free sub-structure of an event structure */
(2)    $B = \mathscr{B}^{mcfES}(\mathscr{E}) = cfp(\mathscr{E})$;
(3)    $n = mcfsetN(\mathscr{E})$;
     /* Step Two: automorphism checking for sub-structure */
(4)   for ($i = 0; i \leq n; i$++) {
(5)       for ($j = 0; i \leq i; j$++) {
     /* Step Two-1: get action set of each maximal conflict-free sub-structure from $B$ */
(6)        $L_{Act}(E_i) = \{a \in Act \mid \exists e \in E_i : l(e) = a\}$
(7)        $L_{Act}(E_j) = \{a \in Act \mid \exists e \in E_j : l(e) = a\}$
     /* $IsNotIdentical$: checking two sets are are identical or not */
     /* $|S|$: the element amount of the set $S$ */
     /* Step Two-2: checking their action sets are identical or not */
(8)        if $((|L_{Act}(E_i| \neq |L_{Act}(E_j)|) \|$
             $IsNotIdentical(L_{Act}(E_i), L_{Act}(E_j))$ {
(9)        break; }
     /* Step Two-3: checking their causal relations are identical or not */
(10)        if $((|\preceq_{\mathscr{E}_i^{\max}}| \neq |\preceq_{\mathscr{E}_j^{\max}}|) \|$
             $IsNotIdentical(\preceq_{\mathscr{E}_i^{\max}}, \preceq_{\mathscr{E}_j^{\max}})$ {
(11)        break; }
     /* Step Two-4: checking their conflict relations are identical or not */
(12)        if $((|\#_{\mathscr{E}_i^{\max}}| \neq |\#_{\mathscr{E}_j^{\max}}|) \|$
             $IsNotIdentical(\#_{\mathscr{E}_i^{\max}}, \#_{\mathscr{E}_j^{\max}})$ {
(13)        break; }
     /* Step Three: automorphism exists, remove the duplicated one and merge for reduction */
(14)        $B = B - \mathscr{E}_j^{\max}$;
(15)        $n = n - 1$;
(16)        } /* end for $j$ */
(17)    } /* end for $i$ */
(18) return $\mathscr{E}_{sr} = wcc(B)$;
END;
```

ALGORITHM 4: Symmetry reduction: $sr(\mathscr{E})$.



(1) $mcfES$slice$(\mathscr{E}_1, \varphi)$ (2) $mcfES$slice$(\mathscr{E}_2, \varphi)$ (3) Slice$ES(\varepsilon, \varphi)$

FIGURE 8: Slice model with respect to φ.

the action a. This transition relation associates a labelled system based on single action transitions with each event structure.

Definition 46 (trace). Let $\mathscr{E} \in \mathbb{E}$. A word $w = a_1 \ldots a_n \in Act^*$ is called *trace* of $\mathscr{E}$ if and only if $\exists X_0, \ldots, X_n \in \text{Conf}F(\mathscr{E})$: $X_0 = \emptyset$ and $X_{i-1} \xrightarrow{a_i} X_i$, $i = 1, \ldots, n$.

Here, let $\text{Trs}(\mathscr{E})$ denote the set of all traces of $\mathscr{E}$.

Definition 47 (interleave trace equivalence). Let $\mathscr{E}_1, \mathscr{E}_2 \in \mathbb{E}$. $\mathscr{E}_1$ and $\mathscr{E}_2$ are called *interleave trace equivalence* ($\mathscr{E}_1 \approx_{it} \mathscr{E}_2$) if and only if $\text{Trs}(\mathscr{E}_1) = \text{Trs}(\mathscr{E}_2)$.

Definition 48 (interleaving bisimulation). Let $\mathscr{E}_1, \mathscr{E}_2 \in \mathbb{E}$. A relation $R \subseteq C(\mathscr{E}_1) \times C(\mathscr{E}_2)$ is called *interleaving bisimulation*

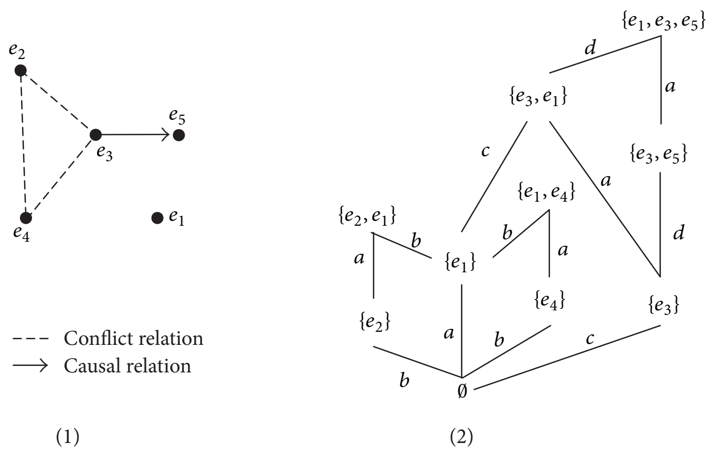


FIGURE 9: An event structure and its family of configurations.

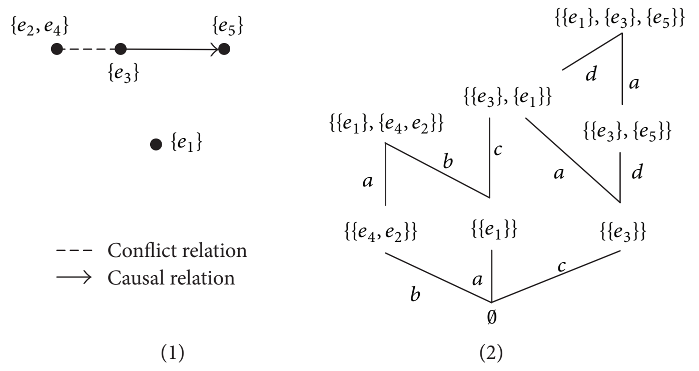


FIGURE 10: Symmetric quotient model of $\mathscr{E}$: $\mathscr{E}_G$.

between $\mathscr{E}_1$ and $\mathscr{E}_2$ if and only if $(\emptyset, \emptyset) \in R$ and $(X, Y) \in R$; then

(1) $X \xrightarrow{a}_{\mathscr{E}_1} X',\ a \in Act \Rightarrow \exists Y' : Y \xrightarrow{a}_{\mathscr{E}_2} Y' \wedge (X', Y') \in R$;

(2) $Y \xrightarrow{a}_{\mathscr{E}_2} Y',\ a \in Act \Rightarrow \exists X' : X \xrightarrow{a}_{\mathscr{E}_1} X' \wedge (X', Y') \in R$.

Definition 49 (interleaving bisimulation equivalent). Let $\mathscr{E}_1, \mathscr{E}_2 \in \mathbb{E}$. $\mathscr{E}_1$ and $\mathscr{E}_2$ are called *interleaving bisimulation equivalent* ($\mathscr{E}_1 \approx_{ib} \mathscr{E}_2$) if and only if there exists an interleaving bisimulation between $\mathscr{E}_1$ and $\mathscr{E}_2$.

7.2. Unified Framework. The unified framework for slicing and symmetry reduction we have set up can be pictured as in Figure 12.

(1) Isomorphism and Equivalence. We are concerned with the translation of concepts and ideas from one side to the other. The following theorems hold.

Theorem 50. *Let $\mathscr{F}$ be a conflict-free event structure, $\mathscr{F} = (E_{\mathscr{F}}, \preceq_{\mathscr{F}}, \emptyset, l_{\mathscr{F}}) \in \mathbb{F}$; then $\mathscr{F} \cong \wp[\mathscr{L}[\mathscr{F}]]$. Similarly, let $P = (D, \sqsubseteq)$ be a prime algebraic complete lattice; then $P \cong \mathscr{L}[\wp[P]]$.*

Theorem 51. *Let $\mathscr{E}$ be a prime event structure, $\mathscr{E} = (E_{\mathscr{E}}, \preceq_{\mathscr{E}}, \sharp_{\mathscr{E}}, l_{\mathscr{E}}) \in \mathbb{E}$; then $\mathscr{E} \cong \wp[\mathscr{L}[\mathscr{E}]]$. Similarly, let $P = (D, \sqsubseteq)$ be a prime algebraic coherent domain; then $P \cong \mathscr{L}[\wp[P]]$.*

Clearly, from Theorems 50 and 51, we have the following.

(1) For any prime event structure $\mathscr{E}$, we have that $\mathscr{E} \cong \wp[\mathscr{L}[\mathscr{E}]]$.

(2) Similarly, for any partial order $P = (D, \sqsubseteq)$, P is a prime algebraic complete lattice or a finitary coherent prime algebraic domain; we have that $P \cong \mathscr{L}[\wp[P]]$.

From Theorems 23, 24, and 25, we have that conflict-free event structures and prime algebraic complete lattices are equivalent; this implies that there is a one-to-one correspondence between a prime event structure and its family of configurations. Similarly, prime event structures and finitary coherent prime algebraic domains are also equivalent; one can be used to represent the other.

(2) Mutual Inverse Operation. For any prime event structure, *cfp* and *wcc* are mutually inverse operators.

We can get the full set of maximal conflict-free substructures of a prime event structure by *cfp* operator.

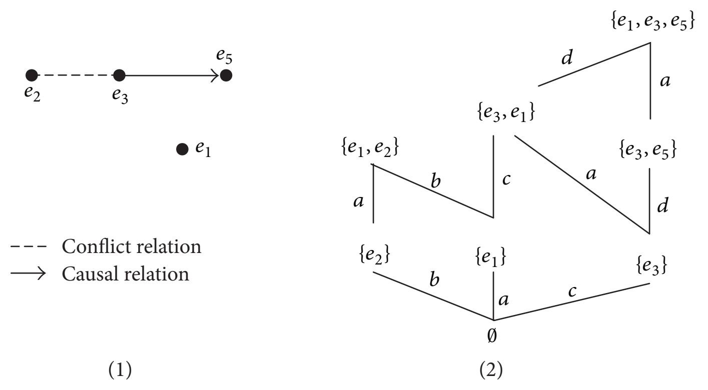


FIGURE 11: Reduced model: $\mathscr{E}'$.

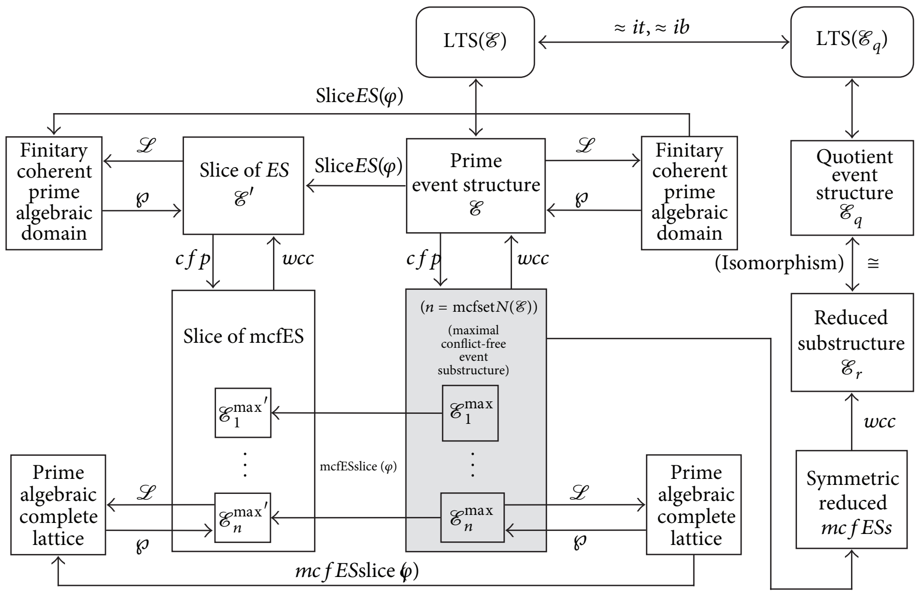

LTS($\mathscr{E}$)
$\approx it, \approx ib$
LTS($\mathscr{E}_q$)
Slice$ES(\varphi)$
Finitary coherent prime algebraic domain
Slice of ES $\mathscr{E}'$
Prime event structure $\mathscr{E}$
Quotient event structure $\mathscr{E}_q$
$\mathscr{L}$
$\wp$
cfp
wcc
(Isomorphism) $\cong$
Slice of mcfES
(n = mcfset$N(\mathscr{E})$)
(maximal conflict-free event substructure)
Reduced substructure $\mathscr{E}_r$
$\mathscr{E}_1^{\max'}$
$\mathscr{E}_n^{\max'}$
$\mathscr{E}_1^{\max}$
$\mathscr{E}_n^{\max}$
mcfESslice (φ)
Prime algebraic complete lattice
Symmetric reduced $mcfESs$
$mcfES$slice (φ)

FIGURE 12: Unified mathematical framework.

Conversely, given the full set of maximal conflict-free substructures of the prime event structure, we can recover the original prime event structure by wcc operator.

(3) Slicing Reduction. Firstly, compared with traditional computation slicing, Figure 12 demonstrates the translation between a conflict-free event structure (or its slice) and prime algebraic complete lattice. This forms the theoretical basis of computation slicing technique proposed by Garg and Mittal, Sen [7, 20].

Secondly, Figure 12 also demonstrates the translation between a prime event structure (or its slice) and finitary coherent prime algebraic domain, which serves as the theoretical basis of our event structure slicing with conflict. A pair of mutually inverse operators, cfp and wcc, act as a link between the two parts.

Thirdly, the slice of a prime event structure can be computed by the following steps:

(1) applying cfp operator to partition the original prime event structure into a full set of maximal conflict-free substructures;

(2) for each maximal conflict-free substructure, applying traditional computation slicing algorithm based on Adding Edges Theorem [7, 8, 20] over its directed graph representation to compute the slice with respect to the given predicate;

(3) applying wcc operator to compose the resulted slices in above step (2) and form a new prime event structure, while the generated event structure is the slice of the original prime event structure.

(4) Symmetry Reduction. Operators cfp and wcc can also play an important role in symmetry reduction.

Symmetry reduction of a prime event structure can be performed by the following steps:

(1) applying cfp operator to partition the original prime event structure into a full set of maximal conflict-free substructures;

(2) checking automorphism among the produced substructures and checking causal relation and action set;

(3) removing duplicated substructure and applying *wcc* operator to compose the resulted structure to form a newly generated prime event structure, which will be symmetry reduced prime event structure.

(5) Trace Equivalence. The relation between original event structure and its quotient model can be specified by Theorems 52 and 53 (See [3]).

Theorem 52. *Let $\mathscr{E} \in \mathbb{E}$ and $\mathscr{E}_G$ be quotient structure of $\mathscr{E}$; then $\mathscr{E}_G \approx_{it} \mathscr{E}$.*

Theorem 53. *Let $\mathscr{E} \in \mathbb{E}$ and let $\mathscr{E}_G$ be the symmetric quotient model for $\mathscr{E}$. Then $\mathscr{E} \simeq_{ib} \mathscr{E}_G$.*

Generally, given an event structure $\mathscr{E}$, in fact, its behavior is exhibited by the labelled transition system (LTS, for short) $\text{LTS}_{\mathscr{E}} = \{\text{Conf}F(\mathscr{E}), Act_{\mathscr{E}}, T_{\mathscr{E}}, \emptyset\}$, where

(1) the configurations $\text{Conf}F(\mathscr{E})$ are states;

(2) the set of labels is the set of actions $Act_{\mathscr{E}}$;

(3) the transitions $T_{\mathscr{E}}$ are single action transitions between every two configurations of $\mathscr{E}$; namely, $T_{\mathscr{E}} \subseteq \text{Conf}F(\mathscr{E}) \times Act_{\mathscr{E}} \times \text{Conf}F(\mathscr{E})$;

(4) the initial configuration (the empty set $\emptyset$) is the initial state.

The above equivalence is based on labelled transition systems whose transitions are single action transitions. As shown in Figure 12, we construct labelled transition system for prime event structure and its quotient model, we will have that their LTSs are interleaving bisimulation and interleave trace equivalence also. Therefore, the following corollary holds.

Corollary 54. *Let $\mathscr{E} \in \mathbb{E}$ and $\mathscr{E}_G$ let be the symmetric quotient model for $\mathscr{E}$. If $LTS = (ConfF(\mathscr{E}), Act, \rightarrow_{\mathscr{E}}, \emptyset)$ and $LTS_G = (ConfF(\mathscr{E}_G), Act, \rightarrow_{\mathscr{E}_G}, \emptyset)$ are induced from $\mathscr{E}$ and $\mathscr{E}_G$, respectively, then $LTS \approx_{it} LTS_G$ and $LTS \approx_{ib} LTS_G$ hold.*

Evidently, we have the following theorems.

Theorem 55. *For any $\mathscr{E} \in \mathbb{E}$, its symmetric quotient model $\mathscr{E}_G$ and symmetric reduced model $\mathscr{E}'$ are isomorphism; that is, $\mathscr{E}_G \cong \mathscr{E}'$.*

Proof. It is not difficult to prove it by constructing a bijection between the events and their orbits.

Theorem 56. *For any $\mathscr{E} \in \mathbb{E}$, its symmetric reduced model $\mathscr{E}'$ is a substructure of $\mathscr{E}$; that is, $\mathscr{E}' \lhd \mathscr{E}$.*

Proof. The proof is straightforward.

(6) Technique Combination. Symmetry reduction technique is orthogonal to slicing reduction and can be used in conjunction with slicing. Thus, it is easy to have the following result.

Theorem 57. *For any $\mathscr{E} \in \mathbb{E}$, let φ be a regular predicate; then the following statements hold: $sr(SliceES(\mathscr{E}, \varphi)) = wcc(mcfESslice(cfp(sr(\mathscr{E})), \varphi))$ and $sr(SliceES(\mathscr{E}, \varphi)) = SliceES(sr(\mathscr{E}), \varphi)$.*

Proof. The proof is straightforward.

8. Conclusion

In this paper, we presented a unified mathematical framework for event structure slicing and symmetric reduction. We described the equivalent relationship between original event structure and its maximal conflict-free event substructures. We proposed two mutually inverse operators: conflict-free partition operator and weak choice composition operator. Both symmetry reduction and slicing reduction can be performed by this pair of operators. We also investigated the related properties, translations, and correspondences between event structures and domains. Essentially, slicing over event structure is a high level extension to the traditional computation slicing based on the model with conflict.

Slicing technique can make the verification of program behavior easier by reducing the size of the state space to be analyzed. Symmetry reduce is another powerful structural reduction technique that can also be applied to narrow down state space. Both quotient model and sliced model produced by reduction are often much smaller than the original model. The consequential model can be used to significantly improve the effectiveness of property verification of the original model.

In future work, on the one hand, we will extend our work to other more complicated event structure models, such as flow event structure [34] and bundle event structure [35, 36]. On the other hand, we hope to implement and test our approach on various verification tools in practice. In addition, we would like to exploit more possible applications and theories.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The project is supported by the Fundamental Research Funds for the Central Universities (DUT14QY05) (1600-851025). The authors would like to thank their colleagues for participating in the research. They also appreciate the anonymous reviewers for their helpful comments.

References

[1] S. Alagar and S. Venkatesan, "Techniques to tackle state explosion in global predicate detection," *IEEE Transactions on Software Engineering*, vol. 27, no. 8, pp. 704–714, 2001.

[2] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha, "Exploiting symmetry in temporal logic model checking," *Formal Methods in System Design*, vol. 9, pp. 77–104, 1996.

[3] J. Jiang and J. Wu, "Symmetry and autobisimulation," in *Proceedings of the 6th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT '05)*, pp. 866–870, IEEE Computer Society Press, December 2005.

[4] J.-Z. Wu and H. Fecher, "Symmetric structure in logic programming," *Journal of Computer Science and Technology*, vol. 19, no. 6, pp. 803–811, 2004.

[5] M. Weiser, *Program slices: formal, psychological, and practical investigations of an automatic program abstraction method [Ph.D. thesis]*, University of Michigan, 1979.

[6] M. Weiser, "Programmers use slices when debugging," *Communications of the ACM*, vol. 25, no. 7, pp. 446–452, 1982.

[7] V. K. Garg and N. Mittal, "On slicing a distributed computation," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS '01)*, D. Harel, D. Kozen, and J. Tiuryn, Eds., Dynamic Logic, pp. 322–329, MIT Press, Phoenix, Ariz, USA, 2001.

[8] A. Sen and V. K. Garg, "Formal verification of simulation traces using computation slicing," *IEEE Transactions on Computers*, vol. 56, no. 4, pp. 511–527, 2007.

[9] E. Duesterwald, R. Gupta, and M. L. Soffa, "Distributed slicing and partial re-execution for distributed programs," in *Proceedings of the 5th Workshop on Language and Compilers for Parallel Computing*, pp. 329–337, 1992.

[10] J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds., *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, vol. 354 of *Lecture Notes in Computer Science*, Springer, Berlin, Germany, 1989.

[11] G. Winskel, "Event structures," in *Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II; Proceedings of an Advanced Course, Bad Honnef, September 1986*, W. Brauer, W. Reisig, and G. Rozenberg, Eds., vol. 255 of *Lecture Notes in Computer Science*, pp. 325–392, Springer, Berlin, Germany, 1987.

[12] P. Madhusudan, "Model-checking trace event structures," in *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS '03)*, pp. 371–380, June 2003.

[13] M. Mukund and P. S. Thiagarajan, "An axiomatization of event structures," in *Foundations of Software Technology and Theoretical Computer Science (Bangalore, 1989)*, vol. 405 of *Lecture Notes in Computer Science*, pp. 143–160, Springer, Berlin, Germany, 1989.

[14] R. J. van Glabbeek and U. Goltz, "Refinement of actions and equivalence notions for concurrent systems," Hildesheimer Informatik-Bericht 6/98, University of Hildesheim, 1998.

[15] H. Hermanns and M. Ribaudo, "Exploiting symmetries in stochastic process algebras," in *Proceedings of the European Simulation Multiconference (ESM '98)*, pp. 763–770, SCS Europe, 1998.

[16] M. Nielsen, G. Plotkin, and G. Winskel, "Petri nets, event structures and domains. I," *Theoretical Computer Science*, vol. 13, no. 1, pp. 85–108, 1981.

[17] M. Nielsen, G. Plotkin, and G. Winskel, "Petri nets, event structures and domains. I," *Theoretical Computer Science*, vol. 13, no. 1, pp. 85–108, 1981.

[18] R. Loogen and U. Goltz, "Modelling nondeterministic concurrent processes with event structures," *Fundamenta Informaticae*, vol. 14, no. 1, pp. 39–73, 1991.

[19] A. Sen, J. Bhadra, V. K. Garg, and J. A. Abraham, "Formal verification of a system-on-chip using computation slicing," in *Proceedings of the International Test Conference (ITC '04)*, pp. 810–819, October 2004.

[20] A. Sen, *Techniques for formal verification of concurrent and distributed program traces [Ph.D. thesis]*, The University of Texas at Austin, Austin, Tex, USA, 2004, http://www.library.utexas.edu/etd/d/2004/senma042/senma042.pdf.

[21] E. A. Emerson and A. P. Sistla, "Symmetry and model checking," in *Computer Aided Verification (Elounda, 1993)*, vol. 697 of *Lecture Notes in Computer Science*, pp. 463–478, Springer, Berlin, Germany, 1993.

[22] A. Miller, A. Donaldson, and M. Calder, "Symmetry in temporal logic model checking," *ACM Computing Surveys*, vol. 38, no. 3, article 8, 2006.

[23] G. Winskel, "Event structures with symmetry," in *Computation, Meaning, and Logic: Articles Dedicated to Gordon Plotkin*, vol. 172 of *Electronic Notes in Theoretical Computer Science*, pp. 611–652, Elsevier, Amsterdam, The Netherlands, 2007.

[24] X. Gao, J. Wu, R. Qiao, and J. Chen, "Theory framework for event structure slicing," in *Proceedings of the 13th IEEE Symposium on Computers and Communications (ISCC '08)*, pp. 714–721, IEEE, Marrakech, Morocco, July 2008.

[25] G. Winskel, "An introduction to event structures," in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (Noordwijkerhout, 1988)*, J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds., vol. 354 of *Lecture Notes in Computer Science*, pp. 364–397, Springer, Berlin, Germany, 1989.

[26] G. Winskel, "An introduction to event structures," in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency (Noordwijkerhout, 1988)*, vol. 354 of *Lecture Notes in Computer Science*, pp. 364–397, Springer, Berlin, Germany, 1989.

[27] G. Winskel and M. Nielsen, "Models for concurrency," in *Handbook of Logic in Computer Science*, vol. 4, pp. 1–148, Oxford University Press, New York, NY, USA, 1995.

[28] N. Mittal and V. K. Garg, "On detecting global predicates in distributed computations," in *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, pp. 3–10, Phoenix, Ariz, USA, April 2001.

[29] V. K. Garg, "Algorithmic combinatorics based on slicing posets," *Theoretical Computer Science*, vol. 359, no. 1–3, pp. 200–213, 2006.

[30] E. M. Clarke, E. A. Emerson, and A. P. Sistla, "Automatic verification of finite state concurrent systems using temporal logic," *ACM Transactions on Programming Languages and Systems*, vol. 8, no. 2, pp. 244–263, 1986.

[31] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*, Cambridge Mathematical Textbooks, Cambridge University Press, Cambridge, UK, 1990.

[32] A. Sen and V. K. Garg, "Detecting temporal logic predicates in distributed programs using computation slicing," in *Proceedings of the 7th International Conference on Principles of Distributed Systems (OPODIS '03)*, December 2003.

[33] N. Mittal, A. Sen, V. K. Garg, and R. Atreya, "Finding satisfying global states: all for one and one for all," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS '04)*, Santa Fe, NM, USA, April 2004.

[34] G. Boudol, "Flow event structures and flow nets," in *Semantics of Systems of Concurrent Processes (La Roche Posay, 1990)*, vol. 469 of *Lecture Notes in Computer Science*, pp. 62–95, Springer, Berlin, Germany, 1990.

[35] H. Fecher, M. Majster-Cederbaum, and J. Wu, "Bundle event structures: a revised cpo approach," *Information Processing Letters*, vol. 83, no. 1, pp. 7–12, 2002.

[36] R. Langerak, "Bundle event structures: a non-interleaving semantics for LOTOS," in *Formal Description Techniques V*, M. Diaz and R. Groz, Eds., pp. 331–346, 1992.