Language Inclusion for Finite Prime Event Structures *

Andreas Fellner[1,2], Thorsten Tarrach[1], and Georg Weissenbacher[2]

[1] AIT Austrian Institute of Technology
1210 Vienna, Austria
[2] TU Wien, 1040 Vienna, Austria
andreas.fellner@ait.ac.at

Abstract. We study the problem of language inclusion between finite, labeled prime event structures. Prime event structures are a formalism to compactly represent concurrent behavior of discrete systems. A labeled prime event structure induces a language of sequences of labels produced by the represented system. We study the problem of deciding inclusion and membership for languages encoded by finite prime event structures and provide complexity results for both problems. We provide a family of examples where prime event structures are exponentially more succinct than formalisms that do not take concurrency into account. We provide a decision algorithm for language inclusion that exploits this succinctness. Furthermore, we provide an implementation of the algorithm and an evaluation on a series of benchmarks. Finally, we demonstrate how our results can be applied to mutation-based test case generation.

Keywords: Event Structures, Language Inclusion, Concurrency, Mutation-based Test Case Generation

1 Introduction

Language inclusion is a fundamental problem in computer science which arises in numerous application domains. In its most familiar form the problem is instantiated with regular languages and finite automata [23]; an incarnation frequently

* This work has received funding from the Electronic Component Systems for European Leadership Joint Undertaking under grant agreement No 737459 (project Productive4.0), which receives support from the European Union Horizon 2020 research and innovation program and Germany, Austria, France, Czech Republic, Netherlands, Belgium, Spain, Greece, Sweden, Italy, Ireland, Poland, Hungary, Portugal, Denmark, Finland, Luxembourg, Norway, Turkey, from the by ECSEL Joint Undertaking under the project H2020 737469 AutoDrive — Advancing failaware, fail-safe, and fail-operational electronic components, systems, and architectures for fully automated driving to make future mobility safer, affordable, and end-user acceptable, from the LogiCS doctoral program W1255-N23 of the Austrian Science Fund (FWF) and by the Vienna Science and Technology Fund (WWTF) through the projects Heisenbugs project VRG11-005.

occurring in formal verification and model checking is language inclusion (and intersection, respectively) for ω-regular languages and Büchi automata [9, Chapter 7]. In the latter application, the goal is to check whether a transition system conforms to a specification given in linear temporal logic. One challenge arising in automata-based model checking is that the verification of concurrent systems relies on the explicit construction of a product automaton whose size can be exponential in the number of processes. Partial Order Reduction (POR, see [41, 20] and [9, Chapter 12], for instance) addresses this problem by exploiting independence between transitions to avoid the construction of the full product automaton: the reduction identifies equivalence classes of words in the language (i.e., executions) obtained by reordering commutative edges/transitions [31] and restricts the exploration to representative members of these classes. POR in its simplest form can be used to check reachability and deadlock problems; for checking temporal logic properties only transitions whose labels are "invisible" to the property are assumed to be independent [9, Chapter 12]. This renders the approach impractical for language inclusion if the alphabets of both languages are the same, e.g., when checking whether a modification is language-preserving – a question arising in the applications that motivated our work (see below).

In this paper, we focus on language inclusion for finite, labeled prime event structures, a representation of bounded executions of concurrent systems in which dependence (and independence) of transitions is made explicit. This representation can be exponentially more succinct than finite automata, as shown in Section 4: there are event structures with n events, such that the smallest NFA expressing the same language has at least 2^n states.

We provide an analysis of the computational complexity of checking language membership as well as inclusion between two event structures, showing that the former is NP-complete and the latter is Π_2^p-complete (Section 3). While a similar results to the former was proven earlier for trace languages [3], to the best of our knowledge, the latter result is novel even in the related domains of bounded trace languages and bounded labeled Petri nets.

Besides showing the complexity of the decision problems, we provide a practical decision algorithm for solving event structure language inclusion in Section 4. By finding suitable embeddings of one event structure in another, the algorithm determines whether the language of the former is included in the language of the latter. The algorithm iteratively refines the event structure whenever two labels occur unordered in the former structure but ordered in the latter. Moreover, the algorithm can provide counterexamples to inclusion encoded as event structures representing words that occur in the former language but not in the latter.

Section 5 provides a qualitative analysis of our representation and an experimental evaluation that highlights advantages and disadvantages of event structures in comparison to an automaton-based representation (for which language inclusion is PSPACE-complete).

Our inclusion algorithm decides whether two systems, represented as event structures, have the same behavior in terms of bounded words over a common vocabulary. This scenario arises in a range of applications: refinement or model

checking, where an implementation is compared against a specification; upgrade or regression checking, where a fixed version of a software is compared against the original version; or mutation-based test case generation, where a small modification (or bug) is introduced in code to obtain a "mutant" of the original program, and the counterexample to inclusion then represents a test case which discriminates between mutant and original. We use the latter scenario, which motivated our research on language inclusion, as an exemplary application of our approach in our experiments (Section 5).

2 Preliminaries

In this section we introduce labeled prime event structures. Throughout this work, we assume that every set of labels $\mathcal{X}$ contains a distinct label ε, which denotes the empty symbol. Concatenation of ε to a word does not change the word.

Definition 1 (FLES). *Given a set of labels $\mathcal{X}$, a finite, $\mathcal{X}$-labeled prime event structure (FLES) is a tuple $\mathcal{E} := \langle E, <, \#, h\rangle$ where E is a finite set of events, $< \subseteq E \times E$ is a strict partial order on E, called* causality relation*, $h : E \to \mathcal{X}$ labels every event with an element of $\mathcal{X}$, and $\# \subseteq E \times E$ is the symmetric, irreflexive* conflict relation *that is* closed under $<$*, i.e. for all $e, e', e'' \in E$, if $e\#e'$ and $e' < e''$, then $e\#e''$.*

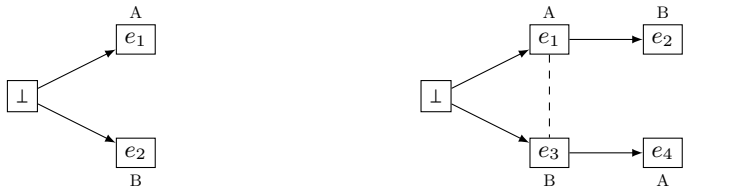
For an event e, we use $\lceil e \rceil$ to denote the history of e as the set of events that must happen before e according to $<$, formally $\lceil e \rceil := \{e' \in E \mid e' < e\}$. We require that there is a special event $\bot \in E$, such that $\lceil \bot \rceil = \varnothing$, for all events $e \in E : \bot < e$, and $h(\bot) = \varepsilon$. We define the direct successors *dsucc* of event e as the set of events that depend on e without there being another event in-between, formally $dsucc(e) = \{e' \in E \mid e < e' \wedge \nexists e'' : e < e'' < e'\}$. We say that two events $e, e' \in E$ are *concurrent* if $e \neq e'$, not $(e < e')$, not $(e > e')$, and not $(e\#e')$.

A central concept in assigning event structures a semantic is the notion of configurations:

Definition 2 (Configuration). *For a FLES $\mathcal{E} := \langle E, <, \#, h\rangle$, a* configuration *of $\mathcal{E}$ is a set of events $C = \{e_1, \ldots, e_n\} \subseteq E$ that is both*

- *Left closed: $\forall e \in C : \forall e' \in E$ such that $e' < e \implies e' \in C$, and*
- *Conflict free: $\forall e, e' \in C : \neg(e\#e')$*

A configuration C is *maximal*, if there is no configuration C' such that $C \subseteq C'$ and $C \neq C'$. We denote by $\mathcal{MC}(\mathcal{E})$ the set of all maximal configurations of an event structure $\mathcal{E}$. A *trace* τ of C is a sequence of events $\langle e_1, \ldots, e_n\rangle$, where every event $e \in C$ occurs exactly once in the sequence and for all $e_i, e_j \in \tau : e_i < e_j \implies i < j$. We denote the set of all traces of a configuration C with $T(C)$. Let $f : C \to X$ be a mapping on C to some set X. For a trace τ of C, we denote by $f(\tau)$ the sequence resulting from point-wise application of f on the elements of



(a) LES with one maximal configuration (b) Event structure with conflicts

Fig. 1: Event structures

τ. Finally, we extend T to event structures by defining it as the union of traces over all maximal configurations. That is, $T(\mathcal{E}) \coloneqq \bigcup_{C \in \mathcal{MC}(\mathcal{E})} T(C)$.

A finite, labeled prime event structure $\mathcal{E}$ represents a finite set of bounded words over an alphabet $\mathcal{X}$, where the bound for the length of words is given by the size of the largest maximal configuration. We call this set the language $\mathcal{L}(\mathcal{E})$.

Definition 3 (Language of C and $\mathcal{E}$). *The language of configuration C of $\mathcal{E}$ is $\mathcal{L}(C) \coloneqq \{h(\tau) \mid \tau \in T(C)\}$. The language of $\mathcal{E}$ is $\mathcal{L}(\mathcal{E}) \coloneqq \{h(\tau) \mid \tau \in T(\mathcal{E})\}$.*

To illustrate this definition we give a small example.

Example 1 (Event structure and configurations). We show two event structures in Figures 1a and 1b. Boxes depict events. Inside every box is its event's identifier, above or below the box is its event's label. If there is no label we implicitly assume the label to be ε. Solid arrows depict direct successors of an event. Dashed lines depict immediate conflicts. Two events e, e' are in immediate conflict if $e \# e'$ and there are no $e_1, e_2 \in E$ such that $e_1 < e \wedge e_1 \# e'$ or $e_2 < e' \wedge e \# e_2$. For better readability, we omit all other causalities and conflicts.

Figures 1a and 1b both represent the language $\{AB, BA\}$. The event structure in Figure 1a has a single maximal configuration consisting of events $\{\perp, e_1, e_2\}$. The event structure in Figure 1b has two maximal configurations: $\{\perp, e_1, e_2\}$ and $\{\perp, e_3, e_4\}$ (due to the conflict between e_1 and e_3 these two events cannot appear in the same configuration).

3 Language Inclusion Problem and Complexity Results

The language inclusion problem for two event structures $\mathcal{E}_1, \mathcal{E}_2$ is to decide whether $\mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)$. In this section we prove a complexity bound for the language inclusion problem. As an intermediate step we look at the membership problem.

3.1 Language Membership is NP-complete

The *finite prime event structure language membership problem* for word w and FLES $\mathcal{E}$ is the problem of deciding whether $w \in \mathcal{L}(\mathcal{E})$. Surprisingly, deciding

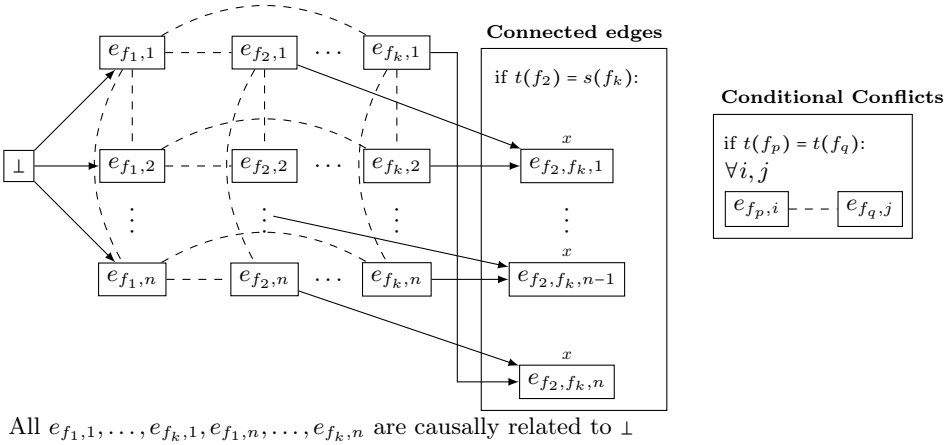


Fig. 2: $\mathcal{E}^G$ for Theorem 2

membership is NP-complete. In contrast, trace membership $\tau \in T(\mathcal{E})$ can be decided in polynomial time. Trace membership can be decided simply by verifying that the set of events of τ forms a maximal configuration of $\mathcal{E}$, which requires to verify left-closure, conflict-freedom, and maximality. All of those can be checked in polynomial time (linear time, assuming linear conflict lookup).

Intuitively, the hardness of language membership comes from the fact that the labeling function does not need to be injective and the role of conflicts, which together rule out a greedy algorithm that consumes the word in question symbol by symbol in a unique way.

Theorem 1. *Finite prime event structure language membership is in NP.*

Proof. Let $\mathcal{E} = \langle E, <, \#, h \rangle$ be an $\mathcal{X}$-labeled FLES and $w = \langle \sigma_1, \ldots, \sigma_n \rangle \in \mathcal{X}^*$ be a word. A trace τ is a polynomially sized certificate for $w \in \mathcal{L}(\mathcal{E})$. Checking that $\tau \in T(\mathcal{E})$ can be done in polynomial time, and checking whether $h(\tau) = w$ can be done in linear time. □

To prove NP-hardness we reduce the Hamiltonian cycle (HC) problem to the membership problem. HC is known to be NP-hard [26]. It is the problem of deciding whether for a directed graph there exists a path that visits all vertices once and that ends in the vertex it started. We use $s(f)$ and $t(f)$ to denote the source and target of a directed edge f.

Theorem 2. *Finite prime event structure language membership is NP-hard.*

Proof. For a directed graph $G = (\{v_1, \ldots, v_n\}, \{f_1, \ldots, f_k\})$ we construct an event structure $\mathcal{E}^G$, such that $x^n \in \mathcal{L}(\mathcal{E}^G)$ iff G has a Hamiltonian cycle. $\mathcal{E}^G$ is shown in Figure 2 and we present the main arguments why this reduction is correct here. A detailed, formal proof is given in Appendix B.

Configurations of the event structure encode a sequence of n edges. If event $e_{f,i}$ is included in the configuration it means that edge f is at position i in the sequence of edges. To ensure that every vertex is visited, edges with the same target are in conflict. Since n edges need to be selected, there are n vertices, and every vertex is a target of some selected edge, every vertex is visited once by the selected edges. To ensure that the sequence of edges actually forms a cycle they need to be connected. Events $e_{f,i}$ and $e_{f',i+1 \bmod n}$ for which the target of f is the source of f' cause an x-labeled event $e_{f,f',i}$. Therefore, only configurations that represent a cycle form the word x^n.

In summary, checking the membership of x^n amounts to checking whether there exists a Hamiltonian cycle in G. The reduction clearly is polynomial. □

3.2 Language Inclusion is Π_2^p-complete

The *finite prime event structure language inclusion problem* for FLES $\mathcal{E}_1$ and $\mathcal{E}_2$ is the problem of deciding whether $\mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)$.

Π_2^p is a complexity class from the polynomial hierarchy. It intuitively represents a $\forall\exists$ quantifier alternation. To show inclusion, we use the definition of Π_2^p given by Wrathall [46], providing semantics for the complexity class in terms of formal languages. These languages should not be confused with the particular type of languages we discuss in this work. In contrast, such languages encode problem instances and candidate witnesses.

Formally, a language L is in Π_2^p iff there exists a polynomially decidable language L', such that $x \in L \Leftrightarrow \forall y_1 \exists y_2 [\langle x, y_1, y_2 \rangle \in L']$. A language L' is polynomially decidable if $w \in L'$ can be decided in polynomial time. The x represents an encoding of the problem instance as a string. The y_1 and y_2 represent string encodings of witnesses to a sub-problem.

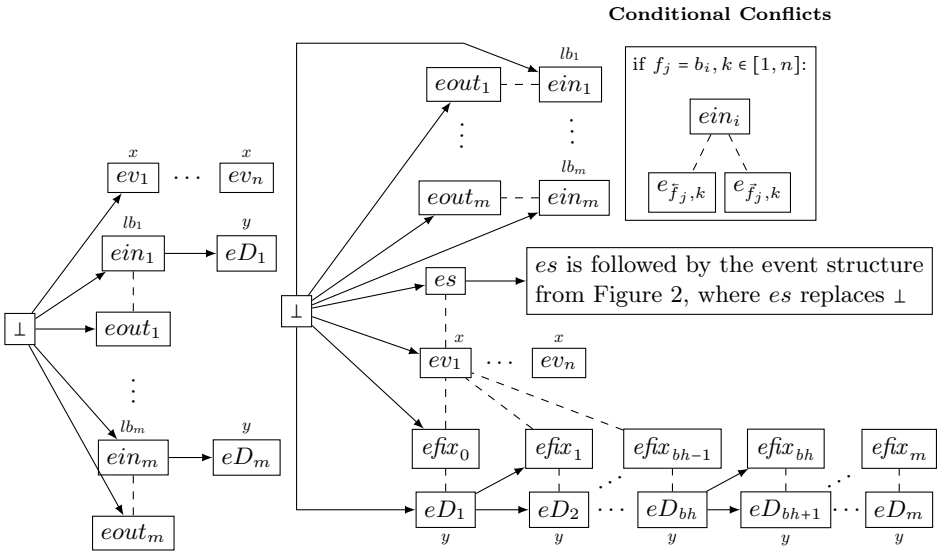
We fix two $\mathcal{X}$-labeled FLES $\mathcal{E}_1 = \langle E_1, <_1, \#_1, h_1 \rangle$ and $\mathcal{E}_2 = \langle E_2, <_2, \#_2, h_2 \rangle$.

Theorem 3. *Finite prime event structure language inclusion is in Π_2^p.*

Proof. Language inclusion $\mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)$ amounts to checking whether $\forall w \in \mathcal{L}(\mathcal{E}_1) \Rightarrow w \in \mathcal{L}(\mathcal{E}_2)$. In terms of traces this can be expressed as $\forall \tau_1 \in T(\mathcal{E}_1).\ \exists \tau_2 \in T(\mathcal{E}_2).\ h_1(\tau_1) = h_2(\tau_2)$, meaning that for every trace in $\mathcal{E}_1$ there has to be a trace in $\mathcal{E}_2$ corresponding to the same word in the common alphabet $\mathcal{X}$.

We define $L := \{\langle \mathcal{E}_1, \mathcal{E}_2 \rangle \mid \mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)\}$ and $L' := \{\langle \langle \mathcal{E}_1, \mathcal{E}_2 \rangle, \tau_1, \tau_2 \rangle \mid \tau_1 \in T(\mathcal{E}_1) \Rightarrow \big(h_1(\tau_1) = h_2(\tau_2) \wedge \tau_2 \in T(\mathcal{E}_2)\big)\}$. By the argument above, we obtain the desired form $x \in L$ iff $\forall y_1 \exists y_2 [\langle x, y_1, y_2 \rangle \in L']$ to show Π_2^p inclusion. Furthermore, L' can be decided deterministically in polynomial time, because trace membership, as well as label equality, can be decided in polynomial time. □

To show Π_2^p hardness, we present a reduction from the Dynamic Hamiltonian Cycle (DHC) problem to the finite prime event structure language inclusion problem. Given an undirected graph $G = (V, F)$ and a set $B \subseteq F$, graph G and B form a DHC if for every set $D \subseteq B$ with $|D| \leq |B|/2$, the graph $G_D = (V, F \setminus D)$ has a Hamiltonian cycle. We define $n := |V|$, $k := |F|$, $m := |B|$, and $bh := \lfloor |B|/2 \rfloor$. Essentially DHC, in comparison to HC, has an additional universal quantifier over subsets of B. DHC is known to be Π_2^p-complete [27].



(a) $\mathcal{E}_1^{G,B}$ (b) $\mathcal{E}_2^{G,B}$

Fig. 3: Event structures for the language inclusion hardness proof. We use ... to indicate omitted events.

Theorem 4. *Finite prime event structure language inclusion is Π_2^p-hard.*

Proof. For an undirected graph $G = (V, F)$ and set $B = \{b_1, \ldots, b_m\} \subseteq F$ we construct event structures $\mathcal{E}_1^{G,B}$ and $\mathcal{E}_2^{G,B}$, such that $\mathcal{L}(\mathcal{E}_1^{G,B}) \subseteq \mathcal{L}(\mathcal{E}_2^{G,B})$ iff G, B satisfy DHC. $\mathcal{E}_1^{G,B}$ and $\mathcal{E}_2^{G,B}$ are shown in Figure 3 and we present the main arguments why this reduction is correct here. A detailed, formal proof, as well as an example, are given in Appendix B.

The idea of the proof is to encode subsets D of B via events ein_i and $eout_i$ with $i \in [1, m]$ in both $\mathcal{E}_1^{G,B}$ and $\mathcal{E}_2^{G,B}$. Events ein_i are labeled with lb_i and represent $b_i \in D$, whereas $eout_i$ are labeled with ε and represent $b_i \notin D$. Furthermore, the cardinality of D is encoded in $\mathcal{E}_1^{G,B}$ via y-labeled events eD_i. In contrast y-labeled events eD_i in $\mathcal{E}_2^{G,B}$ are not used to count $|D|$, but to differentiate whether or not a Hamiltonian cycle is required to show DHC (i.e. whether $|D| \leq \frac{|B|}{2}$). For every $i \in [1, m]$, event $efix_i$ is used to guarantee the existence of maximal configurations in $\mathcal{E}_2^{G,B}$ with i y-labeled events. In case a Hamiltonian cycle is required to show DHC for some set D, we encode G_D using the same event structure as in the proof of Theorem 2, excluding edges from D via conflicts of events ein_i.

Our Hamiltonian cycle encoding used in the proof of Theorem 2 operates on directed edges, but DHC is defined for undirected graphs. Therefore, we replace every edge f in G with two edges in opposing directions, denoted by $\vec{f}$ and $\overleftarrow{f}$.

Clearly, every Hamiltonian cycle in the directed version corresponds to a Hamiltonian cycle in the undirected graph. In order to faithfully represent restricted graphs G_D, we make sure always to exclude all directed edges corresponding to edges in D when looking for a Hamiltonian cycle in G_D.

Every subset D of B is encoded by some word in $\mathcal{L}(\mathcal{E}_1^{G,B})$ via labels lb_i of events ein_i. Since ein_i is in conflict with $eout_i$ maximal configurations can only include one or the other. That is, words exactly enumerate all subsets D of B. Furthermore, similarly as for the proof of Theorem 2, every $w \in \mathcal{L}(\mathcal{E}_1^{G,B})$ contains n times the label x.

Membership of words of $\mathcal{L}(\mathcal{E}_1^{G,B})$ in $\mathcal{L}(\mathcal{E}_2^{G,B})$ only depends on whether the encoded subset $F \setminus D$ induces a Hamiltonian cycle: Words that encode a D such that $|D| > \frac{|B|}{2}$ are always in $\mathcal{L}(\mathcal{E}_2^{G,B})$, because they are trivially accepted by events ev_i. In contrast, for D such that $|D| \leq \frac{|B|}{2}$ the event $efix_{|D|}$ is in concflict with ev_1, thereby preventing a trivial acceptance of words in $\mathcal{L}(\mathcal{E}_1^{G,B})$. Therefore, x labels of words in $\mathcal{L}(\mathcal{E}_2^{G,B})$ that encode D such that $|D| \leq \frac{|B|}{2}$ must be of events caused by es. The events caused by es exactly encode Hamiltonian cycles in G_D, similarly to the proof of Theorem 2.

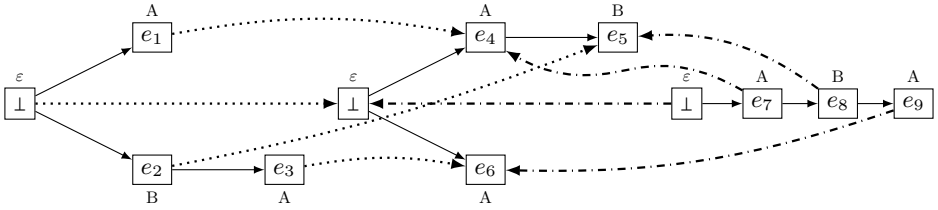
Since the two cases are exhaustive and cover every subset D of B, we get $\mathcal{L}(\mathcal{E}_1^{G,B}) \subseteq \mathcal{L}(\mathcal{E}_2^{G,B})$ iff G and B satisfy DHC. The reduction is polynomial as can be easily observed by the event structures in Figure 3. □

4 Deciding Language Inclusion

In this section, we introduce a decision algorithm for the FLES language inclusion problem. Furthermore, we provide a language preserving translation of event structures into non-deterministic finite automata (NFAs), which allows us to compare our algorithm to NFA language inclusion. We start by introducing necessary concepts for our decision algorithm.

Configuration as an event structure Given an event structure $\mathcal{E} = \langle E, <, \#, h \rangle$ and a configuration C, we denote its corresponding event structure as $\mathcal{E}^C := \langle C, <_{\lceil C \times C}, \varnothing, h_{\lceil C} \rangle$, where $X_{\lceil Y}$ denotes the restriction of X to Y. For ease of presentation, when describing our algorithms, we abuse notation and do not differentiate between a configuration and its corresponding event structure. Furthermore, in the following presentation of the algorithms, we assume that the causality relations and labeling functions of configurations C_i for $i \in \{1, 2\}$ are implicitly given by the event structure interpretation over $\mathcal{E}_i$.

ε-free configurations For every configuration C, there is a configuration with the same language whose only ε-labeled event is $\perp$. This ε-free configuration can be obtained simply by removing all ε-labeled events besides $\perp$ from its corresponding event structure, in particular from C and $<_{\lceil C \times C}$. The resulting ε-free configuration has the same language as the initial configuration, because the causality relation is transitive. Furthermore, ε-labeled events do not modify the words



(a) Structure 1 (b) Structure 2 (c) Structure 3

Fig. 4: Necessary and sufficient embeddings.
$\varphi : \perp \mapsto \perp; e_1 \mapsto e_4; e_2 \mapsto e_5; e_3 \mapsto e_6$ is a necessary embedding (dotted arrows).
$\varphi : \perp \mapsto \perp; e_7 \mapsto e_4; e_8 \mapsto e_5; e_9 \mapsto e_6$ is a sufficient embedding (dash-dotted arrows).

and thus removing them does not influence the language of the configuration. Therefore, in order to improve readability, from hereon we assume without loss of generality that configurations are ε-free. We keep the $\perp$ event to improve readability, even though for our purpose this event is not required neither. Note that ε-labeled events are useful during the construction phase of the event structure representing, for example, hidden transitions or non-deterministic choices.

Embeddings An embedding is a structure-preserving one-to-one mapping between events of two configurations from different event structures. We consider two different types of embeddings that vary in their strictness in terms of structure preservation. Since embeddings are defined between configurations, conflicts do not play a role in these considerations. In order to use these embeddings for deciding language inclusion between two FLES, we assume that in a step prior to searching for embeddings, the maximal configurations of both $\mathcal{E}_1$ and $\mathcal{E}_2$ are computed. This can, for example, be achieved with the algorithm presented in [38].

In the following we consider two configurations C_1 and C_2 of two $\mathcal{X}$-labeled FLES $\mathcal{E}_1 = \langle E_1, <_1, \#_1, h_1 \rangle$ respectively $\mathcal{E}_2 = \langle E_2, <_2, \#_2, h_2 \rangle$.

Definition 4 (Necessary Embedding). *A mapping $\varphi : C_1 \to C_2$ is a* necessary embedding *if A) φ is bijective, B) $\forall e \in C_1 : h_1(e) = h_2(\varphi(e))$, and C) $\forall e \in C_1 : \neg\big(e(<_1 \cup <_2^{\varphi})^+ e\big)$, where $.^+$ denotes transitive closure and $<_2^{\varphi}$ denotes the relation $<_2$ mapped to the events of C_1. Formally $<_2^{\varphi} := \{(\varphi^{-1}(e_1), \varphi^{-1}(e_2)) \mid \exists e_1, e_2 \in C_2.\ e_1 <_2 e_2\}$. For a necessary embedding φ from C_1 to C_2, we write $C_1 \sim_N^{\varphi} C_2$. We write $C_1 \sim_N C_2$ if there exists a necessary embedding φ such that $C_1 \sim_N^{\varphi} C_2$.*

A necessary embedding implies that the two configurations have a common word, by requiring they have the same number of events with the same labels and that their partial orders are not contradicting each other. Note that the relation $\sim_N$ is symmetric, since for a necessary embedding $\varphi : C_1 \to C_2$, φ^{-1} is a necessary embedding from C_2 onto C_1

Example 2. Consider the configurations in Figures 4a and 4b. There are only two label-preserving bijections between the configurations: $\varphi_1 : \bot \mapsto \bot; e_1 \mapsto e_6; e_2 \mapsto e_5; e_3 \mapsto e_4$ and $\varphi_2 : \bot \mapsto \bot; e_1 \mapsto e_4; e_2 \mapsto e_5; e_3 \mapsto e_6$.

The mapping φ_1 is not a necessary embedding, since $e_2(<_1 \cup <_2^{\varphi})e_2$, which violates C). To see this, consider the chain of events $e_2 <_1 e_3 <_2^{\varphi} e_2$, where $e_3 = \varphi^{-1}(e_4), e_2 = \varphi^{-1}(e_5)$, and $e_4 <_2 e_5$. In contrast, φ_2 is a necessary embedding and a witness to the common word ABA of both configurations.

Lemma 1. *Let C_1 and C_2 be maximal configuration of FLES $\mathcal{E}_1$ respectively $\mathcal{E}_2$. $C_1 \sim_N C_2$ if and only if $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \varnothing$.*

The following corollary gives rise to a termination criterion of the decision algorithm. If we find a configuration C in $\mathcal{E}_1$, such that there exists no configuration in $\mathcal{E}_2$ that shares a word with C, we can abort the search and report non-inclusion.

Corollary 1. *Let $C, C_1, \ldots, C_n$ be configurations such that $C \neq \varnothing$. If $(\forall i = 1, \ldots, n : C_i \not\sim_N C)$ then $\mathcal{L}(C) \nsubseteq \bigcup_{i=1}^{n} \mathcal{L}(C_i)$.*

The second type of embedding has a stronger requirement on structure preservation. Intuitively, it requires that the source of such an embedding is at least as strict in terms of causality as the target.

Definition 5 (Sufficient Embedding). *A mapping $\varphi : C_1 \to C_2$ is a* sufficient embedding *if A) φ is bijective, B) $\forall e \in C_1 : h_1(e) = h_2(\varphi(e))$, and C) $\forall e_1, e_2 \in C_1 : \varphi(e_1) <_2 \varphi(e_2) \implies e_1 <_1 e_2$. If there exists a sufficient embedding φ from C_1 to C_2, we write $C_1 \sqsubset_S^{\varphi} C_2$. We write $C_1 \sqsubset_S C_2$ if there exists sufficient embedding φ, such that $C_1 \sqsubset_S^{\varphi} C_2$.*

A sufficient embedding is a witness to language inclusion between configurations. The reason to work with two kinds of embeddings is that we can construct necessary embeddings using a backtracking algorithm. It is easy to check whether a necessary embedding is also sufficient, whereas it is not straight forward to construct a sufficient embedding from scratch.

Example 3. Consider the configurations in Figures 4b and 4c. The mapping $\varphi_1 : \bot \mapsto \bot; e_7 \mapsto e_4; e_8 \mapsto e_5; e_9 \mapsto e_6$ is a sufficient embedding. The only non-trivial causality to check is $e_4 <_2 e_5$, for which we have $\varphi_1^{-1}(e_4) = e_7 <_3 e_8 = \varphi_1^{-1}(e_5)$. In contrast, $\varphi_2 : \bot \mapsto \bot; e_7 \mapsto e_6; e_8 \mapsto e_5; e_9 \mapsto e_4$ is not a sufficient embedding, since in this case $e_4 <_2 e_5$ and $\varphi_1^{-1}(e_4) = e_9 \not<_3 e_8 = \varphi_1^{-1}(e_5)$. This shows that the language of the event structure in Figure 4c is included in language of the event structure in Figure 4b.

The following Lemma provides a connection between sufficient embeddings and language inclusion. In case there exists a sufficient embedding, the respective languages are included.

Lemma 2. *Let C_1 and C_2 be maximal configurations of FLES $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. If $C_1 \sqsubset_S C_2$ then $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$.*

The converse statement is not always true. To see this, consider a configuration $C_1 = \{\bot, e_1, e_1'\}$ such that e_1 and e_1' are concurrent and $h(e_1) = h(e_1') = A$. Furthermore, consider a configuration $C_2 = \{\bot, e_2, e_2'\}$ such that e_2 and e_2' are sequential and $h(e_2) = h(e_2') = A$. Clearly, the configurations have the same language $\{AA\}$. However, there is no sufficient embedding from C_1 to C_2.

Our decision algorithm performs an additional refinement step in such a case and concludes language inclusion only after checking the refined configurations. In Appendix A, we provide a proof that in the case of unique labels, the converse statement also holds.

Splits Our language inclusion decision algorithm continuously performs configuration refinement steps that we call splits. To be precise, we refine the causality relation of its corresponding event structure.

Definition 6 (Split). *Let C be a configuration of event structure $\langle E, <, \#, h\rangle$ and let $e_1, e_2 \in C$ be two concurrent events. The* split *of C on e_1 before e_2 is $C_{e_1<e_2} := \langle C, (< \cup \{(e_1, e_2)\})^+_{\lceil C\times C}, \varnothing, h_{\lceil C}\rangle$ where $.^+$ denotes transitive closure.*

A split on two concurrent events e_1 and e_2 simply adds an additional ordering constraint between the two events. In our algorithm, we always split both ways, creating two new configurations that order concurrent events e_1 and e_2 one way and the other. Note that in order to avoid duplication of events, in practice splits can be implemented via additional, optional causalities on the event structure. The following lemma states that splitting a configuration in both ways produces two new configurations with languages whose union is the original language.

Lemma 3. *Let C be a configuration and $e_1, e_2 \in C$ be concurrent events, then $\mathcal{L}(C) = \mathcal{L}(C_{e_1<e_2}) \cup \mathcal{L}(C_{e_2<e_1})$. If h is injective (labels are unique), then $\mathcal{L}(C_{e_1<e_2}) \cap \mathcal{L}(C_{e_2<e_1}) = \varnothing$.*

The following lemma guarantees progress of our algorithm. It states that if we find a necessary, but not sufficient embedding, there are events that can be used to split C_1. The goal is that after a finite number of splits a sufficient embedding can be established.

Lemma 4. *Let C_1, C_2 be maximal configuration of FLES $\mathcal{E}_1$ respectively $\mathcal{E}_2$. Furthermore, let $C_1 \sim^{\varphi}_N C_2$ and $C_1 \not\sim^{\varphi}_S C_2$. Then there are concurrent events $e, e' \in C_1$, such that $\varphi(e) <_2 \varphi(e')$.*

4.1 Language Inclusion Decision Algorithm

We present our decision algorithm in Algorithm 1. Inputs to the algorithm are finite, labeled prime event structures $\mathcal{E}_1$ and $\mathcal{E}_2$.

The first step of the algorithm is to calculate the maximal configurations of the event structure, which can be done with the algorithm described in [38]. For every maximal configuration C_1 of $\mathcal{E}_1$, the function `Check()` attempts to show that $\mathcal{L}(C_1)$ is a subset of $\mathcal{L}(\mathcal{E}_2)$. This is achieved by searching for sufficient embeddings from (refined versions of) C_1 to maximal configurations of $\mathcal{E}_2$.

Algorithm 1: Language inclusion decision algorithm

Input: Finite, labeled Prime Event Structures $\mathcal{E}_1$ and $\mathcal{E}_2$
Result: $\mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)$
1 $\{C_1^1, \ldots, C_1^n\}, \{C_2^1, \ldots, C_2^m\} \leftarrow$ all maximal configurations of $\mathcal{E}_1$ respectively $\mathcal{E}_2$
2 **return** $\bigwedge_{C_1 \in \{C_1^1, \ldots, C_1^n\}}$ `Check`$(C_1, \{C_2^1, \ldots, C_2^m\})$
3 **Function** `Check`$(C_1, \{C_2^{i_1}, \ldots, C_2^{i_l}\})$**:**
  Result: $\mathcal{L}(C_1) \subseteq \bigcup_{j=1}^{l} \mathcal{L}(C_2^{i_j})$
4 $Candidates \leftarrow \{C_2^{i_1}, \ldots, C_2^{i_l}\}$
5 **foreach** $C_2 \in \{C_2^{i_1}, \ldots, C_2^{i_l}\}$ **do**
6 **if** $\exists \varphi : C_1 \to C_2.\ C_1 \sim_N^{\varphi} C_2$ **then**
7 **return** `SuffOrSplit`$(C_1, C_2, \varphi, Candidates)$
8 **else**
9 $Candidates \leftarrow Candidates \setminus \{C_2\}$
10 **return** `false` ▷ `counter-example` C_1
11 **Function** `SuffOrSplit`$(C_1, C_2, \varphi, Candidates)$**:**
12 **if** $C_1 \sqsubset_S^{\varphi} C_2$ **then**
13 **return** `true`
14 Let $e, e' \in C_1$ be concurrent and $\varphi(e) <_2 \varphi(e')$ ▷ `always exist (Lemma 4)`
15 **return** `SuffOrSplit`$(C_{e<e'}, C_2, \varphi, Candidates) \wedge$
  `Check`$(C_{e'<e}, Candidates)$

In order to construct candidate sufficient embeddings, in Line 6 the algorithm attempts to construct necessary embeddings, using Algorithm 2. In the following line, function `SuffOrSplit()` checks whether a necessary embedding φ is also sufficient. This can be done by checking $\forall e \in C_2 : \forall e' \in dsucc(e) : \varphi^{-1}(e)$ is not concurrent with $\varphi^{-1}(e')$. In case φ is not a sufficient embedding, such a pair of events is guaranteed to exist by Lemma 4. For efficiency, this check can already be done during construction of the necessary embedding.

In case φ is not sufficient, Lemma 4 guarantees the existence of a pair of concurrent events that can be split. The resulting split configurations are recursively checked for language inclusion in Line 15. Lemma 4 guarantees us that φ is a necessary embedding for one of the splits (say $C_{e<e'}$). Therefore, for $C_{e<e'}$ we do not need to construct a new necessary embedding again, but can immediately check whether φ is a sufficient embedding for $C_{e<e'}$.

In case no necessary embedding can be found for some configuration C_1 and its candidates, according to Lemma 1, we can conclude $\mathcal{L}(C_1) \cap \mathcal{L}(\mathcal{E}) = \varnothing$, i.e. all words in C_1 are counter-examples to language inclusion. Once Line 10 is reached we know that C_1 does not share any word with any $\{C_2^1, \ldots, C_2^m\}$, therefore C_1 is a counter-example to language inclusion.

The algorithm terminates, because the notions of necessary and sufficient embedding collapse in case the configuration contains only a single trace, which is the case when the causality relation is a total order on the events of the configuration (see Lemma 6).

Algorithm 2: $\sim_N$ decision algorithm

```
   Input: Configurations C_1, C_2
   Result: φ : C_1 → C_2 if C_1 ~_N^φ C_2, None otherwise
 1 if ∃x ∈ 𝒳. |{e ∈ C_1 | h_1(e) = x}| ≠ |{e ∈ C_2 | h_2(e) = x}| then
 2 |   return false
 3 return NEmbedding({⊥_1}, {⊥_1 ↦ ⊥_2})
 4 NEmbedding(frontier, φ):
   |   Input: frontier stack of events C_1
   |   Input: φ : C_1 → C_2 (partial mapping)
   |   Result: φ : E_1 → E_2 if C_1 ~_N^φ C_2, None otherwise
 5 |   if frontier = ∅ then
 6 |   |   return φ
 7 |   e ← frontier.pop()
 8 |   foreach e' ∈ dsucc_{ℰ^{C_1}}(e) do          ▷ direct successors of e in C_1
 9 |   |   frontier.push(e')
10 |   foreach e'' ∈ C_2 such that h_1(e) = h_2(e'') ∧ e'' ∉ range(φ) do
11 |   |   φ' ← φ ∪ {e ↦ e''}
12 |   |   if ∄e_1, e_2 ∈ E_1. e_1 <_1 e_2 ∧ φ'(e_2) <_2 φ'(e_1) then    ▷ check for cycle
13 |   |   |   φ'' ← NEmbedding(frontier, φ')
14 |   |   |   if φ'' ≠ None then
15 |   |   |   |   return φ''
16 |   return None
```

As the algorithm recursively searches for sufficient embeddings, for efficiency, we can reduce the set of candidate configurations, because in case there is no necessary embedding between two configurations, there is clearly also no necessary embedding between any of their split configurations.

We present the algorithm to construct necessary embeddings in Algorithm 2. Intuitively, the algorithm is a combined depth first search over the causality relation, as well as the space of possible bijective, label-preserving mappings.

The algorithm starts by dismissing configurations that can never have a necessary embedding because the number of events with the same label differs (Line 1). The actual embedding is established with the recursive function `NEmbedding()`. The recursion maintains a frontier of events that are yet to be explored and a partial mapping of already explored events. It ends if the frontier becomes empty (Line 5). The exploration is done on C_1 by adding the successors of the current event e to the frontier (Line 8). Then for every event e'' in C_2 with the same label as e a mapping φ' is created and a cycle check performed. If this mapping does not introduce a cycle we recurs on it (Line 13). The first valid (not `None`) mapping that is returned by a recursion is returned. If no such mapping is found, then `None` is returned. The cycle check in Line 12 basically checks if the two causality relations $<_1$ and $<_2$ are compatible for the mapped events, in the sense that the events can be brought in an order that respects both causality relations. The procedure can be implemented using any of the

well known cycle detection algorithms over the graph with nodes being events of C_1 and edges being causalities $<_1 \cup <_2^{\varphi}$.

The worst-case runtime of the decision algorithm is exponential in $O(2^{2n})$, where $n = |E_1| + |E_2|$, which is not surprising for an algorithm solving a Π_2^p hard problem. There are two dominant factors of the exponential complexity.

First, the number of maximal configurations can be exponential in n and Algorithm 1 potentially has to compare all pairs of maximal configurations. The CCNFS benchmark in Section 5 is an example for an event structure with an exponential number of maximal configurations in n.

Second, the number of mappings between configurations that need to be considered as candidates for necessary embeddings can be exponential in n. That is, algorithm Algorithm 2 has worst case runtime exponential in n. Note that for a fixed mapping, the algorithm performs a linear search over the configuration and the combined causality relation.

Note that the number of possible embeddings decreases with the number of calls to `Check` and the size of maximal configurations decreases relative to n with the number of maximal configurations. Therefore, the amortized runtime should be much better than the worst case complexity.

4.2 Automaton Based Language Inclusion

We provide a language preserving encoding of event structures into non-deterministic finite automata (NFA). The encoding allows us to compare our algorithm to well researched language inclusion algorithms in our evaluation (Section 5).

The encoding has a state for every configuration of the event structure. There is a transition between two states, if the difference between the corresponding configurations is just one event. The transition is labeled with the label of that event. In essence, the encoding is an automaton representation of what is known as the configuration structure of a prime event structure [19].

Definition 7 (Automaton Encoding). *Let $\mathcal{E} = \langle E, <, \#, h\rangle$ be a finite prime event structure with labels $\mathcal{X}$. We define the non-deterministic finite automaton $\mathcal{A}^{\mathcal{E}} = \langle Q^{\mathcal{E}}, \Omega^{\mathcal{E}}, \delta^{\mathcal{E}}, q_0^{\mathcal{E}}, F^{\mathcal{E}}\rangle$ as $Q^{\mathcal{E}} = \{q_C^{\mathcal{E}} \mid C$ is a configuration of $\mathcal{E}\}$, $\Omega^{\mathcal{E}} = \mathcal{X}$, $(q_{C_1}^{\mathcal{E}}, \sigma, q_{C_2}^{\mathcal{E}}) \in \delta^{\mathcal{E}}$ iff there is $e \in E$, such that $C_1 \cup \{e\} = C_2$ and $h(e) = \sigma$, $q_0^{\mathcal{E}} := q_{\{\perp\}}^{\mathcal{E}}$, and $F^{\mathcal{E}} = \{q_C^{\mathcal{E}} \mid C$ is maximal$\}$.*

Lemma 5. *Let $\mathcal{E}$ be a labeled, finite prime event structure, then $\mathcal{L}(\mathcal{E}) = \mathcal{L}(\mathcal{A}^{\mathcal{E}})$.*

The provided encoding is not optimal in general due to conflicts and the fact that events of prime event structures are caused in a unique way, which is a well known caveat of prime event structures [43]. However, for the family of event structures that consists of the $\perp$ event and n concurrent events (c.f. the proof of Theorem 5 in Appendix A), our encoding contains exactly $2^n + 1$ states, which is one state more than the provably optimal NFA accepting the language of the event structure. Furthermore, in our experiments, we apply optimized NFA reduction techniques [30] before checking language inclusion on automata.

Theorem 5. *There is a family of event structures $\mathcal{E}_n$ with events E_n, such that $|E_n| = n + 1$, $|\mathcal{L}(\mathcal{E}_n)| = n!$, and $|Q^{\mathcal{E}}| = 2^n + 1$. Every NFA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{E}_n)$ has at least 2^n states.*

5 Application and Evaluation

Our motivation to investigate event structures and language inclusion was model-based mutation testing. The goal of *model-based testing* (MBT) is to derive test-cases from a model of a system. The model may, for example, be a UML state machine and the test may be a sequence of inputs and outputs of the system. The simplest way of obtaining such test cases would be to randomly explore the state machine and record the produced input/output (IO) sequences. These tests can then be run against an implementation at a later point.

Model-based mutation testing (MBMT) compares the original model to a mutated version of it, where a mutation is a small change in the model, such as removing or adding a transition. A test case is only generated if an observable difference between the original and the mutated model can be witnessed. This form of test case generation can be easily expressed using language inclusion between the two versions of the system: The test is exactly the word that is a member of the mutant, but not of the original.

The application of finite prime event structures to this problem is motivated by three factors. Firstly, models often use concurrent state machine that synchronize rarely. Secondly, mutation analysis on reactive models can be performed by exploring models in bounded segments [14], where a bounded segment refers to all events occurring between two consecutive inputs. These bounded segments can be represented as finite event structures. Thirdly, it is desirable express independence in test cases in order to produce minimal test suites that do not need to list all variations of a test that differ only in terms of independent events. Such test cases can be obtained as counter-examples to language inclusion, as discussed in Section 4.

To this end, we implemented the presented prime event structure language inclusion algorithm in the model-based mutation testing tool MoMuT [14]. MoMuT accepts models written as object-oriented action systems (OOAS). The models can be understood as labeled transition systems, where labels are either observable, controllable or hidden (ε). OOAS models can model highly concurrent systems. In order to construct test cases for such concurrent models efficiently, we need to apply partial order reduction during model exploration. In [38] a partial order reduction based algorithm for constructing labeled prime event structures from transition systems is given. We implemented this algorithm, using a static dependency relation based on variable reads and writes, and use it during model exploration, obtaining finite labeled event structures representing bounded segments of the model. Each segment corresponds to all output or hidden transitions following some input until either a new input is required by the model to progress further or the model is in a terminating state. We operate on

models that exclude infinite sequences of outputs or hidden transitions. Thus, the discussed segments are indeed bounded in our case.

The event structures constructed during partial order reduction are labeled with (potentially hidden) transitions of the explored model. However, for mutation analysis, we want to find observable differences between event structures for given controllable inputs, in contrast to any difference in transition labels. Therefore, in addition to using transition labels during partial order reduction, we use projected, visible (input & output) labels and perform language inclusion on the languages over the latter kind.

During model exploration, which is described in detail in [14], we construct event structures $\mathcal{E}_B$ of the original model and $\mathcal{E}_A$ of mutants, representing bounded segments (as described above) following the same sequence of inputs. We perform language inclusion checks $\mathcal{L}(\mathcal{E}_A) \subseteq \mathcal{L}(\mathcal{E}_B)$ using Algorithm 1 to decide whether the corresponding mutant is killed by the sequence of inputs and a test case can be produced.

In our experimental evaluation, we report measurements of these language inclusion checks during test case generation on a sequence of benchmark models. For comparison and in addition to event structure based language inclusion, we perform language inclusion via automaton encoding, as described in Section 4.2. To this end, we encode the produced event structures as NFAs and check language inclusion using the tool RABIT [30].

5.1 Benchmarks and Results

We use the following benchmarks for our experimental evaluation. All benchmark models, scripts to instantiate the models for any parameter value, and the version of MoMuT used in the experiments can be found in the publicly available artifact of this paper [15], which can be run with the virtual machine provided in [8].

- The **Paxos**(n, m, k) benchmark models the Paxos distributed consensus protocol [28] with n proposers, m acceptors, and k learners. The protocol specifies how the different actors can exchange certain messages to achieve consensus on some proposed value. The actions of the actors are largely independent of each other, which introduces lots of concurrency to the model. Furthermore, test cases extracted from our method should be interesting to concertize and run against implementations of the Paxos algorithm.
- The **Semaphore**(n) benchmark models n threads that are synchronized by a semaphore. Exactly $n-1$ threads are allowed to enter and compute in a critical section at the same time. The amount of parallelism of this model is proportional to n. Furthermore, the model exhibits lots of conflicts, as all operations on the semaphore are in conflict with each other.
- The **ParSum**(n) benchmark models a parallel summation algorithm. The sequence $0, \ldots, n^2-1$ is split into n equally sized chunks, which are summed up concurrently. Then the partial results are summed up centrally when all parallel threads are finished.

Name	$\lvert\mathcal{E}_B\rvert$	**PC**	**Inclusion**		**Non-Incl.**		$\lvert\mathcal{A}^{\mathcal{E}_B}\rvert$	**Inclusion**		**Non-Incl.**	
			Time	Num	Time	Num		Time	Num	Time	Num
Paxos(2,3,1)	80	4.1	-	0	$4.1{\cdot}10^3$	65	23469	TO	TO	TO	TO
Paxos(3,6,1)	716	4.1	-	0	$4.0{\cdot}10^3$	94	TO	TO	TO	TO	TO
Semaphore(3)	9	1.2	-	0	23.8	78	5	-	0	324.0	78
Semaphore(11)	25	2.3	3.5	2	48.9	84	9	-	0	564.2	80
ParSum(3)	18	2.2	1.1	80	170.5	92	218	$2.5{\cdot}10^3$	80	$9.3{\cdot}10^3$	84
ParSum(5)	38	3.8	342.0	84	67.6	94	TO	TO	TO	TO	TO
ParSum(10)	123	8.2	-	0	12.6	23	TO	TO	TO	TO	TO
CCNFS(3)	15	1.7	3.9	88	1.2	88	9	262.7	88	610.9	88
CCNFS(6)	77	2.7	501.8	108	84.0	92	65	379.7	108	$1.2{\cdot}10^3$	92
CCNFS(10)	1045	4.0	$303{\cdot}10^3$	98	$17{\cdot}10^3$	102	59050	TO	TO	TO	TO
AllPar(10)	12	4.0	1.4	56	8.6	144	1025	$10{\cdot}10^3$	56	$82{\cdot}10^3$	144
AllPar(50)	52	17.3	3.2	44	93.3	156	TO	TO	TO	TO	TO
AllPar(500)	502	167.3	361.9	40	$7.7{\cdot}10^3$	160	TO	TO	TO	TO	TO
Sharing(5,20)	111	1.0	7.2	26	4.2	174	23	338.1	26	$1.1{\cdot}10^3$	174
Sharing(50,50)	2601	1.0	$1.9{\cdot}10^3$	38	527.2	162	53	255.4	38	$1.5{\cdot}10^3$	162

Table 1: Benchmark results for language inclusion checks $\mathcal{L}(\mathcal{E}_A) \subseteq \mathcal{L}(\mathcal{E}_B)$ respectively $\mathcal{L}(\mathcal{A}^{\mathcal{E}_A}) \subseteq \mathcal{L}(\mathcal{A}^{\mathcal{E}_B})$.

- The **CCNFS**(n) benchmark models a system with n events and unique labels, such that the $2i'th$ event is in conflict exactly with the $2i{+}1'th$ event. Every set of independent events induces an event resetting the state. This benchmark is interesting, because its number of maximal configurations $2^{\lfloor \frac{n}{2} \rfloor}$ (each configuration contains either $2i$ or $2i{+}1$ for each $i = 1 \ldots \lfloor \frac{n}{2} \rfloor$) is exponential in the number of events n. Due to the high number of maximal configurations, this benchmark is challenging for our algorithm.
- The **AllPar**(n) benchmark models a system with n independent events and unique labels. The benchmark is the ideal case for our algorithm, because its event structure consists of only one maximal configuration with all events in parallel. In contrast, the benchmark is a very bad case for NFA language inclusion, as the smallest NFA to encode all permutations of n symbols is exponential in n (Theorem 5).
- The **Sharing**(n, m) benchmark models a system that has n different prefixes that all share the same suffix of length m. The benchmark particularly exhibits the well known shortcoming of event structures not being able to encode shared causes. The NFA is able to express the common suffix more succinct in comparison to the event structure.

We present the results of our experimental evaluation in Table 1. For every benchmark, we report measurements of language inclusion checks for the largest bounded segment encountered during model exploration. As described above, every such bounded segment corresponds to all output and hidden transitions following some input transition. We report measurements of event structure based language inclusion $\mathcal{L}(\mathcal{E}_A) \subseteq \mathcal{L}(\mathcal{E}_B)$ and automaton based language inclu-

sion $\mathcal{L}(\mathcal{A}^{\mathcal{E}_A}) \subseteq \mathcal{L}(\mathcal{A}^{\mathcal{E}_B})$. We separate the results into the cases where language inclusion holds (**Inclusion**) respectively does not hold (**Non-Incl.**). Column $|\mathcal{E}_\mathbf{B}|$ shows the size of $\mathcal{E}_B$ in terms of the number of its events. Column $|\mathcal{A}^{\mathcal{E}_\mathbf{B}}|$ shows the size of $\mathcal{A}^{\mathcal{E}_B}$ in terms of the number of its states. Columns Num show the number of inclusion checks performed on the respective bounded segment (which is the number of mutants relevant in the segment). Columns Time show the average time for the inclusion checks in milliseconds. Finally, column **PC** shows a measurement of the degree of concurrency. For a single configuration C, the measurement is defined as $|C|/max_{e \in C} depth(e)$ and we report the average measurement of all maximal configurations in the respective event structure.

The reported time for language inclusion of event structures is the time for calculation of the maximal configurations plus the time for the actual language inclusion check. The reported time for language inclusion of automata is the time for the language inclusion check on a pre-reduced automaton. The construction and minimization of the NFAs is not included.

The results show that our language inclusion algorithm performs well on models with a lot of concurrency, i.e. those with high ParCoeff. Furthermore, the automaton translation clearly fails in cases with lots of concurrency that are easy for our method (c.f. the **AllPar** benchmark). For these examples our algorithm is very useful. This result is not surprising, since our method exploits concurrency, whereas the NFA encoding does not include any notion of concurrency. Nevertheless, the result demonstrates that the benefits of exploiting concurrency with our method outweigh optimizations and fine-tuning of a well established language inclusion algorithm that has no notion of concurrency.

However, as the **Sharing** benchmark shows, the inability of prime event structures to encode shared causes of events is a limitation of the approach. In contrast, the reduced automaton representation can be significantly more compact than the event structure representation, rendering the automaton-based language inclusion superior.

6 Related Work

Prime event structures are a widely used formalism to express concurrency of discrete systems [43] that can be obtained from transition systems via the method presented in [38], or its extended version in [34]. There are multiple other variants of event structures, such as stable event structures [43] and flow event structures [5]. Studying language inclusion for these event structure variants is interesting future work.

Event structure containment based on causality and conflict refinement is considered in [43, 44]. However, as we demonstrate in our work, causality preservation is not necessary for language inclusion. In [18, 42] equivalence of event structures under action refinement is investigated. This line of research is orthogonal to our approach, as it considers refinement of event structures, while we compare event structures that can be obtained in multiple different ways.

Moreover, there is almost never language inclusion between an event structure and the event structure with refined actions by design.

Model checking over particular types of event structures has been studied in [36] for event structures labeled with atomic propositions and in [29] for event structures labeled with trace languages. However, the proposed model-checking methods are not based on language inclusion, which is one of the interesting future directions for our research. Instead, formulas are directly interpreted over the event structure.

Several formalisms to express concurrency of discrete systems have been proposed and their relationships have been worked out in [45]. In particular, trace languages and Petri nets are formalisms closely related to event structures, for which languages and language related problems have been studied.

The theory of trace languages [31, 32] studies closure of string languages under independence relations. [6] presents an efficient method to show trace language inclusion over languages defined by non-deterministic finite automata. In [7] decidability results of rational trace languages are studied. In particular, it is shown that language inclusion of rational (closed under union, concatenation, and Kleene-star) trace languages is decidable if and only if the common independence relation is transitive. Language membership for context free and regular trace languages was shown to be NP-complete in [3]. In [4], comparison of concurrent programs via trace languages is studied. The suggested trace languages abstract the program executions by considering statement ordering, as well as read and write accesses on a subset of relevant variables and synchronization primitives. Trace language refinement is then reduced to assertion checking. Interestingly, for Boolean programs this refinement check has complexity Δ_2^P for bounded abstraction precision and Σ_2^P for unbounded abstraction precision. In contrast to arbitrary event labels considered in our work, the authors of [4] consider refinement on languages of a more concrete program and dependency model.

Our problem is orthogonal to trace language inclusion in three aspects. Firstly, we do not assume the independence relations of the compared systems to be equal. Secondly, we do not require the independence relation to be defined over labels. That is, we can study systems where two labels occur concurrently in one place, while the labels occur sequentially in another. This can occur because different events can have the same label. Finally, in contrast to automata, which are often used to define trace languages, event structures are acyclic. Therefore, event structures are less expressive than automata. However, the price of the additional expressivity is that trace language inclusion over automata is undecidable in general [2], whereas our problem is decidable.

Petri nets are a formalism for concurrent systems that is closely related to event structures [35]. A manifold of complexity questions have been studied for Petri nets, see [25, 11, 12] for surveys. In particular, language related problems of labeled Petri nets have been studied, see [37, 17] for an overview over the types of considered languages and complexity results. Since Petri nets typically describe languages on infinite words and many Petri net related problems are

undecidable, complexity results on language related Petri nets problems focus on establishing the boundary between decidability and undecidability. Language inclusion and equivalence were shown to be undecidable for a wide range of types of Petri nets [21, 24, 12]. Language inclusion is decidable for languages of firing of regular Petri nets [40] and certain types of deterministic Petri nets [17]. In contrast, language membership is decidable for a large class of Petri nets and language types [37, 22]. Similarly to trace languages, the additional expressivity of Petri nets over finite prime event structures manifests in increased complexity of solving language inclusion. However, as we demonstrated with our application, finite prime event structures are sufficient for interesting practical problems.

Finite asynchronous automata [13] express concurrent systems succinctly in the same spirit as prime event structures. Furthermore, asynchronous automata accept trace languages [47]. However, to the best of our knowledge, there is neither an algorithm, nor a tool to check language inclusion for (loop free) finite asynchronous automata.

Language inclusion of regular languages is a classic problem of computer science [23, 33, 16, 39]. Algorithms for the problem are well studied and highly optimized [30, 1, 6]. However, as we demonstrate in the evaluation section, our procedure can outperform these algorithms in the realm of highly concurrent systems. Adapting methods for classic automaton based language inclusions to event structures is interesting future work.

7 Conclusion and Future Work

In this paper we showed that the language inclusion problem between two event structures is computationally hard, but our application and evaluation show that there are numerous benchmarks where the use of event structures and their comparison is beneficial. However, the experiments also manifested a well known shortcoming of prime event structures, namely their inability to succinctly encode shared causes of events.

Interesting future work includes adapting our language inclusion method to different variants of event structures that do not suffer this problem. Furthermore, we want to study whether our language inclusion procedure can be used to perform model checking over event structures. Finally, we want to further study the test cases generated for the Paxos distributed consensus algorithm. Concertizing the resulting test cases and running them against an implementation of the protocol might yield interesting results.

References

1. Abdulla, P.A., Chen, Y.F., Holík, L., Mayr, R., Vojnar, T.: When simulation meets antichains. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 158–174. Springer (2010)
2. Bertoni, A., Mauri, G., Sabadini, N.: Equivalence and membership problems for regular trace languages. In: International Colloquium on Automata, Languages, and Programming. pp. 61–71. Springer (1982)

3. Bertoni, A., Mauri, G., Sabadini, N.: Membership problems for regular and context-free trace languages. Information and Computation **82**(2), 135–150 (1989)
4. Bouajjani, A., Enea, C., Lahiri, S.K.: Abstract semantic diffing of evolving concurrent programs. In: Ranzato, F. (ed.) Static Analysis - 24th International Symposium, SAS 2017, New York, NY, USA, August 30 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10422, pp. 46–65. Springer (2017). https://doi.org/10.1007/978-3-319-66706-5_3, https://doi.org/10.1007/978-3-319-66706-5_3
5. Boudol, G.: Flow event structures and flow nets. In: LITP Spring School on Theoretical Computer Science. pp. 62–95. Springer (1990)
6. Černý, P., Clarke, E.M., Henzinger, T.A., Radhakrishna, A., Ryzhyk, L., Samanta, R., Tarrach, T.: From non-preemptive to preemptive scheduling using synchronization synthesis. Formal Methods in System Design **50**(2), 97–139 (Jun 2017)
7. Diekert, V., Métivier, Y.: Partial commutation and traces. In: Handbook of formal languages, pp. 457–533. Springer (1997)
8. Dietsch, D., Jakobs, M.C.: Vmcai 2020 virtual machine (Nov 2019). https://doi.org/10.5281/zenodo.3533104, https://doi.org/10.5281/zenodo.3533104
9. Edmund M. Clarke, J., Grumberg, O., Kroening, D., Peled, D., Veith, H.: Model Checking. MIT Press, 2 edn. (2018)
10. Ellul, K., Krawetz, B., Shallit, J., Wang, M.w.: Regular expressions: New results and open problems. Journal of Automata, Languages and Combinatorics **10**(4), 407–437 (2005)
11. Esparza, J.: Decidability and complexity of Petri net problems - an introduction. In: Reisig, W., Rozenberg, G. (eds.) Lectures on Petri Nets I: Basic Models, Advances in Petri Nets, the volumes are based on the Advanced Course on Petri Nets, held in Dagstuhl, September 1996. Lecture Notes in Computer Science, vol. 1491, pp. 374–428. Springer (1996). https://doi.org/10.1007/3-540-65306-6_20, https://doi.org/10.1007/3-540-65306-6_20
12. Esparza, J., Nielsen, M.: Decidability issues for Petri nets - a survey. Bulletin of the EATCS **52**, 244–262 (1994)
13. Fates, N.: A guided tour of asynchronous cellular automata. In: International Workshop on Cellular Automata and Discrete Complex Systems. pp. 15–30. Springer (2013)
14. Fellner, A., Krenn, W., Schlick, R., Tarrach, T., Weissenbacher, G.: Model-based, mutation-driven test-case generation via heuristic-guided branching search. ACM Trans. Embed. Comput. Syst. **18**(1), 4:1–4:28 (Jan 2019)
15. Fellner, A., Tarrach, T., Weissenbacher, G.: Language Inclusion for Finite Prime Event Structures Artifact (Oct 2019). https://doi.org/10.5281/zenodo.3514619, https://doi.org/10.5281/zenodo.3514619
16. Friedman, E.P.: The inclusion problem for simple languages. Theoretical Computer Science **1**(4), 297–316 (1976)
17. Gaubert, S., Giua, A.: Petri net languages and infinite subsets of m. J. Comput. Syst. Sci. **59**(3), 373–391 (1999). https://doi.org/10.1006/jcss.1999.1634
18. van Glabbeek, R., Goltz, U.: Refinement of actions in causality based models. In: Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems). pp. 267–300. Springer (1989)
19. van Glabbeek, R.J., Plotkin, G.D.: Configuration structures, event structures and Petri nets. Theoretical Computer Science **410**(41), 4111–4159 (2009)
20. Godefroid, P.: Partial-order methods for the verification of concurrent systems (1996)

21. Grabowski, J.: The unsolvability of some Petri net language problems. Inf. Process. Lett. **9**(2), 60–63 (1979). https://doi.org/10.1016/0020-0190(79)90128-5
22. Hack, M.: Decidability questions for Petri Nets. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA (1976), http://hdl.handle.net/1721.1/27441
23. Hopcroft, J.E., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Pearson, 3 edn. (2013)
24. Jancar, P.: Nonprimitive recursive complexity and undecidability for Petri net equivalences. Theor. Comput. Sci. **256**(1-2), 23–30 (2001). https://doi.org/10.1016/S0304-3975(00)00100-6
25. Jones, N.D., Landweber, L.H., Lien, Y.E.: Complexity of some problems in Petri nets. Theor. Comput. Sci. **4**(3), 277–299 (1977). https://doi.org/10.1016/0304-3975(77)90014-7
26. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of computer computations, pp. 85–103. Springer (1972)
27. Ko, K.I., Lin, C.L.: On the complexity of min-max optimization problems and their approximation. In: Minimax and Applications, pp. 219–239. Springer (1995)
28. Lamport, L., et al.: Paxos made simple. ACM Sigact News **32**(4), 18–25 (2001)
29. Madhusudan, P.: Model-checking trace event structures. In: 18th IEEE Symposium on Logic in Computer Science (LICS 2003), 22-25 June 2003, Ottawa, Canada, Proceedings. pp. 371–380. IEEE Computer Society (2003). https://doi.org/10.1109/LICS.2003.1210077
30. Mayr, R., Clemente, L.: Advanced automata minimization. In: ACM SIGPLAN Notices. vol. 48, pp. 63–74. ACM (2013)
31. Mazurkiewicz, A.: Trace theory. In: Advanced course on Petri nets. pp. 278–324. Springer (1986)
32. Mazurkiewicz, A.: Introduction to trace theory. The Book of Traces pp. 3–41 (1995)
33. Meyer, A.R., Stockmeyer, L.J.: The equivalence problem for regular expressions with squaring requires exponential space. In: SWAT (FOCS). pp. 125–129 (1972)
34. Nguyen, H.T.T., Rodríguez, C., Sousa, M., Coti, C., Petrucci, L.: Quasi-optimal partial order reduction. In: Chockler, H., Weissenbacher, G. (eds.) Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II. Lecture Notes in Computer Science, vol. 10982, pp. 354–371. Springer (2018). https://doi.org/10.1007/978-3-319-96142-2_22, https://doi.org/10.1007/978-3-319-96142-2_22
35. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part i. Theoretical Computer Science **13**(1), 85–108 (1981)
36. Penczek, W.: Model-checking for a subclass of event structures. In: Brinksma, E. (ed.) Tools and Algorithms for Construction and Analysis of Systems, Third International Workshop, TACAS '97, Enschede, The Netherlands, April 2-4, 1997, Proceedings. Lecture Notes in Computer Science, vol. 1217, pp. 145–164. Springer (1997). https://doi.org/10.1007/BFb0035386
37. Peterson, J.: Petri Net Theory and the Modeling of Systems. Independently Published (2019), https://books.google.at/books?id=IthLyAEACAAJ
38. Rodríguez, C., Sousa, M., Sharma, S., Kroening, D.: Unfolding-based Partial Order Reduction. In: 26th International Conference on Concurrency Theory (CONCUR 2015). pp. 456–469 (2015)
39. Stearns, R.E., Hunt III, H.B.: On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. SIAM Journal on Computing **14**(3), 598–611 (1985)

40. Valk, R., Vidal-Naquet, G.: Petri nets and regular languages. J. Comput. Syst. Sci. **23**(3), 299–325 (1981). https://doi.org/10.1016/0022-0000(81)90067-2
41. Valmari, A.: Stubborn sets for reduced state space generation. In: International Conference on Application and Theory of Petri Nets. pp. 491–515. Springer (1989)
42. Van Glabbeek, R., Goltz, U.: Refinement of actions and equivalence notions for concurrent systems. Acta Informatica **37**(4-5), 229–327 (2001)
43. Winskel, G.: An introduction to event structures. In: Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems). pp. 364–397. Springer (1988)
44. Winskel, G.: Event structures, stable families and concurrent games (2016)
45. Winskel, G., Nielsen, M.: Handbook of logic in computer science (vol. 4). chap. Models for Concurrency, pp. 1–148. Oxford University Press, Inc., New York, NY, USA (1995), http://dl.acm.org/citation.cfm?id=218623.218630
46. Wrathall, C.: Complete sets and the polynomial-time hierarchy. Theoretical Computer Science **3**(1), 23–33 (1976)
47. Zielonka, W.: Notes on finite asynchronous automata. RAIRO-Theoretical Informatics and Applications **21**(2), 99–135 (1987)

A Lemmas and Proofs

Lemma 6. *Let C be a configuration. $|T(C)| = 1$ if and only if $|C| = 1$ or for all events $e, e' \in C$ either $(e < e')$ or $(e' < e)$.*

Proof. The claim is trivial for $|C| = 1$.

$\Rightarrow$

Let $T(C) = \{\langle e_1, \ldots, e_n\rangle\}$. Since $\langle e_2, e_1, \ldots, e_n\rangle \notin T(C)$, we have that $e_1 < e_2$. Likewise, we can show that $e_i < e_{i+1}$ for all $i \in [1, n-1]$. Therefore, $e_i < e_j$ for $i < j$. Since $C = \{e_1, \ldots, e_n\}$ have shown the claim.

$\Leftarrow$

Since $<$ is irreflexive, we immediately get that $|T(C)| > 0$. From the definition of $T(C)$ follows that $e < e'$ then e appears earlier in any trace in $T(C)$ than e'. Since we assume that every pair of events is ordered by $<$, we have that the order in any trace is fully fixed. In other words, there can only be a single trace. □

Corollary 2. *$|T(C)| > 1$ if and only if there are concurrent events $e, e' \in C$.*

Lemma 1. *Let C_1 and C_2 be maximal configuration of FLES $\mathcal{E}_1$ respectively $\mathcal{E}_2$. $C_1 \sim_N C_2$ if and only if $\mathcal{L}(C_1) \cap \mathcal{L}(C_2) \neq \emptyset$.*

Proof. $\Rightarrow$:

There is a label preserving bijection φ, such that $<_1 \cup <_2^{\varphi}$ is a partial order. Thus, via a depth first search over the events of C_1 with the order $<_1 \cup <_2^{\varphi}$, we can find a sequence of events $\langle e_1, \ldots, e_n\rangle$, such that $e_j(<_1 \cup <_2^{\varphi})e_i$ implies $j < i$. It follows that $e_j <_1 e_i$ implies $j < i$, i.e. $\langle e_1, \ldots, e_n\rangle \in T(C_1)$. Likewise it is the case that $e_j(<_2^{\varphi})e_i$ implies $j < i$ and by the definition of $<^{\varphi}$, we get $\varphi(e_j) <_2 \varphi(e_i)$, i.e. $\varphi(\langle e_1, \ldots, e_n\rangle) \in T(C_2)$. Since φ is label-preserving, $h(\langle e_1, \ldots, e_n\rangle) = h(\varphi(\langle e_1, \ldots, e_n\rangle)) =: w$, which implies $w \in \mathcal{L}(C_1)$ and $w \in \mathcal{L}(C_2)$.

$\Leftarrow$:

Let $t_1 = \langle e_1, \ldots, e_n \rangle$ and $t_2 = \langle e'_1, \ldots, e'_n \rangle$ be traces of C_1 and C_2 respectively, such that $h(t_1) = h(t_2) \in \mathcal{L}(C_1) \cap \mathcal{L}(C_2)$. We show that the mapping $\varphi : e_i \mapsto e'_i$ is a necessary embedding. Clearly φ is bijective and label-preserving. We need to show $\forall e \in C_1 : \neg e(<_1 \cup <_2^{\varphi})e$. Assume the contrary, i.e. $\exists e \in C_1 : e(<_1 \cup <_2^{\varphi})e$. In order to close the cycle and since $<_1$ and $<_2$ are order relations, there must exist an event $e' \in C_1$, such that $e' <_1 e$ and $e <_2^{\varphi} e'$, i.e. $\varphi(e) <_2 \varphi(e')$. This implies, that for all $t \in T(C_1)$ we have that e' appears earlier than e and all traces $t \in T(C_2)$ are such that $\varphi(e)$ appears earlier than $\varphi(e')$. This is a contradiction to $t_1 \in T(C_1), t_2 \in T(C_2)$ and $t_2 = \varphi(t_1)$. □

Lemma 7. *An event e is* maximal *in C, if there is no event $e' \in C$, such that $e < e'$. Let C_1 and C_2 be configurations such that $C_1 \sqsubset_S^{\varphi} C_2$. If e is maximal in C_1 then $\varphi(e)$ is maximal in C_2.*

Proof. Assume there exists e', such that $\varphi(e) <_2 \varphi(e')$. Since φ is a sufficient embedding, we have that $e <_1 e'$, which is a contradiction to e being maximal in C_1. Therefore, no such e' exists. Since φ is bijective, $\varphi(e)$ is maximal in C_2. □

Lemma 8. *Let e be maximal in configuration C, then $C \setminus \{e\}$ is a configuration and for every $\langle e_1, \ldots, e_n \rangle \in T(C \setminus \{e\})$ it is the case that $\langle e_1, \ldots, e_n, e \rangle \in T(C)$.*

Proof. $C \setminus \{e\}$ is conflict free, because C is conflict free. $C \setminus \{e\}$ is left closed, because e is maximal, therefore there is no event e' such that $e < e'$. Note that traces in $T(.)$ contain all events of the configuration. Therefore, from $\langle e_1, \ldots, e_n \rangle \in T(C \setminus \{e\})$ follows $\{e_1, \ldots, e_n, e\} = C$. Furthermore, for every $e' \in C$, such that $e' < e$, we have that $e' \in \{e_1, \ldots, e_n\}$, which implies $\langle e_1, \ldots, e_n, e \rangle \in T(C)$. □

Lemma 9. *Let C be a configuration. $\langle e_1, \ldots, e_n \rangle \in T(C)$ implies that e_n is maximal in C.*

Lemma 2. *Let C_1 and C_2 be maximal configurations of FLES $\mathcal{E}_1$ and $\mathcal{E}_2$ respectively. If $C_1 \sqsubset_S C_2$ then $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$.*

Proof. We show the claim by induction on $|C_1| = |C_2| = n$.

In the base case we have $C_1 = \{\bot\}$ and $C_2 = \{\bot\}$. For these configurations the claim is trivially fulfilled.

Let the induction hypothesis be that the claim holds for all configurations of size n.

Assume that C_1 and C_2 are configurations of size $n+1$, such that $C_1 \sqsubset_S^{\varphi} C_2$. We show $T(C_1) \subseteq \varphi(T(C_2))$. Since φ is bijective and label-preserving, $T(C_1) \subseteq \varphi(T(C_2))$ is equivalent to $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$.

Let $t = \langle e_1, \ldots, e_{n+1} \rangle \in T(C_1)$. We need to show that $\varphi(t) \in T(C_2)$.

Since e_{n+1} is the last event of a trace and we consider only traces that include all events of the configuration, clearly e_{n+1} is maximal in C_1. According to Lemmas 7 and 8, we have that $C_1 \setminus \{e_{n+1}\}$ and $C_2 \setminus \{\varphi(e_{n+1})\}$ are

configurations of size n. Furthermore, φ restricted to $C_1 \setminus \{e_{n+1}\}$ is a witness for $C_1 \setminus \{e_{n+1}\} \sqsubset_S C_2 \setminus \{\varphi(e_{n+1})\}$. From the induction hypothesis we get $\varphi(\langle e_1, \ldots, e_n \rangle) \in T(C_2 \setminus \{\varphi(e_{n+1})\})$. Since $\varphi(e_{n+1})$ is maximal in C_2, from Lemma 8 follows $\varphi(\langle e_1, \ldots, e_n, e_{n+1} \rangle) \in T(C_2)$. □

Lemma 10. *Let C_1 and C_2 be maximal configuration of FLES $\mathcal{E}_1$ and $\mathcal{E}_2$ and for every $e_1 \in C_1$ and $e_2 \in C_2$ it is the case that $|\{e \in C_1 \mid h(e_1) = h(e)\}| = 1$ and $|\{e \in C_2 \mid h(e_2) = h(e)\}| = 1$*
If $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$ then $C_1 \sqsubset_S C_2$.

Proof. We show the claim by induction on $|C_1| = |C_2| = n$.

In the base case we have $C_1 = \{\bot\}$ and $C_2 = \{\bot\}$. For these configurations the claim is trivially fulfilled.

Let the induction hypothesis be that the claim holds for all configurations of size n.

Let e_{n+1} be a maximal event of C_1. Let $M_{e_{n+1}} := \{e \in C_2 \mid e \text{ is maximal and } h(e_{n+1}) = h(e)\}$. We start by showing $\exists e \in M_{e_{n+1}}$, such that $\mathcal{L}(C_1 \setminus \{e_{n+1}\}) \subseteq \mathcal{L}(C_2 \setminus \{e\})$.

Assume the contrary. That is, for every maximal event $e \in M_{e_{n+1}}$ it is the case that $\mathcal{L}(C_1 \setminus \{e_{n+1}\}) \nsubseteq \mathcal{L}(C_2 \setminus \{e\})$. Then $\mathcal{L}(C_1 \setminus \{e_{n+1}\}) \nsubseteq \bigcup_{e \in M_{e_{n+1}}} \mathcal{L}(C_2 \setminus \{e\})$. That is, there is a word $w \in \mathcal{L}(C_1 \setminus \{e_{n+1}\})$ such that $w \notin \bigcup_{e \in M_{e_{n+1}}} \mathcal{L}(C_2 \setminus \{e\})$. Since e_{n+1} is maximal $w \circ h(e_{n+1}) \in \mathcal{L}(C_1)$ ($\circ$ denotes concatenation). Let us define language $\mathcal{L}^\circ := \bigcup_{e \in M_{e_{n+1}}} \mathcal{L}(C_2 \setminus \{e\}) \circ h(e_{n+1})$ ($\circ$ is applied element-wise). Clearly, $w \circ h(e_{n+1}) \notin \mathcal{L}^\circ$.

We claim that no word in $\mathcal{L}(C_2) \setminus \mathcal{L}^\circ$ ends with $h(e_{n+1})$. Assume there is such a word w'. This word corresponds to a trace ending with some event e' with label $h(e_{n+1})$. Since e' is the last event in a trace, and we only consider maximal configurations, it needs to be maximal. Therefore, $e' \in M_{e_{n+1}}$, which is a contradiction to $w' \in \mathcal{L}(C_2) \setminus \mathcal{L}^\circ$. Since $w \circ h(e_{n+1})$ ends with $h(e_{n+1})$, we have $w \circ h(e_{n+1}) \notin \mathcal{L}(C_2) \setminus \mathcal{L}^\circ$.

However, clearly $\mathcal{L}(C_2) = \mathcal{L}^\circ \cup (\mathcal{L}(C_2) \setminus \mathcal{L}^\circ)$. In summary, we found $w \circ h(e_{n+1}) \in \mathcal{L}(C_1)$ such that $w \circ h(e_{n+1}) \notin \mathcal{L}(C_2)$, which is a contradiction to $\mathcal{L}(C_1) \subseteq \mathcal{L}(C_2)$.

Therefore, we can apply the induction hypothesis to $C_1 \setminus \{e_{n+1}\}$ and $C_2 \setminus \{e\}$ for the existing $e \in M_{e_{n+1}}$. That is, $C_1 \setminus \{e_{n+1}\} \sqsubset_S^\varphi C_2 \setminus \{e\}$. We extend φ to C_1 by setting $\varphi(e_{n+1}) := e$. Clearly, φ is bijective and label-preserving. Furthermore, causality preservation for events other than e_{n+1} carries over from $C_1 \setminus \{e_{n+1}\}$ to C_1. What remains to show is that for all $e \in C$ with $\varphi(e) < \varphi(e_{n+1})$ we have $e < e_{n+1}$. Since e_{n+1} is maximal, clearly it is not the case that $e_{n+1} < e$. Assume that e and e_{n+1} are concurrent. Then there is a word, in which $h(e_{n+1})$ appears earlier than $h(e)$. Since events are uniquely labeled and $\varphi(e) < \varphi(e_{n+1})$, this word is not in $\mathcal{L}(\mathcal{E}_2)$, which is a contradiction to $\mathcal{L}(\mathcal{E}_1) \subseteq \mathcal{L}(\mathcal{E}_2)$. Therefore, $e < e_{n+1}$ and φ is a sufficient embedding. □

Lemma 11. *Let C be a configuration and $e_1, e_2 \in C$ be concurrent events, then $T(C) = T(\mathcal{E}^C_{e_1<e_2}) \cup T(\mathcal{E}^C_{e_2<e_1})$ and $T(\mathcal{E}^C_{e_1<e_2}) \cap T(\mathcal{E}^C_{e_2<e_1}) = \varnothing$.*

Proof. From $t \in T(C_{e_1<e_2})$ directly follows $t \in T(C)$, since there is simply one less constraint to fulfill. Thus, we have $T(C_{e_1<e_2}) \subseteq T(C)$. Symmetrically we can show $T(C_{e_2<e_1}) \subseteq T(C)$, thus showing $T(C_{e_1<e_2}) \cup T(C_{e_2<e_1}) \subseteq T(C)$.

Let $t = \langle e'_1, \ldots, e'_n \rangle \in T(C)$. There are $i \neq j$, such that $e'_i = e_1$ and $e'_j = e_2$. In the case $i < j$, t fulfills all constraints to be a trace of $C_{e_1<e_2}$, thus $t \in T(C_{e_1<e_2})$. Likewise $j < i$ implies $t \in T(C_{e_2<e_1})$. In any case $t \in T(C_{e_1<e_2}) \cup T(C_{e_2<e_1})$, showing $T(C) \subseteq T(C_{e_1<e_2}) \cup T(C_{e_2<e_1})$.

Further, the $T(\mathcal{E}^C_{e_1<e_2})$ and $T(\mathcal{E}^C_{e_2<e_1})$ are disjoint because no trace can fulfill both constraints $e_1 < e_2$ and $e_2 < e_1$. □

Lemma 3. *Let C be a configuration and $e_1, e_2 \in C$ be concurrent events, then $\mathcal{L}(C) = \mathcal{L}(C_{e_1<e_2}) \cup \mathcal{L}(C_{e_2<e_1})$. If h is injective (labels are unique), then $\mathcal{L}(C_{e_1<e_2}) \cap \mathcal{L}(C_{e_2<e_1}) = \varnothing$.*

Proof. $\mathcal{L}(C) = \mathcal{L}(C_{e_1<e_2}) \cup \mathcal{L}(C_{e_2<e_1})$ is a direct consequence of Lemma 11 and the definition of the language of prime event structures. If h is injective, i.e. labels are unique, then each word in $\mathcal{L}(C_{e_1<e_2})$ contains the sub sequence $\ldots h(e_1) \ldots h(e_2) \ldots$ whereas each word in $\mathcal{L}(C_{e_2<e_1})$ contains the different sub sequence $\ldots h(e_2) \ldots h(e_1), \ldots$, which shows that no word can be part of both languages. □

Lemma 4. *Let C_1, C_2 be maximal configuration of FLES $\mathcal{E}_1$ respectively $\mathcal{E}_2$. Furthermore, let $C_1 \sim^{\varphi}_N C_2$ and $C_1 \not\sim^{\varphi}_S C_2$. Then there are concurrent events $e, e' \in C_1$, such that $\varphi(e) <_2 \varphi(e')$.*

Proof. Since φ is a label-preserving bijection, from $C_1 \not\sim^{\varphi}_S C_2$ follows that there are events $e, e' \in C$, such that $\varphi(e) <_2 \varphi(e')$ and $e \not<_1 e'$. Towards contradiction assume $e' <_1 e$. Then we have $e'(<_1 \cup <^{\varphi}_2)e'$, which is a contradiction to φ being a necessary embedding. Therefore, e, e' are concurrent and a witness to the claim. □

Lemma 5. *Let $\mathcal{E}$ be a labeled, finite prime event structure, then $\mathcal{L}(\mathcal{E}) = \mathcal{L}(\mathcal{A}^{\mathcal{E}})$.*

Proof. Given a trace $\vec{e} = \langle e_1, \ldots, e_n \rangle$ we define $C^{\vec{e}}_l := \cup^l_{i=1}\{e_i\}$.

Let $w \in \mathcal{L}(\mathcal{A}^{\mathcal{E}})$, then by the definition of $\mathcal{A}^{\mathcal{E}}$, there is a trace $\vec{e} = \langle e_1, \ldots, e_n \rangle$ such that $w = \langle h(e_1), \ldots, h(e_n) \rangle$, such that for every $l \leq n : C^{\vec{e}}_l$ is a configuration and $C^{\vec{e}}_n$ is a maximal configuration. Since all $C^{\vec{e}}_l$ are configurations, we have $\langle e_1, \ldots, e_n \rangle \in T(C^{\vec{e}}_n)$. Since $C^{\vec{e}}_n$ is maximal, we have $w \in \mathcal{L}(\mathcal{E})$.

Conversely, for every $w \in \mathcal{L}(\mathcal{E})$, there is a trace $\vec{e} = \langle e_1, \ldots, e_n \rangle$ such that $w = \langle h(e_1), \ldots, h(e_n) \rangle$ and $\langle e_1, \ldots, e_n \rangle \in T(\mathcal{E})$. From $\vec{e} \in T(\mathcal{E})$ follows that for every $l \leq n : C^{\vec{e}}_l$ is a configuration and that $C^{\vec{e}}_n$ is maximal. By the definition of $\delta^{\mathcal{E}}$, we have for every $i = 1, \ldots, n-1 : (q^{\mathcal{E}}_{C^{\vec{e}}_i}, h(e_i), q^{\mathcal{E}}_{C^{\vec{e}}_{i+1}}) \in \delta^{\mathcal{E}}$. Since $C^{\vec{e}}_n$ is maximal, $q^{\mathcal{E}}_{C^{\vec{e}}_n}$ is accepting. Therefore, $w \in \mathcal{L}(Q^{\mathcal{E}})$. □

Theorem 5. *There is a family of event structures $\mathcal{E}_n$ with events E_n, such that $|E_n| = n + 1$, $|\mathcal{L}(\mathcal{E}_n)| = n!$, and $|Q^{\mathcal{E}}| = 2^n + 1$. Every NFA $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{E}_n)$ has at least 2^n states.*

Proof. The family is given by the $\mathcal{X}$-labeled prime event structures $\mathcal{E}_n := \langle E_n, <_n, \varnothing, h_n \rangle$, where $\mathcal{X} = \{\varepsilon\} \cup \{1, \ldots, n\}$, $E_n = \{\bot\} \cup \{e_1, \ldots, e_n\}$, $<_n := \{(\bot, e_i) \mid i = 1, \ldots, n\}$, and $h_n(e_i) := i$. $\mathcal{E}_n$ encodes exactly all permutations of $\{1, \ldots, n\}$. As shown in [10], no NFA with less than 2^n states can accept the language of permutations of n symbols, showing that every $\mathcal{A}$ with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{E}_n)$ has at least 2^n states.
Furthermore, the number of all permutations of n symbols is $n!$, showing that $|\mathcal{L}(\mathcal{E}_n)| = n!$.
Finally, every non-empty subset of E_n is a configuration. There are $2^n + 1$ such sets, which are all subsets of $\{e_1, \ldots, e_n\}$ together with $\bot$ plus the set $\{\bot\}$. Since states of our encoding correspond one-to-one with configurations, we have $|Q^{\mathcal{E}}| = 2^n + 1$. □

B Detailed Proof of the Language Inclusion Reduction

Theorem 2. *Finite prime event structure language membership is NP-hard.*

Proof. We proof by reduction of HC to FLES language membership.

We use $s(f)$ and $t(f)$ to denote the respectively source and target vertices of directed edge f.

Let $G = (V, F)$ be a directed graph and define $n := |V|$. We assume that G does not contain any self loops, i.e. edges f such that $s(f) = t(f)$, and that $n > 1$. Apart from a graph with only one vertex, a graph contains a Hamiltonian cycle if and only if the same graph without self-loops contains a Hamiltonian cycle. The case $n = 1$ can be trivially decided and is therefore not considered in our reduction. A graph with self-loops can be converted into one without self-loops in linear time by removing self-loops from F.

Given an integer $j \in [1, n]$, we abbreviate $(j \bmod n) + 1$ by $\mathrm{su}(j)$. We say that f is connected to f' if $t(f) = s(f')$. We say that a sequence of edges $\langle f_1, \ldots, f_n \rangle$ is a cycle if for every $j \in [1, n] : f_j$ is connected to $f_{\mathrm{su}(j)}$. A Hamiltonian cycle of G is a cycle $\langle f_1, \ldots, f_n \rangle$ of F, such that $\{t(f_j) \mid j \in [1, n]\} = V$. The decision problem HC: "Does there exist a Hamiltonian cycle of G" is NP-hard [26].

We provide a polynomially sized (in $|V| + |F|$) $\{\varepsilon, x\}$-labeled event structure $\mathcal{E}^G := \langle E, <, \#, h \rangle$, such that $x^n \in \mathcal{L}(\mathcal{E}^G)$ (n times the letter x concatenated) if and only if there exists a Hamiltonian cycle of G. The main idea is that configurations of the event structure translate to connected sequences of edges via events that each represent the assignment of an edge to a position in the sequence and events that express connectedness of two successive edges in the sequence.
The set of events E contains exactly the following **events**:

- The initial event $\bot$
- For every edge $f \in F$ and $j \in [1, n]$, there is an event $e_{f,j} \in E$ representing that f is the j'th element of a sequence of edges.

– For every pair of edges $f, f' \in F$ with $t(f) = s(f')$ and $j \in [1, n]$ there is an event $e_{f,f',j} \in E$ representing that f and f' are successive elements of a sequence of edges.

Formally, we define E as follows:

$$E := \{\bot\} \cup \bigcup_{j=1}^{n} \left(\{e_{f,j} \mid f \in F\} \cup \{e_{f,f',j} \mid f, f' \in F \wedge t(f) = s(f')\}\right)$$

Clearly, $|\mathcal{E}^G| = |E|$ is polynomial in $|F| \leq |G|$. Note that causality, conflict relation, and labeling are always of at most quadratic size in $|E|$.
The causality relation $<$ is the smallest partial order relation containing the following **causalities**:

– For every edge $f \in F$ and $j \in [1, n]$: $\bot < e_{f,j}$ representing that every assignment of a single edge to a position in a sequence of edges is allowed.
– For every pair of edges $f, f' \in F$ with $t(f) = s(f')$, and every $j \in [1, n]$: $e_{f,j} < e_{f,f',j}$ and $e_{f',\mathrm{su}(j)} < e_{f,f',j}$ representing that successive edges assigned in the represented sequence are connected.

Formally, we define $<$ as follows, where X^+ denotes the transitive closure of relation X:

$$<:= \left(\bigcup_{j=1}^{n} \left(\{(\bot, e_{f,j}) \mid f \in F\} \cup \{(e_{f,j}, e_{f,f',j}), (e_{f',\mathrm{su}(j)}, e_{f,f',j}) \mid e_{f,f',j} \in E\}\right)\right)^+$$

The conflict relation $\#$ is the smallest conflict relation closed under $<$ (i.e. $e \# e' \wedge e' < e'' \Rightarrow e \# e''$) containing the following **immediate conflicts**:

C.1 For every edge $f \in F$ and $j, k \in [1, n], j \neq k$: $e_{f,j} \# e_{f,k}$ representing that every edge can only be assigned to one position in a sequence.
C.2 For every pair of edges $f, f' \in F, f \neq f'$ and every $j \in [1, n]$: $e_{f,j} \# e_{f',j}$ representing that every position in the sequence can only be assigned once.
C.3 For every pair of edges $f, f' \in F, f \neq f'$, such that $t(f) = t(f')$ and every $j, k \in [1, n]$: $e_{f,j} \# e_{f,k}$ representing that edges assigned to a sequence must have non-overlapping target vertices.

Formally, $\#$ has the following set of immediate conflicts:

$$\#^i := \bigcup_{j=1}^{n} \left(\bigcup_{i=1, i \neq j}^{n} \left(\{(e_{f,j}, e_{f,i}) \mid f \in F\} \cup \{(e_{f,j}, e_{f',i}) \mid f, f' \in F \wedge f \neq f' \wedge t(f) = t(f')\} \right) \right.$$
$$\left. \cup \{(e_{f,j}, e_{f',j}) \mid f, f' \in F \wedge f \neq f'\} \right)$$

The **labeling function** h is given as follows:

- Every event of the form $e_{f,f',j}$ has label $h(e_{f,f',j}) := x$.
- Every other event in E has label ε.

Foramlly, h is defined as follows:

$$h(e) := \begin{cases} x & \text{for } e = e_{f,f',j} \in E \\ \varepsilon & \text{otherwise} \end{cases}$$

We say that a configuration of $\mathcal{E}^G$ represents a sequence of edge $\langle f_1, \ldots, f_m \rangle$ if it includes events $e_{f_1,i_1}, \ldots, e_{f_m,i_m}$, where $\{i_1, \ldots, i_m\} \subseteq [1,n]$, for every $j \in [1, m-1] : i_j < i_{j+1}$.

Configurations represent sequences of edges: We claim that every configuration (besides $\{\perp\}$) of $\mathcal{E}^G$ represents a sequence of $m \leq n$ edges with pairwise different targets.

The claim follows from the structure of the immediate conflicts: Due to conflicts **C.1**, a configuration cannot contain events $e_{f,j}$ and $e_{f,k}$ for $j \neq k$. Therefore, for every edge, a configuration includes one such event or none. Furthermore, the indices j of events $e_{f,j}$ give rise to a sequence of represented edges. Due to conflicts **C.2**, every index can only be assigned to one position in the sequence. There are at most n possible positions. Therefore, every configuration (besides $\{\perp\}$) represents a sequence of $m \leq n$ edges. Furthermore, the edges must pairwise different targets due to conflicts **C.3**.

Configurations with events $e_{f,f',j}$ represent sequences of (partially) connected edges:

Due to the causes of events $e_{f,f',j}$, we have that a configuration contains $e_{f,f',j}$ if and only if it represents a sequence of edges $f_1, \ldots, f_j = f, f_{\mathrm{su}(j)} = f', \ldots, f_m$.

Hamiltonian cycle $\Rightarrow x^n \in \mathcal{L}(\mathcal{E}^G)$:

Assume G has a Hamiltonian cycle $\langle f_1, \ldots, f_n \rangle$. We claim that the set of events $\perp \cup \{e_{f_j,j}, e_{f_j,f_{\mathrm{su}(j)},j} \mid j \in [1,n]\}$ is a maximal configuration. The set is causally closed, since the edges are connected. The set is conflict free, because every position is assigned exactly once (no conflicts among **C.1** and **C.2**) and since the sequence is a Hamiltonian cycle, the targets are not overlapping (no conflict among **C.3**). The configuration is maximal, since clearly no further event of the form $e_{f,i}$ can be added, since the assignment of edges to positions is fixed, and no event of the form $e_{f,f',i}$ can be added, since these events are directly induced by the assignment of edges to positions in the sequence. Furthermore, this maximal configuration contains exactly n events labeled by x and all other events are labeled by ε, showing $x^n \in \mathcal{L}(\mathcal{E}^G)$.

$x^n \in \mathcal{L}(\mathcal{E}^G) \Rightarrow$ Hamiltonian cycle:

If $x^n \in \mathcal{L}(\mathcal{E}^G)$, then $\mathcal{E}^G$ has a maximal configuration C that includes n events of the form $e_{f,f',j}$.

Assume that C does not represent a Hamiltonian cycle. That is, two successive edges in the represented sequence of edges that are not connected, or not all vertices are visited by the sequence.

The former case cannot be true, due to presence of events $e_{f,f',j}$ in C, showing that f is connected to f' and the causalities of such events ($e_{f,j}$ and $e_{f',\mathrm{su}(j)}$),

implying that f and f' are successive edges in the sequence of edges represented by C.

To see why the latter case cannot be true, consider that in order for a connected sequence of n edges not to visit one of the n vertices of the graph, it needs to visit some vertex twice. That is, it needs to include edges that have the same target vertex. C can not represent two different edges with the same target due to conflicts **C.3**. Furthermore, C can not represent the same edge twice, due to conflicts **C.1**. Therefore, the sequence of edges represented by C can not contain two edges with the same target.

Therefore, the maximal configuration C represents a Hamiltonian cycle in G. □

Theorem 4. *Finite prime event structure language inclusion is Π_2^p-hard.*

Proof. We prove the claim by reduction of DHC to FLES language inclusion. Let $G = (V, F)$ be a finite, undirected graph and let $B \subseteq F$. Let $V = \{v_1, \ldots, v_n\}$, $F = \{f_1, \ldots, f_k\}$, and $B = \{b_1, \ldots, b_m\}$. Let $bh := \mathsf{floor}(\frac{|B|}{2})$. Using the same arguments of generality as in the proof of Theorem 2, we assume that G does not contain any self loops, i.e. edges f such that $s(f) = t(f)$, and that $n > 1$.

For this proof, we will use the same method to encode Hamiltonian cycles of directed graphs into event structures as in the proof of Theorem 2. Therefore, the first step of our reduction is to encode G as a directed graph $\vec{G} = (V, \vec{F})$ with $\vec{F} := \{\vec{f}, \overleftarrow{f} \mid f = (u, v) \in F \wedge \vec{f} = \langle u, v\rangle \wedge \overleftarrow{f} = \langle v, u\rangle\}$, where (u, v) and $\langle u, v\rangle$ denote undirected, respectively directed, edges between vertices u and v. A self loop-free graph G has an undirected Hamiltonian cycle if and only if $\vec{G}$ has a directed Hamiltonian cycle. Intuitively this is true, because we introduce for each undirected edge an edge in each direction that connect the vertices in either direction, just as the undirected edge does.

We provide $\{\varepsilon, x, y, lb_1, \ldots, lb_m\}$-labeled event structures $\mathcal{E}_1^{G,B} = \langle E_1, <_1, \#_1, h_1\rangle$ and $\mathcal{E}_2^{G,B} := \langle E_2, <_2, \#_2, h_2\rangle$ such that $|\mathcal{X}|$, $|\mathcal{E}_1^{G,B}| := |E_1|$, and $|\mathcal{E}_2^{G,B}| := |E_2|$ are polynomial in $|G| := |V| + |F|$ and G, B has a DHC if and only if $\mathcal{L}(\mathcal{E}_1^{G,B}) \subseteq \mathcal{L}(\mathcal{E}_2^{G,B})$. X^+ denotes the transitive closure of relation X.

We define the components of $\mathcal{E}_1^{G,B}$ as follows, where $\#_1^i$ denotes immediate conflicts:

$$E_1 := \{\bot\} \cup \bigcup_{i=1}^{n} \{ev_i\} \cup \bigcup_{i=1}^{m} \{ein_i, eD_i, eout_i\}$$

$$<_1 := \left(\{(\bot, ev_1)\} \cup \bigcup_{i=2}^{n} \{(ev_{i-1}, ev_i)\} \cup \bigcup_{i=1}^{m} \{(\bot, ein_i), (\bot, eout_i), (ein_i, eD_i)\}\right)^+$$

$$\#_1^i := \bigcup_{i=1}^{m} \{(ein_i, eout_i)\}$$

$$h_1(e) := \begin{cases} x & \text{for } e \in \{ev_1, \ldots, ev_n\} \\ lb_i & \text{for } e = ein_i \\ y & \text{for } e \in \{eD_1, \ldots, eD_m\} \\ \varepsilon & \text{otherwise} \end{cases}$$

For the definition of $\mathcal{E}_2^{G,B}$, we make use of the event structure $\mathcal{E}^G$ encoding all Hamiltonian cycles in $\vec{G}$ defined in the proof of Theorem 2 and embed it into $\mathcal{E}_2^{G,B}$ using a new root event $es \in E_2$:

$$E^{HC} := \bigcup_{j=1}^{n} \left(\{e_{f,j} \mid f \in \vec{F}\} \cup \{e_{f,f',j} \mid f, f' \in \vec{F} \wedge t(f) = s(f')\}\right)$$

$$<^{HC} := \bigcup_{j=1}^{n} \left(\{(es, e_{f,j}) \mid f \in \vec{F}\} \cup \{(e_{f,j}, e_{f,f',j}), (e_{f',\mathrm{su}(j)}, e_{f,f',j}) \mid e_{f,f',j} \in E^{HC}\}\right)$$

$$\#^{HC} := \bigcup_{j=1}^{n} \left(\bigcup_{i=1, i\neq j}^{n} \left(\{(e_{f,j}, e_{f,i}) \mid f \in \vec{F}\} \cup \{(e_{f,j}, e_{f',i}) \mid f, f' \in \vec{F} \wedge f \neq f' \wedge t(f) = t(f')\}\right) \right.$$
$$\left. \cup \{(e_{f,j}, e_{f',j}) \mid f, f' \in \vec{F} \wedge f \neq f'\} \right)$$

Finally, we define the components of $\mathcal{E}_2^{G,B}$ as follows, where $\#_2^i$ denotes immediate conflicts:

$$E_2 := \{\bot, es\} \cup \bigcup_{i=1}^{n} \{ev_i\} \cup \bigcup_{i=1}^{m} \{ein_i, eout_i, eD_i, efix_{i-1}\} \cup E^{HC}$$

$$<_2 := \Big(\{(\bot, es), (\bot, ev_1), (\bot, efix_0), (\bot, eD_1)\} \cup \bigcup_{i=2}^{n} \{(ev_{i-1}, ev_i)\} \cup \bigcup_{i=1}^{m} \{(\bot, ein_i), (\bot, eout_i)\} \cup \bigcup_{i=1}^{m-1} \{(eD_i, efix_i), (eD_i, eD_{i+1})\} \cup <^{HC} \Big)^+$$

$$\#_2^i := \{(es, ev_1)\} \cup \bigcup_{i=1}^{m} \{(ein_i, eout_i), (eD_i, efix_{i-1})\} \cup \bigcup_{i=1}^{bh} \{(efix_i, ev_1) \cup \#^{HC}\} \cup \bigcup_{i=1}^{m} \bigcup_{j=1}^{n} \{(ein_i, e_{\vec{f}_i,j}), (ein_i, e_{\bar{f}_i,j})\}$$

$$h_2(e) := \begin{cases} x & \text{for } e = e_{f,f',j} \in E_2 \text{ for some } f, f' \in F \text{ and } j \in [1, n] \\ x & \text{for } e \in \{ev_1, \ldots, ev_n\} \\ y & \text{for } e \in \{eD_1, \ldots, eD_m\} \\ lb_i & \text{for } e = ein_i \\ \varepsilon & \text{otherwise} \end{cases}$$

From the definition of $\mathcal{E}_1^{G,B}$ and $\mathcal{E}_2^{G,B}$, we can immediately see that the reduction is polynomial. Notice that the causality and the conflict relation, as well as the number of edges of the graph is always at most quadratic in the number of events and vertices in G.

Both event structures $\mathcal{E}_1^{G,B}$ and $\mathcal{E}_2^{G,B}$ encode subsets D of B. In particular, every word $w \in \mathcal{L}(\mathcal{E}_1^{G,B}) \cup \mathcal{L}(\mathcal{E}_2^{G,B})$ encodes a subset $D(w) := \{b_i \mid lb_i \in w\}$ of B.

Consider an arbitrary word $w \in \mathcal{L}(\mathcal{E}_1^{G,B})$. We first gather some properties of its symbols and then proceed to show under which condition $w \in \mathcal{L}(\mathcal{E}_2^{G,B})$ holds. Firstly, w must contain the symbol x exactly $|V|$ times due to events ev_i. Secondly, every event ein_i of $\mathcal{E}_1$ causes an event eD_i with label y. Therefore, $w \in \mathcal{L}(\mathcal{E}_1^{G,B})$ encodes the cardinality of $D(w)$ by the number of y symbols in w. Finally, since all events ev_i are concurrent to all events ein_j and all events ein_j are pairwise concurrent, symbols lb_i and x can appear in any permutation.

To show under which conditions $w \in \mathcal{L}(\mathcal{E}_2^{G,B})$ holds, we distinguish whether y occurs in w more often or less or equal than bh times.

Firstly, consider the case that w contains the symbol y more often than bh. That is, w encodes $D(w)$ with $|D(w)| > bh$. Since DHC does not require the existence of a Hamiltonian cycle in $G_{D(w)}$, the word should be contained in $\mathcal{L}(\mathcal{E}_2^{G,B})$ for any graph G. Consider the set of events $C_{D(w)} := \{\bot\} \cup \{ein_j \mid b_j \in D(w)\} \cup \{eout_j \mid b_j \notin D(w)\} \cup \lceil efix_{|D(w)|} \rceil \cup \{ev_1, \ldots, ev_n\}$. The set is a maximal configuration, since $efix_{|D(w)|}$ is not in conflict with ev_1, due to $|D(w)| > bh$. Furthermore, $w \in \mathcal{L}(C_{D(w)})$ because the y-labeled events eD_i are pairwise concurrent with the x-labeled events $\{ev_1, \ldots, ev_n\}$ which are in turn pairwise concurrent with the lb_i-labeled $\{ein_j \mid b_j \in D(w)\}$ events. Furthermore, the lb_i-labeled events $\{ein_j \mid b_j \in D(w)\}$ are pairwise concurrent to each other. There-

fore, $\mathcal{L}(C_{D(w)})$ includes exactly all permutations of these symbols, in particular $w \in \mathcal{L}(C_{D(w)})$. In summary, for words w which encode $D(w)$, such that $|D(w)| > bh$, we have $w \in \mathcal{L}(\mathcal{E}_2^{G,B})$.

Secondly, consider the converse case that w encodes a set $D(w)$ with $|D(w)| \leq bh$. Any maximal configuration $C_{D(w)}$ such that $w \in \mathcal{L}(C_{D(w)})$ must include events $\{\bot\} \cup \{ein_j \mid b_j \in D(w)\} \cup \{eout_j \mid b_j \notin D(w)\} \cup \lceil efix_{|D(w)|} \rceil$.

In contrast to the first case, the event ev_1 is in conflict with $efix_{|D(w)|}$, because $|D(w)| \leq bh$. However, the event es is not in conflict with $efix_{|D(w)|}$. Hamiltonian cycles are encoded in the sub-event structure following es, which can be seen by the arguments presented in the proof of Theorem 2. However, due to conflicts of events ein_i with $e_{\vec{f}_i,j}$ and $e_{\overleftarrow{f}_i,j}$, only events $e_{\vec{f},j}$ and $e_{\overleftarrow{f},j}$ can be included in $C_{D(w)}$ with $f \notin D(w)$, i.e. edges of the graph $G_{D(w)}$. Thus, analogous to the proof of Theorem 2, $C_{D(w)}$ include $|V|$ x-labeled events if and only if $G_{D(w)}$ has a Hamiltonian cycle.

In summary, for a word $w \in \mathcal{L}(\mathcal{E}_1)$, we have $w \in \mathcal{L}(\mathcal{E}_2)$ if and only if $|D(w)| > bh$ or $|D(w)| \leq bh$ and $G_{D(w)}$ contains a Hamiltonian cycle. Furthermore, for every $D \subseteq B$, there is a word $w \in \mathcal{L}(\mathcal{E}_1)$, such that $D = D(w)$.

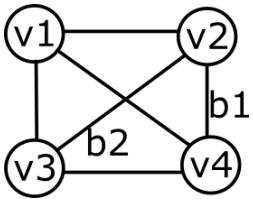
Therefore, we get $\mathcal{L}(\mathcal{E}_1^{G,B}) \subseteq \mathcal{L}(\mathcal{E}_2^{G,B})$ if and only if G and B satisfy DHC. □

Example 4. Consider the graph shown in Figure 5a. The graph and the set of edges $\{b_1, b_2\}$ form a dynamic Hamiltonian cycle.

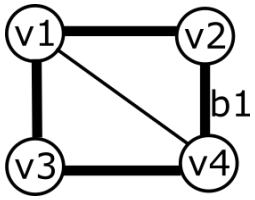
Figures 6 and 7 show the event structures $\mathcal{E}_1$ and $\mathcal{E}_2$ constructed according to the reduction in the proof of Theorem 4 on this example graph.

The bold events in Figures 6 and 7 show the configurations that correspond to the Hamiltonian cycle that can be obtained when b_2 is removed. The configuration where b_1 is removed is similar.

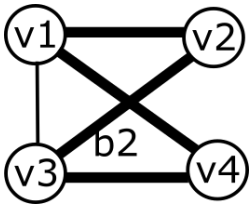
The remaining cases are removing none or both of b_1 and b_2. In the case that both are removed, the number of y labels is 2 and therefore eD_2 has to be part of the configuration, enabling $ev_1, \ldots, ev_4$ to generate 4 times x. In case no b_i is removed, the same configuration as in Figure 7 can be used to show existence of a Hamiltonian cycle with small changes to accommodate for the different number of y and the non-existence of lb_2.



(a) Graph with dynamic Hamiltonian cycle.



(b) Cycle without in_2.



(c) Cycle without in_1.

Fig. 5: Graph

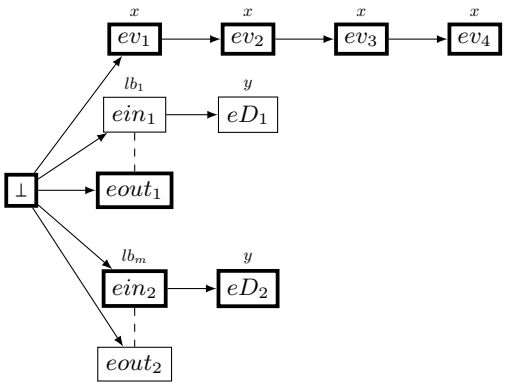
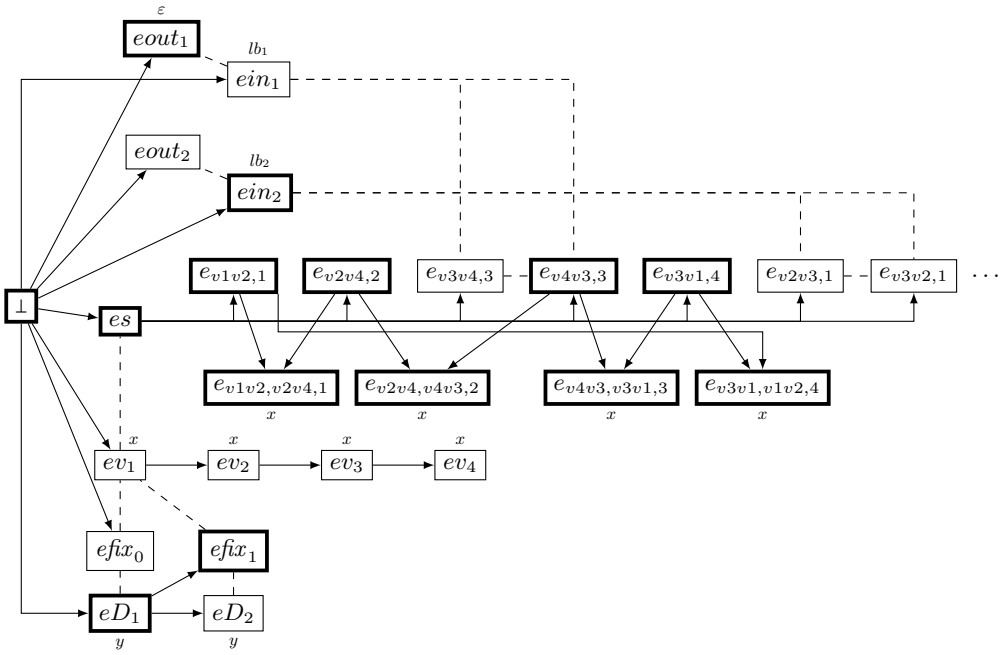


Fig. 6: Example Event structure $\mathcal{E}_1$



For space reasons only selected events of the form $e_{f,i}$ and $e_{f,f',i}$ are drawn. For clarity we name the edges $\vec{f}$ and $\overleftarrow{f}$ by the vertices they connect, e.g. $v1v2$ and $v2v1$.

Fig. 7: Example Event structure $\mathcal{E}_2$