# Symbolic Causality Checking using SAT-Solving

Adrian Beer[1], Uwe Kühne[2], Florian Leitner-Fischer[1],
Stefan Leue[1], Rüdiger Prem[2]

[1] *University of Konstanz,* [2] *Airbus Defence and Space*

**Abstract.** Causality Checking is an automated technique supporting a formal causality analysis of system models. In this paper we present a symbolic variant of causality checking. The proposed approach is based on bounded model checking and SAT solving. We show that this symbolic approach leads to a causality checking method which is efficient for large and complex systems. The technique is evaluated on industrial sized system models and compared to an existing explicit state causality checking approach.

## 1 Introduction

The size and complexity of modern software-driven safety critical systems is increasing at a high rate. In this situation, classical manual safety analysis techniques like reviewing, fault tree analysis [20] and failure mode and effect analysis [13] can only be applied to very limited parts of the architecture of a system. Furthermore, these techniques are more suitable for analyzing faults in hardware systems rather than in software driven embedded systems. The demand for automated methods and tools supporting the safety analysis of the architecture of software-driven safety-critical systems is growing.

In previous work, an algorithmic, automated safety-analysis technique called *causality checking* was proposed [16]. Causality checking is based on model checking. In model checking, the model of the system is given in a model checker specific input language. The property is typically given in some temporal logic. The model checker verifies whether the model acts within the given specifications by systematically generating the state space of the model. If the model does not fulfill the specification, an error trace leading from the initial state of the model to the property violation is generated. One trace only represents one execution of the system. In order to understand all possibilities of how an error can occur in a system, all possible error traces have to be generated and inspected. Manually locating reasons for property violations using these traces is problematic since they are often long, and typically large numbers of them can be generated in complex systems. Causality checking is an algorithmic, automated analysis technique working on system traces which supports explaining why a system violates a property. It uses an adaption of the notion of actual causality proposed by Halpern and Pearl [12]. The result of the causality checking algorithm is a combination of events that are causal for an error to happen. The event combinations are represented by formulae in Event Order Logic (EOL) [18], which can be fully translated into LTL, as is shown in [3]. The EOL formulae produced by causality checking represent the causal events in a more compact way than counterexamples since they only contain the events and the relation between those events that are considered to be causal for a property violation. It was shown that the explicit-state causality checking approach is efficient for system models for which state-of-the-art explicit model checking is efficient as well [16].

Although the explicit-state causality checking method was shown to be efficient for small to medium sized models, for system models that cannot be efficiently processed by explicit state model checkers the causality computation is also not efficient. In this paper we introduce a new causality checking approach based on Bounded Model Checking (BMC) [6]. BMC can efficiently find errors in very large systems where explicit model checking runs out of resources. One drawback of BMC is that it is not a complete technique since it cannot prove the absence of errors in a system beyond a predefined bound on the length of the considered execution traces. For the proposed symbolic causality checking method this means that completeness for the computed causalities can only be guaranteed for system runs up to the given bound.

In explicit causality checking all traces through a system have to be generated in order to gain insight into the causal events. The symbolic causality checking approach presented in this paper uses the underlying SAT-solver of the bounded model checker in order to generate the causal event combinations in an iterative manner. This means that only those error traces are generated that give new insight into the system. Traces that do not give new information are automatically excluded from the bounded model checking algorithm by constraining the SAT-solver with the already known information. With this technique a large number of error traces can be ruled out that would need to be considered in the explicit approach, which contributes to the efficiency of the symbolic approach. We implemented this approach as an addition to the NuSMV2 model checker [9].

*Structure of the Paper.* In Section 2 we will present the foundations of our work. In Section 3 we describe the proposed symbolic approach to causality checking. In Section 4 we experimentally evaluate the symbolic causality checking approach by comparing it to explicit-state causality checking. Related work will be discussed in Section 5 before we conclude in Section 6.

## 2 Preliminaries

### 2.1 Running Example

We will illustrate the formal framework that we present in this paper using the running example of a simple railroad crossing system. In this system, a train can approach the crossing (Ta), enter the crossing (Tc), and finally leave the crossing (Tl). Whenever a train is approaching, the gate shall close (Gc) and will open again when the train has left the crossing (Go). It might also be the case that the gate fails (Gf). The car approaches the crossing (Ca) and crosses the crossing if the gate is open (Cc) and finally leaves the crossing (Cl). We are interested in finding those events that are causal for the hazard that the car and the train are in the crossing at the same time.

### 2.2 System Model

The system model that we use in this paper is that of a transition system [2]:

**Definition 1 (Transition System).** *A transition system $M$ is a tuple ($S$, $\mathcal{A}$, $\rightarrow$, $I$, $AP$, $L$) where $S$ is a finite set of states, $\mathcal{A}$ is a finite set of actions/events, $\rightarrow \subseteq S \times \mathcal{A} \times S$ is a transition relation, $I \subseteq S$ is the set of initial states, $AP$ is the set of atomic propositions, and $L$:$S \rightarrow 2^{AP}$ is a labeling function.*

**Definition 2 (Execution Trace).** *An execution trace $\pi$ in $M$ is defined as an alternating sequence of states $s \in S$ and actions $a \in \mathcal{A}$ ending with a state. $\pi = s_0$ $\alpha_1$ $s_1$ $\alpha_2$ $s_2$ ... $\alpha_n$ $s_n$, s.t. $s_i \xrightarrow{\alpha_{i+1}} s_{i+1}$ for all $0 \leq i < n$.*

An execution sequence which ends in a property violation is called an error trace or a counterexample. In the railroad crossing example, $s_0 \xrightarrow{\text{Ta}} s_1 \xrightarrow{\text{Gf}} s_2 \xrightarrow{\text{Tc}} s_3 \xrightarrow{\text{Ca}} s_4 \xrightarrow{\text{Cc}} s_5$ is a counterexample, because the train and the car are inside the crossing at the same time.

### 2.3 Linear Temporal Logic

Linear Temporal Logic (LTL) [19] is a propositional modal logic based on a linear system execution model. An LTL formula can be used to express properties of infinite paths in a given system model.

**Definition 3 (Syntax of Linear Temporal Logic).** *An LTL formula $\varphi$ over a set of atomic propositions* AP *is defined according to the following grammar:*

$$\begin{aligned}\varphi ::= \ \text{TRUE} \ &| \ a \ | \ \varphi_1 \wedge \varphi_2 \ | \varphi_1 \vee \varphi_2 \ | \neg\varphi \ | \bigcirc\varphi \ | \Box\varphi \\ &| \Diamond\varphi \ | \varphi_1 \ \boldsymbol{U} \ \varphi_2\end{aligned}$$

*where $a \in$* AP.

The operators $\bigcirc$, $\Box$, $\Diamond$ and **U** are used to express temporal behavior, such as *"in the next state sth. happens"*($\bigcirc$), *"eventually sth. happens"*($\Diamond$) and *"sth. is always true"*($\Box$). The **U**-operator denotes the case that *"$\varphi_1$ has to be true until $\varphi_2$ holds"*. We use $M \vDash_l \varphi$ to express that an LTL formula $\varphi$ holds on a system model $M$ and $\pi \vDash_l \varphi$ for a execution trace in $M$.

The property that we want to express in the railroad crossing is that the train and the car shall never be in the crossing at the same time: $\Box\neg(\text{Tc} \wedge \text{Cc})$.

The properties that are expressible in LTL can be separated into two classes, safety and liveness properties. Safety properties can be violated by a finite prefix of an infinite path, while liveness properties can only be violated by an infinite path. For now, causality checking has only been defined for safety properties.

### 2.4 Event Order Logic

As was shown in [17], Event Order Logic (EOL) can express occurrence and ordering constraints on the actions occurring along a trace. Every EOL formula can be translated into an equivalent LTL formula [3].

**Definition 4 (Syntax of the Event Order Logic).** *Simple event order logic formulae are defined over the set $A$ of event variables:*

$$\phi ::= a \mid \phi_1 \wedge \phi_1 \mid \phi_1 \vee \phi_2 \mid \neg\phi$$

*where $a \in A$ and $\phi$, $\phi_1$ and $\phi_2$ are simple EOL formulae. Complex EOL formulae are formed according to the following grammar:*

$$\begin{aligned} \psi ::= \phi \mid \psi_1 \wedge \psi_1 \mid \quad \psi_1 \vee \psi_2 \quad \mid \psi_1 \mathbin{\dot\wedge} \psi_2 \mid \psi_1 \mathbin{\dot\wedge_{[}} \phi \\ \mid \psi_1 \mathbin{\dot\wedge_{]}} \phi \mid \psi_1 \mathbin{\dot\wedge_{<}} \phi \mathbin{\dot\wedge_{>}} \psi_2 \end{aligned}$$

*where $\phi$ is a simple EOL formula and $\psi$, $\psi_1$ and $\psi_2$ are complex EOL formulae.*

We define that a transition system $M$ satisfies the EOL formula $\psi$, written as $M \vDash_e \psi$, iff $\exists \pi \in M.\ \pi \vDash_e \psi$. The informal semantics of the operators can be given as follows.

- $\psi_1 \mathbin{\dot\wedge} \psi_2$: $\psi_1$ has to happen before $\psi_2$.
- $\psi_1 \mathbin{\dot\wedge_{[}} \phi$: $\psi_1$ has to happen at some point and afterwards $\phi$ holds forever.
- $\phi \mathbin{\dot\wedge_{]}} \psi_1$: $\phi$ has to hold until $\psi_1$ holds.
- $\psi_1 \mathbin{\dot\wedge_{<}} \phi \mathbin{\dot\wedge_{>}} \psi_2$: $\psi_1$ has to happen before $\psi_2$, and $\phi$ has to hold all the time between $\psi_1$ and $\psi_2$.

For example, the formula Gc $\dot\wedge$ Tc states that the gate has to close before the train enters the crossing. The full formal semantics definition for EOL is given in [18].

### 2.5 Causality Reasoning

Our goal is to identify the events that cause a system to reach a property violating state. Therefore, it is necessary to formally define what "cause" means in our context. We will use the same definition of causality that was proposed in [14] as an extension of the *structural equation model* by Halpern and Pearl [12]. In particular this extension accounts for considering the order of events in a trace to be causal. For example, an event $a$ may always occur before an event $b$ for an error to happen, but if $b$ occurs first and $a$ afterwards there is no error. In this case, $a$ occurring before $b$ is considered to be causal for the error to happen.

**Definition 5 (Cause for a property violation [12,16]).** *Let $\pi$,$\pi'$ and $\pi''$ be paths in a transition system $M$. The set of event variables is partitioned into sets $Z$ and $W$. The variables in $Z$ are involved in the causal event chain for a property violation while the variables in $W$ are not. The valuations of the variables along a path $\pi$ are represented by $val_z(\pi)$ and $val_w(\pi)$, respectively. $\psi_\wedge$ denotes the rewriting of an EOL formula $\psi$ where the ordering operator $\dot\wedge$ is replaced by the normal EOL operator $\wedge$, all other EOL operators are left unchanged. An EOL formula $\psi$ consisting of event variables $X \subseteq Z$ is considered to be a cause for an effect represented by the violation of an LTL property $\varphi$, if the following conditions hold:*

- *AC 1: There exists an execution $\pi$ for which both $\pi \vDash_e \psi$ and $\pi \nvDash_l \varphi$*

- *AC 2.1: $\exists \pi'$ s.t. $\pi' \not\models_e \psi \wedge (val_x(\pi) \neq val_x(\pi') \vee val_w(\pi) \neq val_w(\pi'))$ and $\pi' \models_l \varphi$. In other words, there exists an execution $\pi'$ where the order and occurrence of events is different from execution $\pi$ and $\varphi$ is not violated on $\pi'$.*
- *AC 2.2: $\forall \pi''$ with $\pi'' \models_e \psi \wedge (val_x(\pi) = val_x(\pi'') \wedge val_w(\pi) \neq val_w(\pi''))$ it holds that $\pi'' \not\models_l \varphi$ for all subsets of $W$. In words, for all executions where the events in $X$ have the value defined by $val_x(\pi)$ and the order defined by $\psi$, the value and order of an arbitrary subset of events on $W$ has no effect on the violation of $\varphi$.*
- *AC 3: The set of variables $X \subseteq Z$ is minimal: no subset of $X$ satisfies conditions AC 1 and AC 2.*
- *OC 1: The order of events $X \subseteq Z$ represented by the EOL formula $\psi$ is not causal if the following holds: $\pi \models_e \psi$ and $\pi' \not\models_e \psi$ and $\pi' \not\models_e \psi_\wedge$*

The EOL formula $\mathrm{Gf} \wedge ((\mathrm{Ta} \wedge (\mathrm{Ca} \dot{\wedge} \mathrm{Cc})) \dot{\wedge}_< \neg \mathrm{Cl} \dot{\wedge}_> \mathrm{Tc})$ is a cause for the occurrence of the hazard in the railroad crossing example since it fulfills all of the above defined conditions (AC 1-3, OC 1).

### 2.6 Bounded Model Checking

The basic idea of Bounded Model Checking (BMC) [6] is to find error traces, also called counterexamples, in executions of a given system model where the length of the traces that are analyzed are bounded by some integer $k$. If no counterexample is found for some bound $k$, it is increased until either a counterexample is found or some pre-defined upper bound is reached. The BMC problem is efficiently reduced to a propositional satisfiability problem, and can be solved using propositional SAT solvers. Modern SAT solvers can handle satisfiability problems in the order of $10^6$ variables.

Given a transition system $M$, an LTL formula $f$ and a bound $k$, the propositional formula of the system is represented by $[[M, f]]_k$. Let $s_0, ..., s_k$ be a finite sequence of states on a path $\pi$. Each $s_i$ represents a state at time step $i$ and consists of an assignment of truth values to the set of state variables. The formula $[[M, f]]_k$ encodes a constraint on $s_0, ..., s_k$ such that $[[M, f]]_k$ is satisfiable iff $\pi$ is a witness for $f$. The propositional formula $[[M, f]]_k$ is generated by unrolling the transition relation of the original model $M$ and integrate the LTL property in every step $s_i$ of the unrolling. The generated formula $[[M, f]]_k$ of the whole system is passed into a propositional SAT solver. The SAT solver tries to solve $[[M, f]]_k$. If a solution exists, this solution is considered to be a counterexample of the encoded LTL property.

## 3 Symbolic Causality Checking

### 3.1 Event Order Normal Form

In order to enable the processing of EOL formulas and counterexamples in the symbolic causality algorithm it is necessary to define a normal form for EOL formulas that we refer to as the event order normal form (EONF). EONF permits the unordered *and*- ($\wedge$) and *or*-operator ($\vee$) only to appear in a formula if they are not sub formulas in any ordered operator or if they are sub formulas of the between operators $\dot{\wedge}_<$ and $\dot{\wedge}_>$.

**Definition 6.** *Event Order Normal Form (EONF) The set of EOL formulas over a set $\mathcal{A}$ of event variables in event order normal form (EONF) is given by:*

$$\phi ::= a \mid \neg\phi \qquad\qquad \phi_{\wedge} ::= \phi \mid \neg\phi_{\wedge} \mid \phi_{\wedge_1} \wedge \phi_{\wedge_2}$$

$$\psi ::= \phi \mid \phi_1 \mathbin{\dot\wedge} \phi_2 \mid \phi_1 \mathbin{\dot\wedge_{[}} \phi_2 \mid \phi_1 \mathbin{\dot\wedge_{]}} \phi_2 \mid \phi_1 \mathbin{\dot\wedge_{<}} \phi_2 \mathbin{\dot\wedge_{>}} \phi_3$$

$$\psi_{\wedge} ::= \psi \mid \phi_{\wedge} \mid \psi_{\wedge_1} \wedge \psi_{\wedge_2} \mid \psi_{\wedge_1} \vee \psi_{\wedge_2}$$

*where $a \in \mathcal{A}$ and $\phi$ are simple EOL formulas only containing single events and $\phi_{\wedge}$, $\phi_{\wedge_1}$, $\phi_{\wedge_2}$ and $\phi_{\wedge_3}$ are EOL formulas only containing the $\wedge$-operator, $\psi$ is a EOL formula containing the ordered operator, and $\psi_{\wedge}$, $\psi_{\wedge_1}$ and $\psi_{\wedge_2}$ are EOL formulas containing the $\wedge$-operator and / or the $\vee$-operator which can be combined with formulas in EONF containing ordered operators.*

Every EOL formula can be transformed into an equivalent EOL formula in EONF by rewriting using the equivalence rules defined in [3]. For instance, the EOL formula Ta $\dot\wedge$ Gc $\dot\wedge$ Tc can be expressed in EONF as $\psi_{\text{EONF}} = (\text{Ta} \mathbin{\dot\wedge} \text{Gc}) \wedge (\text{Gc} \mathbin{\dot\wedge} \text{Tc}) \wedge (\text{Ta} \mathbin{\dot\wedge} \text{Tc})$.

### 3.2 EOL Matrix

For the symbolic causality computation with bound $k$ we focus on sequence of events $\pi_e = e_1 e_2 e_3 \ldots e_k$ derived from paths of type $\pi = s_0 \xrightarrow{e_1} s_1 \xrightarrow{e_2} s_2 \ldots$. We use a matrix in order to represent the ordering and occurrence of events along a trace. This matrix is called EOL matrix.

**Definition 7 (EOL matrix).** *Let $E = \{e_1, e_2, e_3, \ldots, e_k\}$ an event set and $\pi_e = e_1 e_2 e_3 \ldots e_k$ the corresponding sequence. Then a function $o$ is defined for entries where $i \neq j$:*

$$o(e_i, e_j) = \begin{cases} \{\mathsf{TRUE}\} & \text{if } e_i \mathbin{\dot\wedge} e_j \\ \varnothing & \text{otherwise} \end{cases}$$

*The EOL matrix $M_E$ is created as follows:*

$$M_E = \begin{pmatrix} \varnothing & o(e_1, e_2) & \cdots & o(e_1, e_k) \\ o(e_2, e_1) & \varnothing & \cdots & o(e_2, e_k) \\ \vdots & \vdots & \ddots & \vdots \\ o(e_k, e_1) & o(e_k, e_2) & \cdots & \varnothing \end{pmatrix}$$

*where the generated entries in the matrix are sets of events or the constant set $\{\mathsf{TRUE}\}$. The empty set $\varnothing$ is also permitted which means no relation for the corresponding event configuration was found.*

**Definition 8 (Union of EOL Matrices).** *Let $M_E, M_{E_1}, M_{E_2}$ be EOL Matrices with the same dimensions. The EOL matrix $M_E$ is the union of $M_{E_1}$ and $M_{E_2}$ according to the following rule:*

$$M_{E(i,j)} = M_{E_1(i,j)} \cup M_{E_2(i,j)} \tag{1}$$

*for every entry $(i, j)$ in the matrices.*

The union of two EOL matrices represents the component-wise disjunction of two matrices. The EOL matrix $M_E$ for an example event sequence in the railroad crossing $\pi$ = Ca Cc Gf and a refinement EOL Matrix $M'_E = M_E \cup M_{E_{\pi'}}$ using the sequence $\pi'$ = Gf Ca Cc is created as follows:

$$\begin{array}{l} e_1 = \mathrm{Ca} \\ e_2 = \mathrm{Cc} \\ e_3 = \mathrm{Gf} \end{array} \quad M_E = \begin{pmatrix} \varnothing & \{\mathsf{TRUE}\} & \{\mathsf{TRUE}\} \\ \varnothing & \varnothing & \{\mathsf{TRUE}\} \\ \varnothing & \varnothing & \varnothing \end{pmatrix} \quad M'_E = \begin{pmatrix} \varnothing & \{\mathsf{TRUE}\} & \{\mathsf{TRUE}\} \\ \varnothing & \varnothing & \{\mathsf{TRUE}\} \\ \{\mathsf{TRUE}\} & \{\mathsf{TRUE}\} & \varnothing \end{pmatrix} \qquad (2)$$

### 3.3 EOL Matrix to Propositional Logic Translation

In order to use the information stored in the EOL Matrix in the symbolic causality checking algorithm a translation from the matrix into propositional logic is needed. First the Matrix is translated into an EOL formula in EONF and afterwards the EOL formula is translated into propositional logic.

**Definition 9 (Translation from EOL matrix to EOL formula).** *Let $M_E$ a EOL matrix which contains the EOL formula $\psi_E$ and the event set $E$. $M_{E(i,j)}$ is the set of events in the entry $(i,j)$ in $M_E$ and $e_{(i,j)} \in M_{E(i,j)}$. $e_i$ and $e_j$ denote the ordered events, respectively. Then $\psi_E$ is defined as follows:*

$$\psi_E = \bigwedge_{i=0}^{i=k} \bigwedge_{j=0}^{j=k} \begin{cases} e_i \wedge e_j & \text{if } e_{(i,j)} = \{\mathsf{TRUE}\} \text{ and } e_{(j,i)} = \{\mathsf{TRUE}\} \text{ and } i \neq j \\ e_i \mathbin{\dot\wedge} e_j & \text{if } e_{(i,j)} = \{\mathsf{TRUE}\} \text{ and } e_{(j,i)} \neq \{\mathsf{TRUE}\} \text{ and } i \neq j \end{cases}$$

**Lemma 1 (without proof).** *An EOL formula $\psi_E$ obtained from Definition 9 from an EOL matrix $M_E$ is always in Event Order Normal Form (EONF).*

Using this translation the EOL Matrix from Equation 2 is translated into the following EOL formula in EONF: $\psi_{\mathrm{EONF}} = (\mathrm{Ca} \mathbin{\dot\wedge} \mathrm{Cc}) \wedge (\mathrm{Gf} \wedge \mathrm{Ca}) \wedge (\mathrm{Gf} \wedge \mathrm{Cc})$.

The generated EOL formula can be efficiently translated into an equivalent LTL formula as it was shown in [3]. As mentioned in Section 2.3, only safety properties are considered for the symbolic causality checking approach. Since safety properties can only be violated by finite prefixes of system executions, it is necessary for our purposes to adapt the definition of a bounded semantics for LTL as defined in [6]:

**Definition 10 (Bounded Semantics for LTL).** *Let $k \geq 0$, and let $\pi$ be a prefix of an infinite path and $\pi_e = e_0 e_1 e_2 \ldots$ the sequence of events of $\pi$. Let $\psi_{LTL}$ an LTL formula obtained by translating an EOL formula $\psi$ into LTL. $\psi_{LTL}$ is valid along $\pi$ up to bound $k$, represented by $\pi \vDash_k^0 \psi_{LTL}$, if the following holds:*

$$\begin{array}{lll}
\pi \vDash_k^i p & \textit{iff} & p = e_i \\
\pi \vDash_k^i \neg p & \textit{iff} & p \neq e_i \\
\pi \vDash_k^i f \wedge g & \textit{iff} & \pi \vDash_k^i f \textit{ and } \pi \vDash_k^i g \\
\pi \vDash_k^i f \vee g & \textit{iff} & \pi \vDash_k^i f \textit{ or } \pi \vDash_k^i g \\
\pi \vDash_k^i \Box f & \textit{iff} & \forall j, i \leq j \leq k.\ \pi \vDash_k^j f \\
\pi \vDash_k^i \Diamond f & \textit{iff} & \exists j, i \leq j \leq k.\ \pi \vDash_k^j f \\
\pi \vDash_k^i \bigcirc f & \textit{iff} & i < k \textit{ and } \pi \vDash_k^{i+1} f \\
\pi \vDash_k^i f \boldsymbol{U} g & \textit{iff} & \exists j, i \leq j \leq k.\ \pi \vDash_k^j g \textit{ and } \forall n, i \leq n \leq k.\ \pi \vDash_k^n f
\end{array}$$

For converting the LTL formula $\psi_{\text{LTL}}$ into a propositional logic formula, the standard translation scheme for translating LTL into propositional logic for a given bound $k$ as described in [6] is used.

### 3.4 The Symbolic Causality Checking Algorithm

In order to decide whether a certain event is causal for a property violation it is necessary to know that there exists a counterexample which violates a property (AC 1). Additionally, there have to exist other traces with other events and orderings that do not lead into a violating state (AC 2). As a consequence all combinations of events have to be known. In the explicit state causality checking approach [16], all paths through a system have to be computed in order to find all causal events and orderings for a property violation. In order to overcome the drawback of calculating all possible paths we propose the use of an iterative scheme involving BMC and symbolic constraints on the underlying SAT solver. The symbolic constraint is used in order to find only those paths that contain new information on event orderings and occurrences. This new information is used to strengthen the constraints on the SAT Solver.

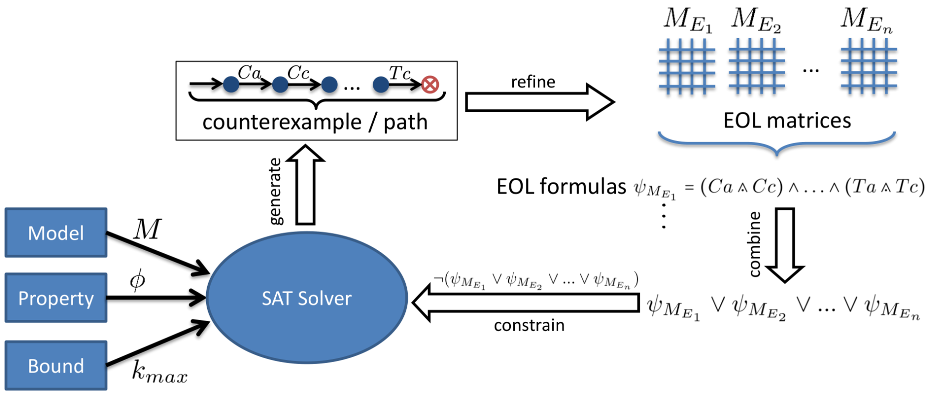
$M_{E_1}$
$M_{E_2}$
$M_{E_n}$
...
$Ca$
$Cc$
$Tc$
...
counterexample / path
refine
EOL matrices
generate
EOL formulas $\psi_{M_{E_1}} = (Ca \mathbin{\dot\wedge} Cc) \wedge \ldots \wedge (Ta \mathbin{\dot\wedge} Tc)$
combine
Model
Property
Bound
$M$
$\phi$
$k_{max}$
SAT Solver
$\neg(\psi_{M_{E_1}} \vee \psi_{M_{E_2}} \vee \ldots \vee \psi_{M_{E_n}})$
constrain
$\psi_{M_{E_1}} \vee \psi_{M_{E_2}} \vee \ldots \vee \psi_{M_{E_n}}$

**Fig. 1.** The iteration schema of the symbolic causality checking algorithm

In Figure 1 the informal iteration schema of the proposed algorithm is presented. The inputs are the model $M$, the property $\phi$ and an upper bound $k_{\max}$ for the maximum length of the considered paths. The algorithm starts at level $k = 0$:

**Step 1: Generation of Traces.** The model $M$ together with the LTL property $\phi$ and the bound $k$ is converted into a propositional logic formula $[[M, \neg\phi]]_k$. $[[M, \neg\phi]]_k$ is inserted into a SAT solver. The SAT solver tries to find a path that fulfills the given formula. If such a path is found, the algorithm has discovered a counterexample and continues at step 2. Otherwise the bound $k$ is increased until the first counterexample is found or the maximum bound $k_{\max}$ is reached.

**Step 2: Matching of EOL Matrices.** When a new path $\pi$ is discovered the set of events $E_1$ occurring on this trace is compared to the already known EOL matrices. If there is an EOL matrix $M_{E_2}$ covering a set of events $E_2$ and if

$E_1 = E_2$, the newly discovered orderings of events in $E_1$ is used to refine the EOL matrix $E_2 \coloneqq E_2 \cup E_1$ according to Definition 8. If there is no matching matrix, a new EOL matrix is created representing a new class of causes [16] containing the ordering of events in $\pi$.

**Step 3: Combination of new constraints.** All EOL matrices $M_{E_i}$ are translated into EOL formulas $\psi_{M_{E_i}}$ according to Definition 9. The translated EOL formulas $\psi_{M_{E_i}}$ are combined disjunctively. In order to exclude the already found orderings from being found again in the next iteration, the result is negated which results in $\varphi' = \neg(\psi_{M_{E_1}} \vee \psi_{M_{E_2}} \vee \ldots \vee \psi_{M_{E_n}})$ with $n$ the number of EOL matrices that have been computed so far.

**Step 4: Constraining the SAT Solver.** The formula $\varphi'$ is translated into a propositional logic formula $[[\varphi']]_k$ for a given bound $k$. $[[\varphi']]_k$ is then used a as an additional constraint for the SAT Solver (Definition 10). Afterwards, the algorithm starts over at Step 1.

When the algorithm terminates, the result is stored in the EOL matrices $M_{E_i}, 0 \leq i \leq n$ where $n$ is the number of EOL matrices found during the search.

### 3.5 Soundness and Completeness

We show that the results generated with the described algorithm are sound up to the pre-defined maximum bound $k$. Afterwards we will discuss the completeness of the symbolic causality algorithm.

**Definition 11 (Candidate Set (adapted from [18])).** *Let $n$ the number of EOL matrices $M_{E_i}, 0 \leq i \leq n$, $\neg\phi$ the negation of an LTL property and $\sum_C$ the set of all possible counterexamples. The disjunction of all EOL formulas $\psi = \bigvee_{i=0}^{n} \psi_{M_{E_i}}$ generated from the matrices $M_{E_i}$, is a compact description of all counterexamples. The candidate set $\boldsymbol{CS}(\neg\phi) = \{\pi \in \sum_C \mid \forall \pi' \in \sum_C . \pi' \subseteq \pi \Rightarrow \pi' = \pi\}$ contains the minimal set of counterexamples through the system that satisfy $\psi$.*

Notice that the candidate set is minimal in the sense that removing an event from some trace in the candidate set means that the resulting trace no longer is a counterexample.

**Theorem 1.** *The candidate set satisfies the causality conditions described in Definition 5.*

**Event Non-Occurrence Detection.** According to the AC 2.2 test the occurrence of events that are not considered as causal must not prevent the effect from happening. In other words, the non-occurrence of an event can be causal for a property violation. Therefore, we have to search such events and include their non-occurrence in the EOL formulas. In Figure 2 an example is presented which explains this procedure for an EOL formula $\psi = \text{Ca} \wedge \text{Cc} \wedge \text{Ta} \wedge \text{Gc} \wedge \text{Tc}$. Trace 1 is the minimal trace ending in a property violation. Trace 2 is non-minimal and also ends in a property violation with the events *Ca, Cc, Ta, Gc, Gf, Tc*. In trace 3 a new event *Cl* appears between *Cc* and *Ta* and no property violation is detected. This means that the appearance of the event has prevented

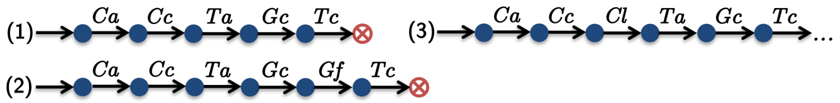

**Fig. 2.** Three example traces for the EOL-formula $\psi$ = Ca $\dot{\wedge}$ Cc $\dot{\wedge}$ Ta $\dot{\wedge}$ Gc $\dot{\wedge}$ Tc. Trace 1 is the minimal trace. While trace 2 (non-minimal) ends in a property violation, trace 3 does not.

the property violation. In order to transform this appearance into a cause for the hazard, the occurrence is negated and introduced into the EOL formula $\psi = \ldots Cc \dot{\wedge}_< \neg Cl \dot{\wedge}_> Ta \ldots$ The new clause states that *"if between 'the car is on the crossing' and 'the train is approaching the crossing', 'the car does NOT leave the crossing', the hazard does happen"*. In other words: The non-occurrence of $Cl$ is causal for the property violation.

A second pass of the algorithm needs to be done in order to find these non-occurrences. For this second pass the input parameters have to be altered compared to the first pass. The EOL Matrix definition also needs to be extended in order to account for the the possible non-occurrence of events.

**Definition 12 (Extended EOL matrix).** *Let $E = \{e_1, e_2, e_3, \ldots, e_k\}$ an event set and $\pi_e = e_1 e_2 e_3 \ldots e_k$ the corresponding sequence. The function $o$ is defined for entries where $i \neq j$ and the function $d$ is defined for entries where $i = j$:*

$$o(e_i, e_j) = \begin{cases} \{\mathsf{TRUE}\} & \text{if } e_i \dot{\wedge} e_j \\ \phi & \text{if } e_i \dot{\wedge}_< \phi \dot{\wedge}_> e_j \\ \varnothing & \text{otherwise} \end{cases} \qquad d(e_i) = \begin{cases} \phi & \text{if } \phi \dot{\wedge}_] e_i \\ \varnothing & \text{otherwise} \end{cases}$$

*The EOL matrix $M_E$ is created as follows:*

$$M_E = \begin{pmatrix} d(e_1) & o(e_1, e_2) & \cdots & o(e_1, e_k) \\ o(e_2, e_1) & d(e_2) & \cdots & o(e_2, e_k) \\ \vdots & \vdots & \ddots & \vdots \\ o(e_k, e_1) & o(e_k, e_2) & \cdots & d(e_k) \end{pmatrix}$$

*where the generated entries in the matrix are sets of events or the constant set $\{\mathsf{TRUE}\}$. The empty set $\varnothing$ indicates that no relation for the corresponding event configuration was found.*

The function $o$ gives true as a result if $e_1$ occurs before $e_2$ and gives $\phi$ as a result if $e_1$ occurs before $e_2$ and $\phi$ is true between $e_1$ and $e_2$. The function $d$ returns $\phi$ if $\phi$ is always occurring before $e_i$. According to the extended EOL Matrix definition it is possible to insert EOL formulas of the form $e_i \dot{\wedge}_< \phi \dot{\wedge}_> e_j$ and $\phi \dot{\wedge}_] e_i$ into the matrix. This can be used to insert conditions such as $\psi = Cc \dot{\wedge}_< \neg Cl \dot{\wedge}_> Ta$. The special case $e \dot{\wedge}_[ \phi$ is not considered here because this will never occur when analyzing safety properties. If a hazard state is reached no future occurrence of any event can prevent the hazard. The formula $e \dot{\wedge}_[ \phi$ would encode such a behavior.

**Definition 13 (Extended Translation for AC 2.2).** *Let $M_E$ an EOL matrix which contains the EOL formula $\psi_E$ and the event set $E$. $M_{E(i,j)}$ is the set of* *events in the entry $(i,j)$ in $M_E$ and $e_{(i,j)} \in M_{E(i,j)}$. $e_i$ and $e_j$ denote the ordered events, respectively. Then $\psi_E$ is defined as follows:*

$$\psi_E = \bigwedge_{i=0}^{i=k} \bigwedge_{j=0}^{j=k} \bigwedge_{\substack{\forall e_{(i,j)} \\ \in M_{E(i,j)}}} \begin{cases} e_i \wedge e_j & \textit{if } e_{(i,j)} = \textsf{\textit{TRUE}} \textit{ and } e_{(j,i)} = \textsf{\textit{TRUE}} \textit{ and } i \neq j \\ e_i \mathbin{\dot\wedge} e_j & \textit{if } e_{(i,j)} = \textsf{\textit{TRUE}} \textit{ and } e_{(j,i)} \neq \textsf{\textit{TRUE}} \textit{ and } i \neq j \\ e_i \mathbin{\dot\wedge_<} \phi \mathbin{\dot\wedge_>} e_j & \textit{if } \phi = e_{(i,j)} \textit{ and } i \neq j \\ \phi \mathbin{\dot\wedge_]} e_i & \textit{if } e_{(i,j)} = \phi \textit{ and } i = j \end{cases}$$

The translation from EOL formulas into LTL and further into propositional logic is done according to Definition 10.

*Input parameters to the non-occurrence detection.* In the second pass of the algorithm, the input parameters for the SAT solver have to be changed. Now the algorithm searches for paths in the system that do not end in a property violation, while fulfilling the EOL formulas that have been found so far. For instance, in Figure 2 trace 3 does also fulfill the displayed EOL formula. In order to find those paths the inputs to the SAT solver are the original LTL property $\phi$, the original EOL formulas $\psi_{M_{E_i}}$, the model and the bound $k$. The paths obtained with this method contain the events that prevent the property violation. These events are inserted into a matching EOL matrix. Since the EOL matrices are used to search for the new paths there is always a matching matrix available to the algorithm. The matching of EOL matrices for the AC2.2 condition is defined as follows.

**Definition 14 (Matching of paths to EOL Matrix for AC2.2.).** *Let $\pi$ be a path discovered by the second pass, $E_\pi$ the set of events occurring on $\pi$ and $E_i$ the event sets of all $n$ EOL matrices. Then the matching EOL matrix is defined according to the following function:*

$$\mathit{match}(\pi) = \{M_{E_i} | \exists i, 0 \leq i \leq n.\ \forall j, 0 \leq j \leq n,\ m_i = \mathit{max}\left(|E_j \cap E_\pi|\right)\}$$

The *match* function returns the EOL matrix $M_{E_i}$ whose event set $E_i$ has the largest number of common events with the event set $E_\pi$. Note that there is always a unique maximum for this number: From the definition of the matching of EOL matrices in the first and the second pass of the algorithm two paths containing the same events are merged into one EOL matrix. This means all EOL matrices contain a unique set of events.

The refinement of the matching EOL matrix is conducted according to Definition 8 and 12.

**Theorem 2 (Soundness of AC2.2).** *For every EOL matrix $M_E$ with the number of events $i = |E|$ the condition AC2.2 is fulfilled for a maximum number of events $x$ that prevent the property violation from happening and $x = k_{max} - i$.*

**Completeness.** With symbolic causality checking we can only find event combinations and their orderings up to a predefined bound $k_{\text{max}}$.

**Theorem 3.** *All EOL matrices discovered with the symbolic algorithm are complete in terms of conditions AC1,AC2.1, AC2.2, AC3 and OC1 up to the bound $k_{max}$.*

```
   Data: $\phi$ the property, $S$ the model, $k_{max}$ the maximum depth of the search
   Result: The causal events for a property violation stored in $M_{\text{list}}$
1  $k := 0$;
2  $\psi :=$ FALSE;                                                /*EOL formula*/
3  $M_{\text{list}} :=$ empty List of EOL matrices;
4  while $k < k_{max}$ do
5      $\pi :=$ solve($\neg\phi$, $S$, $\neg\psi$, $k$);
6      while $\pi$ is not empty do
7          $m :=$ getMatchingMatrix($M_{\text{list}}$, $\pi$);              /*Definition 8*/
8          refineEOLMatrix($m$, $\pi$);                              /*Definition 7*/
9          $\psi :=$ getEOLformula($M_{\text{list}}$);                     /*Definition 9,10*/
10         $\pi :=$ solve($\neg\phi$, $S$, $\neg\psi$, $k$);
11     end
12     $\pi :=$ solve($\phi$, $M$, $\psi$);
13     while $\pi$ is not empty do
14         $m :=$ getMatchingMatrixAC2_2($M_{\text{list}}$, $\pi$);         /*Definition 14*/
15         refineEOLMatrixAC2_2($m$, $\pi$);                         /*Definition 12*/
16         $\psi :=$ getEOLformulaAC2_2($M_{\text{list}}$);                /*Definition 13,10*/
17         $\pi :=$ solve($\neg\phi$, $S$, $\neg\psi$, $k$);
18     end
19     $k =: k + 1$;
20 end
```

**Algorithm 1:** Symbolic causality checking algorithm

A proof can be built via structural induction over the generation of the EOL matrices using the minimality argument of the discovered counterexamples.

The completeness of condition AC2.2 is linked to the soundness of this condition and can be proven up to a certain number of events that prevent the property violation from happening. The completeness depends on the number of events in all EOL matrices and the upper bound $k_{\text{max}}$. For example, in Figure 2 trace 3 is at least one step longer than the path resulting in a property violation. This means that if, for example, the maximum bound for the algorithm is set to 5, trace 1 violating the property is found, but trace 3 is not found.

**The Algorithm.** The pseudo code for the Symbolic Causality Checking Algorithm is presented in Algorithm 1. The function *solve* (Line 5, 10, 12 and 17) converts the input parameters into propositional logic formulas and runs the SAT solver. The result of *solve* is a path of length $k$ satisfying the given constraints.

## 4 Evaluation

In order to evaluate the proposed approach, we have implemented the symbolic causality checking algorithm within the symbolic model checker NuSMV2 [9]. Our CauSeMV extension of NuSMV2 computes the causality relationships for a given NuSMV model and an LTL property. The models that we analyze are the Railroad example from Section 2.1, an Airbag Control Unit [1], an Airport Surveillance Radar System (ASR) [4] and a automotive Electronic Control Unit (AECU) that we developed together with an industrial partner. The NuSMV

| | states | transitions | paths (explicit) | iterations (symbolic) |
|---|---|---|---|---|
| Railroad | 133 | 237 | 47 | 6 |
| Airbag | 155,464 | 697,081 | 20,300 | 24 |
| ASR1 | $1 \cdot 10^6$ | $7 \cdot 10^6$ | $1 \cdot 10^6$ | 27 |
| ASR2a | $4,6 \cdot 10^7$ | $3,3 \cdot 10^8$ | $1.5 \cdot 10^7$ | 32 |
| AECU | $7.5 \cdot 10^7$ | $8.6 \cdot 10^8$ | - | 70 |
| ASR2b | $1 \cdot 10^{12}$ | $1 \cdot 10^{13}$ | - | 208 |

**Table 1.** Model sizes in the explicit case and iterations needed for the symbolic approach.

models used in the experiments were automatically synthesized from higher-level design models using the QuantUM tool [15]. The ASR model consists of 3 variants. In the first variant there is only one computation channel for the radar screen (ASR1). In the second and third variant models there are two identical computation channel to raise the availability of the system. In the first two channel variant model the availability of a second channel is modeled by a counter counting component errors (ASR2a), while in the second variant the second channel is a complete copy of the first channel (ASR2b).

All experiments were performed on a PC with an Intel Xeon Processor with 8 Cores (3.60 Ghz) and 144GBs of RAM. We compare our results with the results for the explicit state causality checking approach presented in [16], which were performed on the same computer. For all case studies, a maximum bound of $k = 20$ was chosen. The explicit approach is prallelized across all 8 cores, while the symbolic approach only uses one core. In Table 1 the sizes of the different analyzed models are shown. Additionally we compare the number of paths that have to be stored for the explicit causality computation to the iterations needed in the symbolic setting. For the AECU and the ASR2b the number of traces in the explicit case could not be computed, because the experiments run out of memory.

| | | RT (sec.) | Mem. (MB) | | | RT (sec.) | Mem. (MB) |
|---|---|---|---|---|---|---|---|
| Railroad | explicit | 0.73 | 17.9 | ASR2a | explicit | 91.22 | 826.73 |
| | symbolic | 17.16 | 121.55 | | symbolic | 186.48 | 300.54 |
| Airbag | explicit | 1.61 | 18.53 | AECU | explicit | 238.13 | 10,900.00 |
| | symbolic | 34.55 | 192.36 | | symbolic | 63.0 | 183.7 |
| ASR1 | explicit | 9.24 | 50.97 | ASR2b | explicit | OOM | OOM |
| | symbolic | 50.97 | 303.34 | | symbolic | 2,924.74 | 1,452.45 |

**Table 2.** Experimental results comparing the explicit state approach to the symbolic approach for $k_{max} = 20$.

**Discussion** Table 2 presents a comparison of the computational resources needed to perform the explicit and the symbolic causality checking approaches. In order to make the values comparable we limit the search depth for the explicit approach to $k_{max} = 20$ as we have done for the symbolic approach.

The results illustrate that for the comparatively small railroad crossing model, the airbag model as well as the ASR1 model the explicit state causality checking outperforms the symbolic approach both in terms of time and memory. For the ASR2 and the AECU models the symbolic approach uses less memory and finishes the computation faster than in the explicit case. These results reflect a frequent encountered observation when comparing explicit state and symbolic techniques: For small models explicit state model checking is faster and uses less memory since the bounded model checker faces a lot of memory overhead due to the translation of the system into propositional logic. On the other hand, for large models such as ASR2 and AECU the explicit techniques need a lot of memory in order to explicitly store all paths needed to compute the causality classes while the SAT/BMC based symbolic approach represents whole sets of paths symbolically using propositional logic formulas.

## 5 Related Work

In [5,10,11] a notion of causality was used to explain the violations of properties in different scenarios. While [5,11] use symbolic techniques for the counterexample computation, they focus on explaining the causal relationships for a single counterexample and thus only give partial information on the causes for a property violation. All of the aforementioned techniques rely on the generation of the counterexamples prior to the causality analysis while our approach computes the necessary counterexamples on-the-fly. In [8] and [7], a symbolic approach to generate Fault Trees [20] is presented. In this approach all single component failures have to be known in advance while in our approach these failures are computed as a result of the algorithm. They do not use an explicitly defined notion of causality, contrary to what we do. The ordering and the non-occurrence of events can not be detected in this approach as being causal for a property violation.

## 6 Conclusion and Future Work

We have discussed how causal relationships in a system can be established using symbolic system and cause representations together with bounded model checking. The symbolic causality checking approach presented in this paper was evaluated on six case studies, four of them industrially sized, and compared to the explicit state causality checking approach. It was observed that symbolic causality checking outperforms explicit state causality checking on large models both in terms of computation time and memory consumption.

In future work the influence of different SAT solving strategies on the speed of discovering new event orderings in the system have to be evaluated. Furthermore, we plan to transform the EOL formulas in EONF into a compact representation in order to enable an automatic Fault Tree generation.

## References

1. Aljazzar, H., Fischer, M., Grunske, L., Kuntz, M., Leitner-Fischer, F., Leue, S.: Safety Analysis of an Airbag System Using Probabilistic FMEA and Probabilistic Counterexamples. In: Proc. of QEST 2009, Sixth International Conference on the Quantitative Evaluation of Systems. IEEE Computer Society (2009)
2. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)

3. Beer, A., Leitner-Fischer, F., Leue, S.: On the relationship of event order logic and linear temporal logic. Tech. Rep. soft-14-01, Univ. of Konstanz, Germany (January 2014), http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-14-01.pdf, available from: http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-14-01.pdf
4. Beer, A., Kühne, U., Leitner-Fischer, F., Leue, S., Prem, R.: Analysis of an Airport Surveillance Radar using the QuantUM approach. Technical Report soft-12-01, Chair for Software Engineering, University of Konstanz (2012), http://www.inf.uni-konstanz.de/soft/research/publications/pdf/soft-12-01.pdf
5. Beer, I., Ben-David, S., Chockler, H., Orni, A., Trefler, R.: Explaining counterexamples using causality. In: Proceedings of CAV 2009. LNCS, Springer (2009), http://dx.doi.org/10.1007/978-3-642-02658-4_11
6. Biere, A., Cimatti, A., Clarke, E., Zhu, Y.: Symbolic model checking without BDDs. Springer (1999)
7. Bozzano, M., Cimatti, A., Tapparo, F.: Symbolic Fault Tree Analysis for Reactive Systems. In: Proc. of ATVA 2007. LNCS, vol. 4762. Springer (2007)
8. Bozzano, M., Villafiorita, A.: Improving System Reliability via Model Checking: The FSAP/NuSMV-SA Safety Analysis Platform. In: Proc. of SAFECOMP 2003. LNCS, vol. 2788, pp. 49–62. Springer (2003)
9. Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: Nusmv 2: An opensource tool for symbolic model checking. In: Computer Aided Verification, 14th International Conference, CAV 2002. Lecture Notes in Computer Science, vol. 2404, pp. 359–364. Springer (2002)
10. Gössler, G., Métayer, D.L., Raclet, J.B.: Causality analysis in contract violation. In: Runtime Verification. LNCS, vol. 6418, pp. 270–284. Springer Verlag (2010)
11. Groce, A., Chaki, S., Kroening, D., Strichman, O.: Error explanation with distance metrics. International Journal on Software Tools for Technology Transfer (STTT) 8(3) (2006)
12. Halpern, J., Pearl, J.: Causes and explanations: A structural-model approach. Part I: Causes. The British Journal for the Philosophy of Science (2005)
13. International Electrotechnical Commission: Analysis Techniques for System Reliability - Procedure for Failure Mode and Effects analysis (FMEA), IEC 60812 (1991)
14. Kuntz, M., Leitner-Fischer, F., Leue, S.: From probabilistic counterexamples via causality to fault trees. In: Proceedings of the Computer Safety, Reliability, and Security - 30th International Conference, SAFECOMP 2011. LNCS, Springer (2011)
15. Leitner-Fischer, F., Leue, S.: QuantUM: Quantitative safety analysis of UML models. In: Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages (QAPL 2011). EPTCS, vol. 57, pp. 16–30 (2011)
16. Leitner-Fischer, F., Leue, S.: Causality checking for complex system models. In: Proc. 14th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI2013). LNCS, Springer (2013)
17. Leitner-Fischer, F., Leue, S.: On the synergy of probabilistic causality computation and causality checking. In: Proc. of the Twentieth SPIN Symposium on Model Checking Software, SPIN 2013. LNCS, Springer Verlag (2013)
18. Leitner-Fischer, F., Leue, S.: Probabilistic fault tree synthesis using causality computation. International Journal of Critical Computer-Based Systems 4, pp. 119–143 (2013)
19. Pnueli, A.: The temporal logic of programs. In: Foundations of Computer Science, 1977., 18th Annual Symposium on. pp. 46–57. IEEE (1977)
20. Vesely, W.E., Goldberg, F.F., Roberts, N.H., Haasl, D.F.: Fault Tree Handbook (2002), http://handle.dtic.mil/100.2/ADA354973

# A Appendix

## A.1 Causes for the Railroad-Crossing

$$\begin{array}{ll}
\text{Cause 1:} & \text{Cause 2:} \\
(Ca \mathbin{\dot{\wedge}} Cc) \wedge (Ca \wedge Ta) \wedge & (Ca \wedge Gf) \wedge (Ca \wedge Ta) \wedge \\
(Ca \mathbin{\dot{\wedge}_{<}} \neg Tl \mathbin{\dot{\wedge}_{>}} Ta) \wedge (Ca \mathbin{\dot{\wedge}} Gc) \wedge & (Ca \mathbin{\dot{\wedge}_{<}} \neg Tl \mathbin{\dot{\wedge}_{>}} Cc) \wedge \\
(Ca \mathbin{\dot{\wedge}} Tc) \wedge (Cc \wedge Ta) \wedge & (Ca \wedge Tc) \wedge (Ca \mathbin{\dot{\wedge}} Cc) \wedge \\
(Cc \mathbin{\dot{\wedge}_{<}} \neg Cl \mathbin{\dot{\wedge}_{>}} Gc) \wedge (Cc \mathbin{\dot{\wedge}} Gc) \wedge & (Gf \wedge Ta) \wedge (Gf \mathbin{\dot{\wedge}} Tc) \wedge \\
(Ta \mathbin{\dot{\wedge}} Tc) \wedge (Gc \mathbin{\dot{\wedge}} Tc) \wedge & (Gf \wedge Cc) \wedge (Ta \mathbin{\dot{\wedge}} Tc) \wedge \\
(Gc \mathbin{\dot{\wedge}_{<}} \neg Cl \mathbin{\dot{\wedge}_{>}} Tc) \wedge (Cc \mathbin{\dot{\wedge}} Tc) \wedge & (Tc \mathbin{\dot{\wedge}_{<}} \neg Tl \mathbin{\dot{\wedge}_{>}} Ca) \wedge \\
(Tc \mathbin{\dot{\wedge}_{<}} \neg Tl \mathbin{\dot{\wedge}_{>}} Ca) \wedge (Ta \mathbin{\dot{\wedge}} Gc) & (Tc \wedge Cc) \wedge (Cc \mathbin{\dot{\wedge}_{<}} \neg Tl \mathbin{\dot{\wedge}_{>}} tc)
\end{array}$$

Cause 1 represents the case where both the car and the train are approaching the crossing and the car stays on the crossing until the gate closes and finally the train enters the crossing. Cause 2 represents the case where the gate fails at an arbitrary point in time and the car and the train approach and enter the crossing in any possible combination. Both causes are consistent with the results from the explicit state causality checking [16].

## A.2 Proofs

*Proof (Soundness of the symbolic approach w.r.t. AC1, Theorem 1).* Let $\neg\phi$ the negated LTL property and $\psi$ the EOL formula representing the candidate set $\mathbf{CS}(\neg\phi)$. According to Definition 11, all counterexamples $\pi \in \mathbf{CS}(\neg\phi)$ are traces satisfying $\pi \vDash_l \neg\phi$. $\pi \vDash_e \psi$ holds by the definition of the creation of the EOL Matrices. Therefore AC1 holds for all $\pi \in \mathbf{CS}(\neg\phi)$. □

The proofs for the conditions AC2.1, AC3 and OC1 can be constructed in a similar way as shown in [18]. The only problem left to proof is the AC 2.2 constraint.

*Proof (proof sketch for Theorem 2: Soundness of the symbolic approach w.r.t. AC2.2).* Let $\pi \in \mathbf{CS}(\neg\phi)$ be a path of length $i$ in the candidate set of the property violation and $k_{\max}$ the upper bound on the search depth. If $i = k_{\max} - 1$ and there exists a single event that prevents the hazard from happening, the algorithm finds exactly those traces containing this single event and all orderings when processing level $k_{\max}$. If $i = k_{\max} - x$, the same argument applies, and up to $x$ events are found that can prevent the error from happening. □