Towards Symbolic Causality Checking using SAT-Solving

Adrian Beer
University of Konstanz
Department of Computer and Information Science
Chair for Software Engineering

adrian.beer@uni.kn
Joint work with

Uwe Kühne
Airbus Defence & Space
(form. EADS Deutschland)

Florian Leitner-Fischer
Chair for Software Engineering
University of Konstanz

Stefan Leue
Chair for Software Engineering
University of Konstanz
Motivation

♦ Complexity of software and systems is increasing

♦ Classical manual safety analysis is often not feasible for complete systems

♦ We need a technique to identify hazardous events in large systems automatically

♦ Model-based engineering helps to maintain complexity and enables many automatic analysis techniques

› Our Goal:
  Given a system model and a property:
  Compute the events that are causal for a property violation
Outline

1. Introduction to Causality Computation
2. Symbolic Causality Checking
3. Evaluation and Comparison
4. Conclusion and Outlook
Railway Crossing Example

**The Real World:**

Requirement:

*The train and the car are never on the crossing at the same time.*
Railway Crossing Example

- Model of The Real World:

Property (LTL):

$$\Box \neg (\text{train\_crossing} \land \text{car\_crossing})$$
Introduction

Model Checking is the basis for our approach:

- Model of the software (transition system, Kripke structure)
- Model checking algorithm
  - Explicit state space search
  - BDD-based MC
  - Bounded MC (SAT-based)
- Requirement specification (assertions, temporal logic, automata)

\( M \models S \)
Introduction

- Model Checking is the basis for our approach:

  \[ \text{M} \models \text{S} \]

  - model of the software (transition system, Kripke structure)
  - model checking algorithm
  - requirement specification (assertions, temporal logic, automata)

- If model violates the property, one counterexample is generated

- Problem:
  - Only one counterexample is not enough to understand causal relationships
  - Manual analysis of all paths is not feasible
Idea:

- Automatically find event orderings and occurrences that are causal for a property violation.

Automatic causality computation basis:

1. A compact representation of event relationships
2. Formal definition of causality
Introduction

1. **A compact representation of event relationships:**
   - Convert counterexamples into general causal relationships
   - Easy to determine causal relationships between events

**Event Order Logic (EOL):**

- Unordered operators: \( a \land b, a \lor b, \neg a \)

- Ordered operators:
  - \( a \land b \), \( a \) occurs always before \( b \)
  - \( a \land] b \), \( a \) occurs always until \( b \) occurs
  - \( a \land< c \land> b \), between \( a \) and \( b \), \( c \) occurs always

- Example: \( \psi = Gf \land (Ca \land Cc) \)
2. **Formal definition of causality:**
Adaption of the “actual cause” definition by Halpern & Pearl


4 Conditions to define causality:

- **AC1:** At least one counterexample has to lead into a property violation. Otherwise there can not exist a cause either

\[ \psi = Ca \land Cc \land Ta \land Gc \land Tc \]

One representative of the set of counterexamples described by \( \psi \):
AC2.1: If at least one of the causal events does not happen, the effect does not happen either

Example:
AC2.2: If the causal events occur, the occurrence of other events can not prevent the effect

Example:

\[ \psi = Ca \land Cc \land \neg Cl \land Ta \land Gc \land Tc \]

“The non-occurrence of \( Cl \) is also a cause for the property violation”
\textbf{AC3}: no true subset of the causal events satisfies AC1 and AC2 (minimality)

Example:
OC1: The ordering of events can be causal for a property violation

Example:

“if the gate closes before the car is on the crossing, the hazard does not occur.”
Outline

1. Introduction to Causality

2. Symbolic Causality Checking

3. Evaluation and Comparison

4. Conclusion and Outlook
Symbolic Causality Checking

- Until now:
  - Explicit state model checking used for generation of paths
  - Problem:
    - Generation of all possible counterexamples AND good paths is necessary to fulfill causality conditions (AC1-OC1)
    - Time and memory limitations for large systems
      → storage of many long paths inefficient
Symbolic Causality Checking

1. **Idea:**
   - Use Bounded Model Checking (BMC) to generate only necessary counterexamples and paths through the system
   - BMC also takes model of the system and an LTL property as input.
     - Additional: Upper bound $k_{max}$ representing the maximum execution length of the system

1. Unrolling of the transition relation up to $k$: 
Symbolic Causality Checking

1. **Idea:**
   - Use Bounded Model Checking (BMC) to generate only necessary counterexamples and paths through the system
   - BMC also takes model of the system and an LTL property as input.
     - Additional: Upper bound $k_{max}$ representing the maximum execution length of the system
   
2. The LTL property is negated and also projected onto a path of length $k$:

\[ \neg \phi = \lozenge (Tc \land Cc) \]

\[ s_1(Tc \land Cc) \lor s_2(Tc \land Cc) \lor \ldots \]
Symbolic Causality Checking

1. Idea:
   - Use Bounded Model Checking (BMC) to generate only necessary counterexamples and paths through the system
   - BMC also takes model of the system and an LTL property as input.
     - Additional: Upper bound $k_{max}$ representing the maximum execution length of the system

3. Both are inserted into an SAT solver
   → If the generated formula is satisfiable the result represents a counterexample to the property.
Symbolic Causality Checking

2. **Idea:**
   - Iteratively learn new causal relationships between events.
   - Store the event orderings and occurrences in an efficient data structure.
   - Constraining the underlying SAT solver with the already known information.
Symbolic Causality Checking

- EOL Matrix:
  - Efficient storage of event occurrences and orderings

- Example: \( \psi = Ca \land Cc \land Gf \)

\[
\begin{pmatrix}
Ca & Cc & Gf \\
\emptyset & \emptyset & \text{TRUE} \\
\emptyset & \emptyset & \emptyset
\end{pmatrix}
\]
Symbolic Causality Checking

- **EOL Matrix:**
  - Normal form is needed to process formula to matrix conversion efficiently:
  
  - Event Order Normal Form
    - Forbids unordered operators as sub-formulas of ordered EOL formulas

- **Example:** \( \psi = C_a \land (C_c \land G_f) \iff (C_a \land C_c) \land (C_a \land G_f) \land (C_c \land G_f) \)

<table>
<thead>
<tr>
<th>( C_a )</th>
<th>( C_c )</th>
<th>( G_f )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \emptyset )</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>( \emptyset )</td>
<td>TRUE</td>
</tr>
<tr>
<td>( \emptyset )</td>
<td>TRUE</td>
<td>( \emptyset )</td>
</tr>
</tbody>
</table>
Symbolic Causality Checking

- EOL Matrix:
  - Storing of non-occurrences is important as well

- Example: \[ \psi = (T_a \land_\prec \neg Cl \land_\succ T_c) \land (T_a \land_\prec \neg Cl \land_\succ G_c) \]

\[
\begin{pmatrix}
T_a & \emptyset & \neg Cl & \emptyset \\
G_c & \emptyset & \emptyset & \emptyset \\
T_c & \emptyset & \emptyset & \emptyset \\
\end{pmatrix}
\]
Symbolic Causality Checking

- **Iteration schema of the algorithm:**
  1. Generate counterexample
  2. Refine EOL matrices with new information
  3. Combine EOL matrices to generate EOL formula for all found causes
  4. Constrain the SAT Solver

\[
\psi_{M_{E_1}} = (C_a \land C_c) \land \ldots \land (T_a \land T_c)
\]

\[
\psi_{M_{E_1}} \lor \psi_{M_{E_2}} \lor \ldots \lor \psi_{M_{E_n}}
\]
Symbolic Causality Checking

2 passes of the algorithm have to be done:

1. Find event orderings & occurrences:
   - Input parameters:
     - Model of the system \( M \)
     - Current level \( k \)
     - Negated LTL property \( \neg \phi \)
     - Negated EOL formula \( \neg \psi \)

2. Non-occurrences detection:
   - Input parameters:
     - Model of the system \( M \)
     - Current level \( k \)
     - LTL property \( \phi \)
     - EOL formula \( \psi \)

Find events on traces that lead into a property violation.
(Bad paths)

Find events that prevent a property violation.
(Good paths)
Outline

1. Introduction to Causality
2. Symbolic Causality Checking
3. Evaluation and Comparison
4. Conclusion and Outlook
Evaluation of the Case Study

- Comparison of Symbolic Causality Checking to Explicit State Causality Checking:

<table>
<thead>
<tr>
<th></th>
<th>Run time (sec.)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MC</td>
<td>CC1</td>
</tr>
<tr>
<td>Railroad</td>
<td></td>
<td></td>
</tr>
<tr>
<td>explicit</td>
<td>0.01</td>
<td>0.12</td>
</tr>
<tr>
<td>symbolic</td>
<td>0.01</td>
<td>0.16</td>
</tr>
<tr>
<td>Airbag</td>
<td></td>
<td></td>
</tr>
<tr>
<td>explicit</td>
<td>0.96</td>
<td>148.52</td>
</tr>
<tr>
<td>symbolic</td>
<td>0.02</td>
<td>4.81</td>
</tr>
</tbody>
</table>

- Symbolic CauSeMV extension to NuSMV model checker
- Symbolic and Explicit experiments conducted on the same PC
  - Intel Xeon Processor 3.60 Ghz, 144GBs RAM
- In Symbolic case: $k_{max} = 10$ in both experiments
Evaluation of the Case Study

- The found causal relationships are the same in the explicit and the symbolic case

- Problem in Symbolic Causality Checking:
  - Soundness and completeness can only be guaranteed up to $k_{max}$

- However:
  - Number of new found event occurrences and orderings decreases with the search depth $k$
  - Paths with long causal event chains are less important in practice:
    - Many combinations of failures are less likely to happen
Outline

1. Introduction to Causality
2. Symbolic Causality Checking
3. Evaluation and Comparison
4. Conclusion and Outlook
Conclusion and Outlook

Summary:
- Classical manual safety analysis is often not feasible for whole systems
- Symbolic causality checking automatically identifies hazardous events in large systems:
  - Find causal event relationships by iterative learning of hazardous events using Bounded Model Checking
  - Constraining the underlying SAT solver in order to exclude already known information
- Implementation of CauSeMV in NuSMV
Conclusion and Outlook

♦ Outlook

Integration of CauSeMV into automated verification tool chain
  – Automatic generation of NuSMV models from High-level UML/SysML models
  – Automatic generation of Fault Trees, FMEAs, ...
  – QuantUM (Tool DEMO)

Evaluation using large industrial models

– For example: Safety Analysis of an Airport Surveillance Radar

  In explicit case: $> 10^8$ states, $> 10^9$ transitions
**Tool DEMO**

## Automation Result

**Title:** SpinCause result  
**Created on:** Mon Jan 20 14:40:46 CET 2014

**State:** http://open-services.net/ns/auto#complete  
**Verdict:** http://open-services.net/ns/auto#passed

**Reload**  
**Produced by request**  
**Reports on plan**  
**Fault Tree**

<table>
<thead>
<tr>
<th>NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>transitions</td>
<td>26602</td>
</tr>
<tr>
<td>states</td>
<td>6083</td>
</tr>
<tr>
<td>propability</td>
<td>2.1197109658158444E-6</td>
</tr>
<tr>
<td>level</td>
<td>18</td>
</tr>
<tr>
<td>errorCount</td>
<td>1262</td>
</tr>
<tr>
<td>depth</td>
<td>35</td>
</tr>
</tbody>
</table>