SpinCause: A Tool for Causality Checking

Florian Leitner-Fischer

University of Konstanz
Chair for Software and Systems Engineering

florian.leitner@uni-konstanz.de
http://se.uni-konstanz.de/

SPIN 2014

software engineering systems

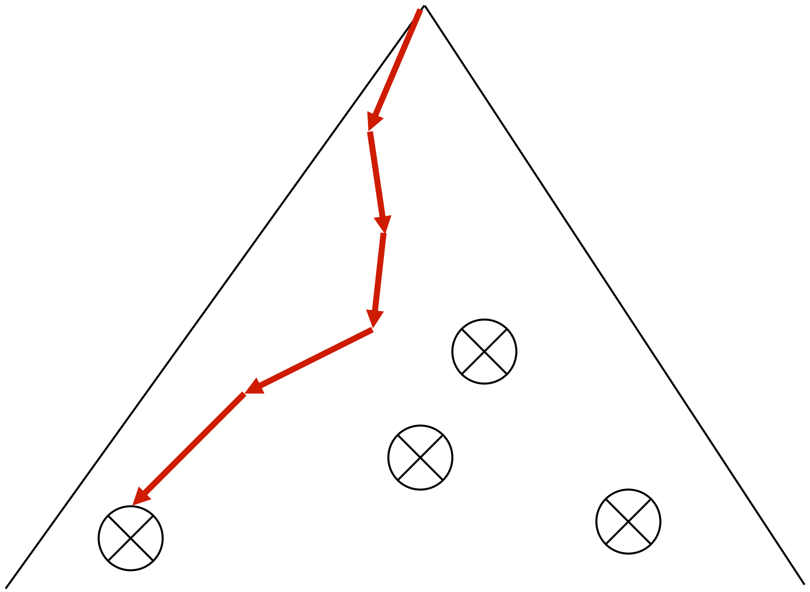
Joint work with

Stefan Leue

Chair for Software and Systems Engineering
Department of Computer and Information Science
University of Konstanz
Germany

Model Checking Result

- the path into a property violating state
  - called an error path or counterexample



Motivation

- **Model Checking**
  - returns counterexamples for property violation
  - but **what is the cause for the property violation?**
- **Manual Counterexample Analysis**
  - tedious
  - error prone
  - essentially impossible for large models
- **Our Solution:**
  - **algorithmic causality computation**

Causality

- **(Naive) Lewis Counterfactual Reasoning**

  c is causal for e (effect / hazard) if, had c not happened, then e would not have happened either

  - logical foundation of some software debugging techniques, e.g.,
    - delta debugging
    - nearest neighbor techniques
  - best suited for single cause failures

Halpern / Pearl Structural Equation Model (SEM)

- **Key Ideas**
  - events are represented by boolean variables
    - specified using structural equations
  - computes minimal boolean disjunction and conjunction of causal events
  - causal dependency of events represented by causal networks
  - reference

    J. Halpern and J. Pearl, "Causes and explanations: A structural-model approach. Part I: Causes," *The British Journal for the Philosophy of Science*, 2005.

Halpern / Pearl Structural Equation Model (SEM)

- **Actual Causality Conditions**
  - AC1: ensures that there exists a trace where the boolean combination of causal events **c** and the effect **e** occur
  - AC2:
    1. if at least one of the causal events does not happen, the effect **e** does not happen
    2. if the causal events occur, the occurrence of other events can not prevent the effect
  - AC3: no subset of the causal events satisfies AC1 and AC2 (minimality)

Causality Checking

- **Causality Checking**
  - algorithmically computes
    - **causal events**
    - causal **event order**
    - causal **non-occurrence** of events
  - for a non-reachability property violation
    - $G \neg$ (unsafe state)

Algorithmics

- **Implementation Variants**
  - qualitative causality checking
    - state-space exploration with DFS / BFS
    - causality computation is made on-the-fly

      Florian Leitner-Fischer and Stefan Leue:
      Causality Checking for Complex System Models,
      In Proceedings of 14th International Conference on Verification, Model Checking,
      and Abstract Interpretation (VMCAI2013), LNCS, Volume 7737, Springer Verlag, 2013.

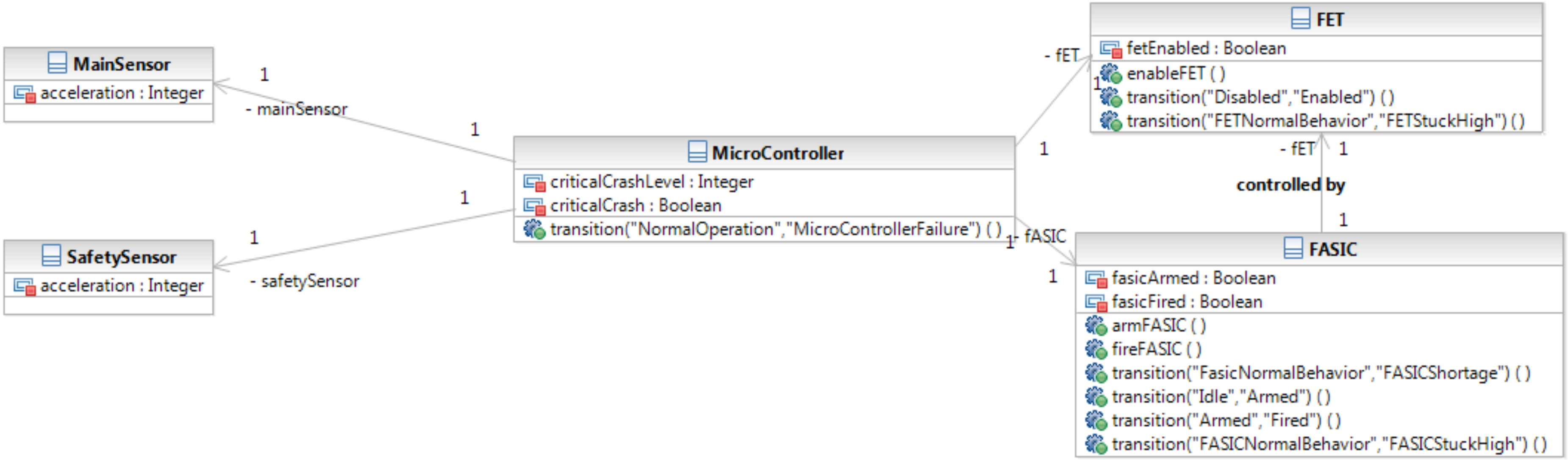
  - optimization
    - iterative approach using BFS / parallel BFS
    - first run:
      - compute causal event combinations
    - second run:
      - compute causal event order and causal non-occurence
  - both implementations based on SpinJa
    - code.google.com/p/spinja/

Algorithmics

- **Implementation Variants**
  - probabilistic causality computation
    - translate PRISM model to Promela
    - qualitative causality checking is used to compute causal events
    - probability of causal events is computed

Florian Leitner-Fischer and Stefan Leue:
On the Synergy of Probabilistic Causality Computation and Causality Checking,
In Proceedings of International SPIN Symposium on Model Checking (SPIN 2014),
LNCS, Volume 7976, pp 246-263, Springer Verlag, 2013.

Airbag Case Study

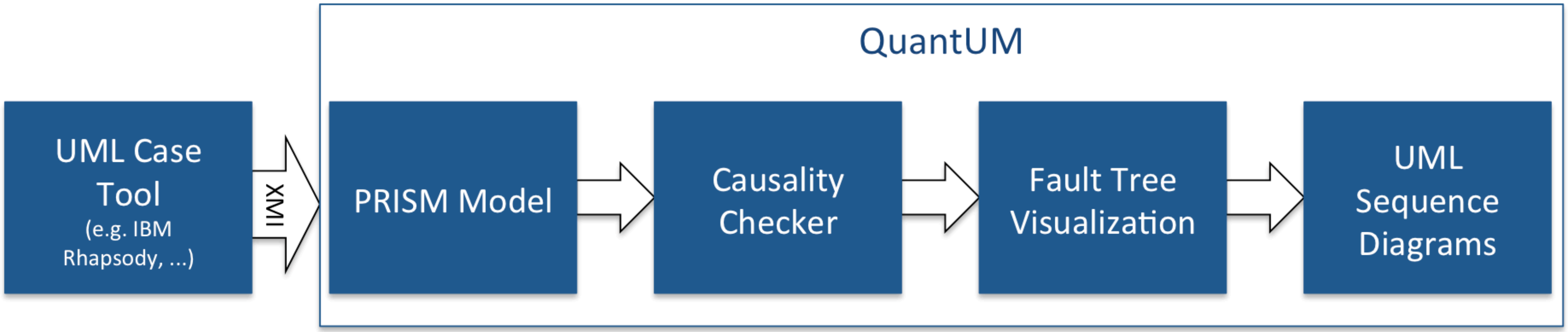


Airbag Case Study - Result

- **Event Order Logic Formulas**
  - Boolean event occurrence conditions: $a \wedge b$, $a \vee b$, $\neg a$
  - Boolean event ordering conditions: $a \mathbin{\dot\wedge} b$
    - a and b occur, and a occurs before b
  - interval operators: $a \mathbin{\dot\wedge}_{[} b$, $a \mathbin{\dot\wedge}_{]} b$, $a \mathbin{\dot\wedge}_{<} b \mathbin{\dot\wedge}_{>} c$
    - a occurs until eventually b will hold in every state
    - a always holds until eventually b occurs
    - in the interval delimited by a and c, b always holds
  - event order logic can be translated to LTL

  - airbag case study: causal event orders
  - $(\mathrm{FASICShortage})$
  - $(\mathrm{FETStuckHigh} \mathbin{\dot\wedge} \mathrm{FASICStuckHigh})$
  - $(\mathrm{MicroControllerFailure} \mathbin{\dot\wedge} \mathrm{enableFET} \mathbin{\dot\wedge} \mathrm{armFASIC} \mathbin{\dot\wedge} \mathrm{fireFASIC})$
  - $(\mathrm{FETStuckHigh} \mathbin{\dot\wedge} \mathrm{MicroControllerFailure} \mathbin{\dot\wedge} \mathrm{armFASIC} \mathbin{\dot\wedge} \mathrm{fireFASIC})$
  - $(\mathrm{MicroControllerFailure} \mathbin{\dot\wedge} \mathrm{enableFET} \mathbin{\dot\wedge} \mathrm{FASICStuckHigh})$

QuantUM: A Tool For Model-Based Functional Safety Analysis



www.quantum-tool.com

Industrial Case Studies

Airbag Control Unit

TRW Automotive

2 952 states in the analysis model.
fault tree computed within 1.24 seconds with 18 MBs memory

Airport Surveillance Radar

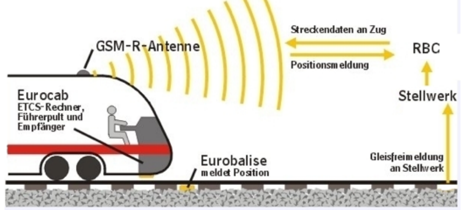
AIRBUS DEFENCE & SPACE

46 million states in the analysis model.
fault tree computed within 22 minutes with 12 GBs memory

Driver Assistance System

German Premium Automotive OEM
36 million states in the analysis model
fault tree computed within 20 minutes with 5 GBs memory

European Train Control System: Train Odometer Controller



unife THE EUROPEAN RAIL INDUSTRY
ERTMS A UNIFE website

11 722 states in the analysis model.
fault tree computed within 1.5 seconds with 19 MBs memory

and many more …

DEMO

Conclusion

- **Summary**
  - causality checking is a technique complementing model checking
    - aim: algorithmic support for the debugging of models
  - supports PRISM and Promela models and non-reachability properties

- **More Info**
  - http://se.uni-konstanz.de/causalitychecking/

- **Future Work**
  - causality checking in a symbolic environment
  - causality checking for liveness properties