Combining Real-Time Model-Checking and Fault Tree Analysis

Andreas Schäfer

University of Oldenburg
Contents

► What is Fault Tree Analysis (FTA)?
► Motivation
► Duration Calculus with Liveness
► Fault Tree Semantics
► Model-Checking Fault Trees
► Fault Tree guided Model-Checking
► Future Work
Fault Tree Analysis

Afternoon-Session exceeds 1.5h

≥ 1

One talk takes more than 30min & Technical Problems

≥ 1

Too many slides & Watch defective & Video projector failure & Notebook failure

Motivation
DCL
Semantics
DCL MC
Model-Checking
FT guided MC
Future Work

Real-Time Model-Checking and Fault Tree Analysis – p.3/17
Fault Tree Analysis

But not based on rigid formal semantics

Afternoon-Session exceeds 1.5h

≥ 1

One talk takes more than 30min

≥ 1

Technical Problems

Too many slides

Watch defective

Video projector failure

Notebook failure

One talk takes more than 30min

Technical Problems

But not based on rigid formal semantics

Video projector failure

Notebook failure
Motivation

- Verifying: Fault Tree is constructed properly, e.g. every cause is covered
- Fault Tree guided Model-Checking
Logic: Duration Calculus with Liveness (DCL) (Skakkebæk, 1994)

Extending DC with expanding modalities $\triangleright, \triangleleft, \Diamond_L, \Box_L$ to express liveness properties.

Example: $\Diamond_L[session] > 90$
DCL Semantics for Fault Trees

Based on a Duration Calculus Semantics (Reif, Schellhorn, Thums, 2000)

- Events formalised by Duration Calculus with Liveness formulae
- For each gate two proof obligations
  - sufficiency (the sub-events always cause the event to happen)
  - necessity (no possible cause is forgotten)
- Fault Tree semantics: Conjunction of all proof obligations.
Gate-types

Additional to normal gate-types

- **AND-Gate** &
- **OR-Gate** ≥ 1
Gate-types

Additional to normal gate-types

- AND-Gate &
- OR-Gate ≥ 1

Gate-types for cause-consequence relationships

- asynchronous OR-gate > 1 CC
- asynchronous AND-gate & CC
- synchronous AND-gate & SCC
Gate-types

Additional to normal gate-types

- **AND-Gate** \(\&\)
- **OR-Gate** \(\geq 1\)

Gate-types for cause-consequence relationships

- asynchronous OR-gate \(\geq 1\) \(\text{CC}\)
- asynchronous AND-gate \(\&\) \(\text{CC}\)
- synchronous AND-gate \(\&\) \(\text{SCC}\)
Semantics for OR-gates

OR-gate: sufficient condition

\[
\left( \bigvee_{i=1}^{n} E_i \right) \Rightarrow E
\]

OR-gate: necessary condition

\[
E \Rightarrow \left( \bigvee_{i=1}^{n} E_i \right)
\]
Semantics for other gates

- **OR-acc-gate necessary condition**
  \[ \neg \left( \bigvee_{i=1}^{n} \diamond E_i \{ \diamond_L / \} ; E \{ \diamond_L / \} \right) \]

- **AND-acc-gate necessary condition**
  \[ \neg \left( \bigwedge_{i=1}^{n} \diamond E_i \{ \diamond_L / \} ; E \{ \diamond_L / \} \right) \]

- **AND-scc-gate: necessary condition**
  \[ \neg \left( \neg \diamond \bigwedge_{i=1}^{n} E_i \{ \diamond_L / \} ; E \{ \diamond_L / \} \right) \]

- ...
DCL Model-Checking

- **DCL is undecidable**
- **Decidable subset: Phase Automata (Tapken, 2001)**
- **Model-Checker MobyDC checks if set of automata has common run.**
- **Not closed under complementation**

---

**Diagram:**

- **start_timer**
- **empty or rupture (0,w]**
- **fill or full (0,60]**
- **tank_timer**
Restriction to three common event types (Gorski, 1994)

Formalisation in DCL:

\[ \Diamond_L (\pi \land b \leq \ell) \]  
\[ \Diamond_L \square_L [\pi] \land \neg \Diamond_L ([\pi]; \neg [\pi]) \]  
\[ \Diamond_L ([\pi_1] \land b_1 \leq \ell \leq e_1; \ldots; [\pi_n] \land b_n \leq \ell \leq e_n) \]

where \( \pi_i \land \pi_j \equiv \text{false for } i \neq j \)

- Effectively construct Phase Automata for these patterns
- Class is closed under complementation
Formal system model in terms of Phase Automata
Formal system model in terms of Phase Automata
Formal system model in terms of Phase Automata

For every event (based on pattern):
  Translation:
    Duration Calculus Formula $\rightarrow$ Phase Automaton
  Construction of the Complement Automaton
Model-Checking Fault Trees

- Formal system model in terms of Phase Automata
- For every event (based on pattern):
  - Translation: Duration Calculus Formula $\rightarrow$ Phase Automaton
  - Construction of the Complement Automaton
Model-Checking Fault Trees

- Formal system model in terms of Phase Automata
- For every event (based on pattern):
  - Translation:
    - Duration Calculus Formula $\rightarrow$ Phase Automaton
  - Construction of the Complement Automaton
- Checking all proof obligations by emptiness tests
Example

- Afternoon-Session exceeds 1.5h
  - ≥ 1
    - One talk takes more than 30min
    - Technical problems
Example

Formalisation of system by set $\mathcal{M}$ of Phase Automata
Example

- Afternoon-Session exceeds 1.5h

  - ≥ 1

  - One talk takes more than 30min
  - Technical problems

- Formalisation of system by set $\mathcal{M}$ of Phase Automata
- Formalisation of events by DCL formulae.
Example

- Formalisation of system by set $\mathcal{M}$ of Phase Automata
- Formalisation of events by DCL formulae.

Afternoon-Session exceeds 1.5h

- One talk takes more than 30min

Technical problems

- Model-Checking

FT guided MC

Future Work

A $\diamond L[session] \geq 90$

B $\diamond L[talk] \geq 30$

C $\diamond L[Tech_{ERR}]$
Example

- Formalisation of system by set $\mathcal{M}$ of Phase Automata
- Formalisation of events by DCL formulae.
- Phase Automata $\mathcal{A}_A$ and $\mathcal{A}_{\neg A}$ for event A
Example

Formalisation of system by set $\mathcal{M}$ of Phase Automata

Formalisation of events by DCL formulae.

Phase Automata $\mathcal{A}_A$ and $\mathcal{A}_{\neg A}$ for event $A$
Example

- Formalisation of system by set $M$ of Phase Automata
- Formalisation of events by DCL formulae.
- Phase Automata $A_A$ and $A_{\neg A}$ for event $A$ and $A_B$, $A_{\neg B}$, $A_C$, $A_{\neg C}$ for events $B$ and $C
Example continued

- **Necessary condition**

\[
M \Rightarrow (A \Rightarrow (B \lor C))
\]

iff

\[
M \parallel A \parallel A \parallel B \parallel C = 0
\]

- **Sufficient condition**

\[
M \Rightarrow (A \Leftarrow (B \lor C))
\]

iff

\[
M \parallel A \parallel (A \parallel B \lor A \parallel C) = 0
\]
Case Study Single-Track-Line-Segment

To verify: No collision of trains on single track

Typical approach:
Checking the whole system with Uppaal
But too complex for Uppaal.
Fault Tree guided Model-Checking

Case Study Single-Track-Line-Segment

To verify: No collision of trains on single track

Typical approach:
Checking the whole system with Uppaal
But too complex for Uppaal.

�

Real-Time Model-Checking and Fault Tree Analysis – p.15/17
To verify: No collision of trains on single track
To verify: No collision of trains on single track

Typical approach:
Checking the whole system with Uppaal (Timed Automata)
But *too complex for Uppaal.* ✗
Fault Tree guided Model-Checking

 Fault Tree Analysis leads to set of basic events
Fault Tree guided Model-Checking

- Fault Tree Analysis leads to set of basic events
- FT-Verification shows: All necessary conditions hold. (≤ 10 sec for each gate)
Fault Tree guided Model-Checking

- Fault Tree Analysis leads to set of basic events
- FT-Verification shows: All necessary conditions hold. (≤ 10 sec for each gate)
- Verifying with Uppaal: *Basic events* can not occur (about 1 h for the hardest one)
Fault Tree guided Model-Checking

- Fault Tree Analysis leads to set of basic events
- FT-Verification shows: All necessary conditions hold. ($\leq$ 10 sec for each gate)
- Verifying with Uppaal: Basic events can not occur (about 1 h for the hardest one)
- $\Rightarrow$ Collision impossible ✓
Future Work

➤ Tool support
➤ Further evaluation:
  ➤ Different patterns are needed?
  ➤ Increased model-checking performance?
➤ Fault Tree completion from counter examples
➤ Considering probabilities