MOVEP 2012 Tutorial
Safety, Dependability and Performance Analysis of Extended AADL Models

Part 4: Safety and Dependability Analysis

European Space Agency
European Space Research and Technology Centre

RWTH Aachen University
Software Modeling and Verification Group
Thomas Noll

Fondazione Bruno Kessler
Centre for Scientific and Technological Research
Alessandro Cimatti

MOVEP 2012 School; December 7, 2012; Marseille, France

Contents of Overview

Outline

Safety Assessment

Objectives

- Analyse system behaviour under all possible operational conditions, in particular in presence of malfunctions of its components
- Determine the conditions under which safety hazards can occur
- Ensure that a system meets the safety requirements that are required for its deployment and use

Safety Assessment

Objectives

- Analyse system behaviour under all possible operational conditions, in particular in presence of malfunctions of its components
- Determine the conditions under which safety hazards can occur
- Ensure that a system meets the safety requirements that are required for its deployment and use

Requirements

- Particularly important for safety-critical systems, where unexpected behavior may cause significant loss of money or human lives!
- Carried out in parallel with system design
- Typically needed for certification of safety-critical systems

Safety Assessment

Properties of interest - some examples (qualitative):

- "If no more than 3 components fail, then I never have a total loss of hydraulic power"
- "No single point of failure can cause unavailability of both the primary and secondary power systems"
- "Find all combinations of basic faults which may cause total loss of hydraulic power"

Safety Assessment

Properties of interest - some examples (qualitative):

- “If no more than 3 components fail, then I never have a total loss of hydraulic power”
- “No single point of failure can cause unavailability of both the primary and secondary power systems”
- “Find all combinations of basic faults which may cause total loss of hydraulic power”

Properties of interest - some examples (quantitative):

- “The probability of a total loss of hydraulic power is less than 10^{-7}”
- “The probability that both the primary and secondary power systems fail during the same mission is less than 10^{-9}”

Safety Assessment

Safety Assessment Techniques

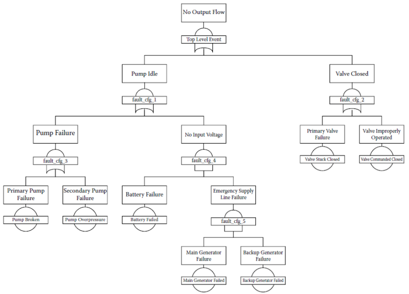
- Several safety assessment techniques, e,g.:
  - Fault Tree Analysis (FTA)
  - Failure Mode and Effects Analysis (FMEA)

Safety Assessment

Safety Assessment Techniques

- Several safety assessment techniques, e,g.:
  - Fault Tree Analysis (FTA)
  - Failure Mode and Effects Analysis (FMEA)

Fault Tree



FMEA Table

Ref. No.	*Item*	*Failure Mode*	*Failure Cause*	*Local Effects*	*System Effects*	*Detection Means*	*Severity*	*Corrective Actions*
1	Pump	Fails to operate	Comp. broken No input flow	Coolant temperature increases	Reactor temperature increases	Temperature alarm	Major	Start secondary pump Switch to secondary circuit
2	Valve	Stuck closed	Comp. broken	Excess liquid	Reactor pressure increases	Coolant level sensor	Critical	Open release valve
3		Stuck open	Comp. broken	Insufficient liquid	Reactor temperature increases	Coolant level sensor	Critical	Open tank valve

References

- Safety Critical Systems (Storey, Addison-Wesley 1996)
- System Safety (Leveson, Addison-Wesley 1995)
- Formal Safety Assessment (Bozzano, Villafiorita, Taylor & Francis 2010)

Outline

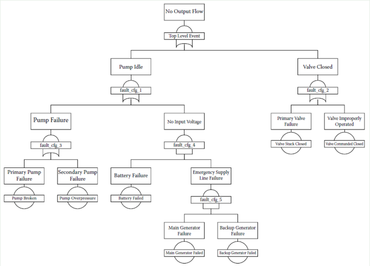
1. Introduction
2. Fault Tree Analysis
3. Failure Mode and Effects Analysis
4. Ongoing Activities
5. Tool Support

Fault Tree Analysis (FTA)

Main Features

- Deductive technique (top-down)
- Graphical representation of the effects of faults on system requirements (using Boolean gates)
- Widespread use in aerospace, avionics, and other domains
- Qualitative model that can be evaluated quantitatively

Fault Tree



Fault Tree Analysis (FTA)

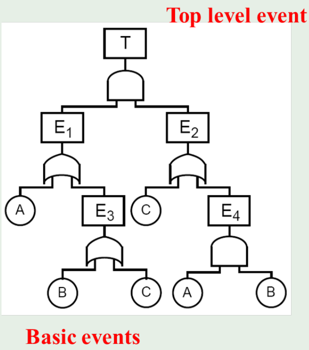
FTA requires:

- Specifying a Top Level Event (TLE) representing an undesired condition
- Find all possible chains of basic events that may cause the TLE to occur

A Fault Tree:

- Is a systematic representation of such chains of events
- Uses logical gates to represent the interrelationships between events and TLE, e.g. AND, OR

Fault Tree

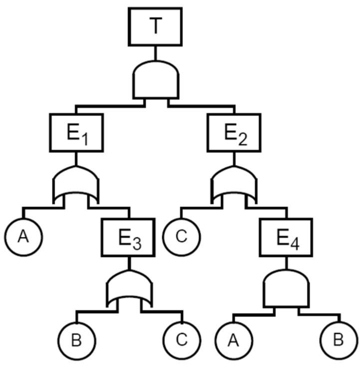


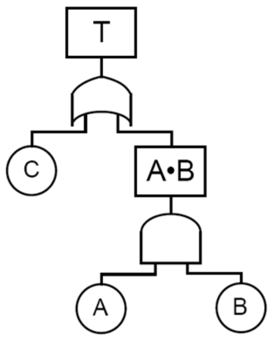
Fault Tree Analysis (FTA)

Logical formula associated to a FT

The FTs below have the same associated logical formula:
$(A \vee (B \vee C) \wedge (C \vee (A \wedge B)) \equiv (C \vee (A \wedge B))$

Logically Equivalent Fault Trees



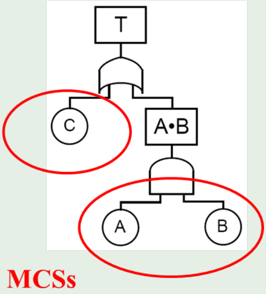


Fault Tree Analysis (FTA)

Minimal Cut Sets (MCSs)

- This shape is of particular interest: representation in terms of Minimal Cut Sets (MCSs)
- Minimal cut set = "smallest set of basic events which, conjoined, cause the top level event to occur"
- Logically: Disjunctive Normal Form (DNF) = disjunction of conjunctions of basic events
- The fault tree on the right has two MCSs: C (single point of failure) and $A \wedge B$ (cut set of order 2)

MCSs



Cut Sets

Fault Configuration

$\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$. A fault configuration FC is a subset of failure mode variables, that is, $FC \subseteq \mathcal{F}$

Cut Sets

Fault Configuration

$\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$. A fault configuration FC is a subset of failure mode variables, that is, $FC \subseteq \mathcal{F}$

Cut Set

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $FC \subseteq \mathcal{F}$ be a fault configuration, and $TLE \in \mathcal{P}$. We say that FC is a cut set of TLE, written $cs(FC, TLE)$ if there exists a trace $s_0, s_1, \ldots, s_k$ for $\mathcal{M}$ such that:

- $s_k \models TLE$
- $\forall f \in \mathcal{F}\ f \in FC \iff \exists i \in \{0, \ldots, k\}\ (s_i \models f)$

Minimal Cut Sets

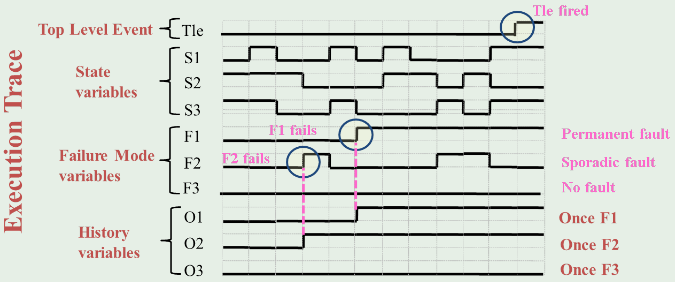
Minimal Cut Sets

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $F = 2^{\mathcal{F}}$ be the set of all fault configurations, and $TLE \in \mathcal{P}$.

The set of minimal cut sets of TLE is the set of cut sets of TLE that are minimal wrt set inclusion. Formally:

- $CS(TLE) = \{FC \in F \mid cs(FC, TLE)\}$
- $MCS(TLE) = \{cs \in CS(TLE) \mid \forall cs' \in CS(TLE)\ (cs' \subseteq cs \rightarrow cs' = cs)\}$

Cut Sets



- History variables remember past failure events
- O_i is true if and only if F_i is true at some point in the past:

$$\mathcal{R}^o = \begin{cases} O_i \rightarrow next(O_i) \\ \neg O_i \rightarrow (next(O_i) \leftrightarrow next(F_i)) \end{cases}$$

- $F_1 \wedge F_2$ is a cut set

Algorithms for FTA

Symbolic Algorithms for FTA

Several algoritmhs:

- BDD-based algorithms
  - Forward algorithm
  - Backward algorithm
- SAT-based algorithms

Algorithms for FTA

Symbolic Algorithms for FTA

Several algoritmhs:

- BDD-based algorithms
  - Forward algorithm
  - Backward algorithm
- SAT-based algorithms

Algorithms Optimizations

- Dynamic Pruning
- Backward algorithm with DCOI (Dynamic Cone of Influence)

Algorithms for FTA

Symbolic Algorithms for FTA

Several algoritmhs:

- BDD-based algorithms
  - Forward algorithm
  - Backward algorithm
- SAT-based algorithms

Algorithms Optimizations

- Dynamic Pruning
- Backward algorithm with DCOI (Dynamic Cone of Influence)

An Example

- BDD-based forward algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

FTA-Forward

FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;

FTA-Forward

FTA: BDD-based Forward Algorithm

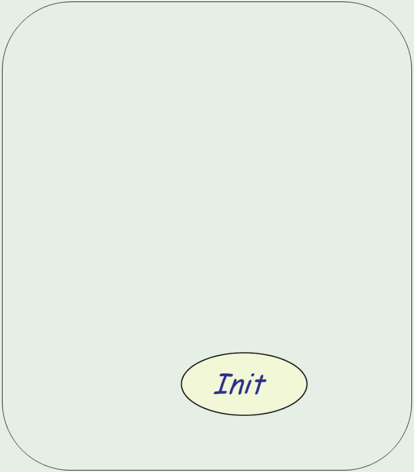
function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;

$\mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;

$\mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;

FTA-Forward

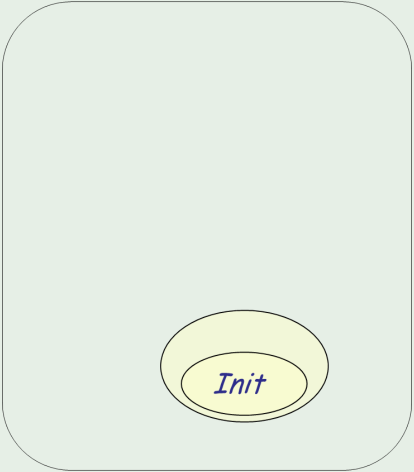


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
  $temp := Reach$;
  $Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
  $Front := Reach \setminus temp$;
end while;

FTA-Forward

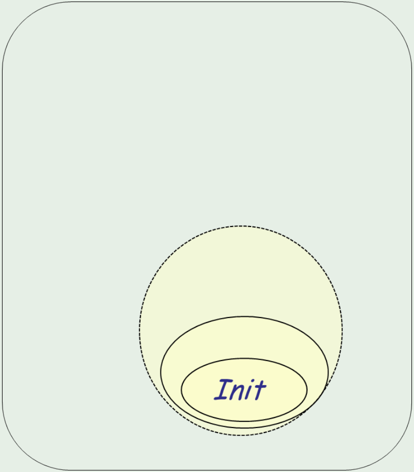


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, TIe)

```
M := Extend(M, R°);
Reach := I ∩ (o = f);
Front := I ∩ (o = f);
while (Front ≠ ∅) do
    temp := Reach;
    Reach := Reach ∪
             fwd_img(M, Front);
    Front := Reach \ temp;
end while;
```

FTA-Forward



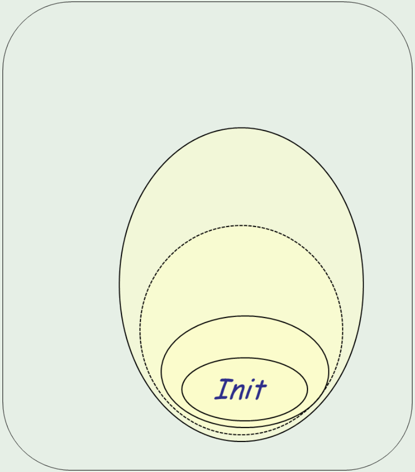
FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

```
    M := Extend(M, R°);
    Reach := I ∩ (o = f);
    Front := I ∩ (o = f);
    while (Front ≠ ∅) do
        temp := Reach;
        Reach := Reach ∪
                 fwd_img(M, Front);
        Front := Reach \ temp;
    end while;
```

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
$temp := Reach$;
$Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
$Front := Reach \setminus temp$;
end while;

FTA-Forward

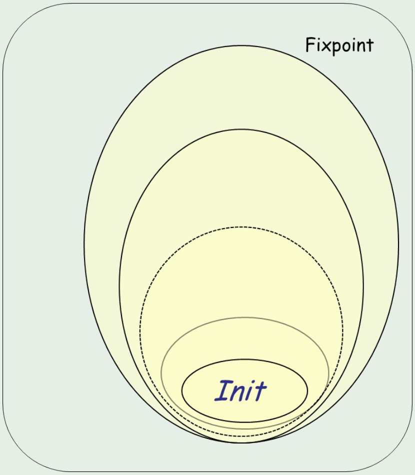


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
  $temp := Reach$;
  $Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
  $Front := Reach \setminus temp$;
end while;

FTA-Forward

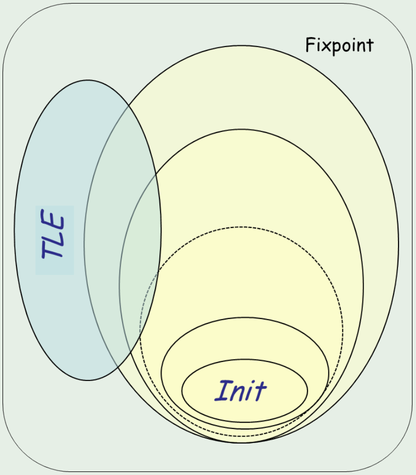


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
  $temp := Reach$;
  $Reach := Reach \cup$
    $fwd\_img(\mathcal{M}, Front)$;
  $Front := Reach \setminus temp$;
end while;
$CS := Project(\underline{o}, Reach \cap Tle)$;

FTA-Forward

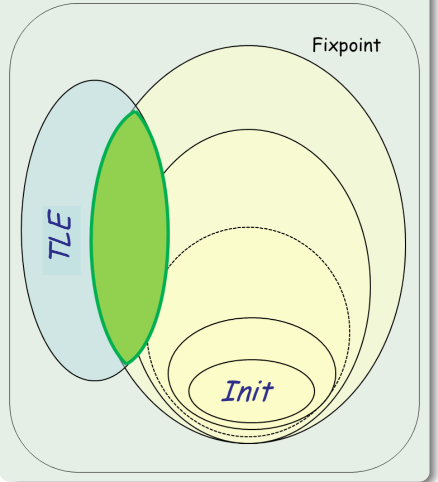


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, Tle)

$\mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
$\mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$\mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while ($\mathit{Front} \neq \emptyset$) **do**
  $\mathit{temp} := \mathit{Reach}$;
  $\mathit{Reach} := \mathit{Reach} \cup$
    $\mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
  $\mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
end while;
$\mathit{CS} := \mathit{Project}(\underline{o}, \mathit{Reach} \cap \mathit{Tle})$;

FTA-Forward

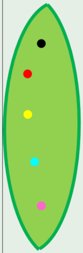


FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while ($Front \neq \emptyset$) **do**
    $temp := Reach$;
    $Reach := Reach \cup$
        $fwd\_img(\mathcal{M}, Front)$;
    $Front := Reach \setminus temp$;
end while;
$CS := Project(\underline{o}, Reach \cap Tle)$;

FTA-Forward



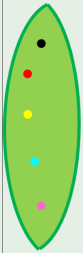
	CS1	CS2	CS3	CS4	CS5
S1	0	1	1	0	1
S2	1	1	0	1	1
S3	1	0	1	0	1
S4	1	0	1	1	1
S5	0	1	1	0	0
F1	0	1	1	1	0
F2	0	0	1	0	1
F3	0	0	1	0	0
O1	0	1	1	1	1
O2	1	1	1	0	1
O3	1	0	1	0	1

FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

  $\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
  $Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
  $Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
  while $(Front \neq \emptyset)$ **do**
    $temp := Reach$;
    $Reach := Reach \cup$
      $fwd\_img(\mathcal{M}, Front)$;
    $Front := Reach \setminus temp$;
  end while;
  $CS := Project(\underline{o}, Reach \cap Tle)$;

FTA-Forward



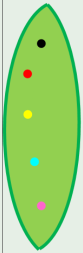
	CS1	CS2	CS3	CS4	CS5
S1					
S2					
S3					
S4					
S5					
F1					
F2					
F3					
O1	0	1	1	1	1
O2	1	1	1	0	1
O3	1	0	1	0	1

FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
  $temp := Reach$;
  $Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
  $Front := Reach \setminus temp$;
end while;
$CS := Project(\underline{o}, Reach \cap Tle)$;
$MCS := Minimize(CS)$;

FTA-Forward

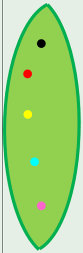


	CS1	CS2	CS3	CS4	CS5
S1					
S2					
S3					
S4					
S5					
F1					
F2					
F3					
O1	0	1	1	1	1
O2	1	1	1	0	1
O3	1	0	1	0	1

function FTA-Forward ($\mathcal{M}$, *Tle*)

- $\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
- $Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
- $Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
- **while** $(Front \neq \emptyset)$ **do**
  - $temp := Reach$;
  - $Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
  - $Front := Reach \setminus temp$;
- **end while**;
- $CS := Project(\underline{o}, Reach \cap Tle)$;
- $MCS := Minimize(CS)$;

FTA-Forward

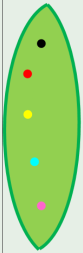


	CS1	CS2	CS3	CS4	CS5
S1					
S2					
S3					
S4					
S5					
F1					
F2					
F3					
O1	0			1	
O2	1			0	
O3	1			0	

function FTA-Forward ($\mathcal{M}$, Tle)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
$\quad temp := Reach$;
$\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
$\quad Front := Reach \setminus temp$;
end while;
$CS := Project(\underline{o}, Reach \cap Tle)$;
$MCS := Minimize(CS)$;
return $Map_{\underline{o} \to \underline{f}}(MCS)$;

FTA-Forward



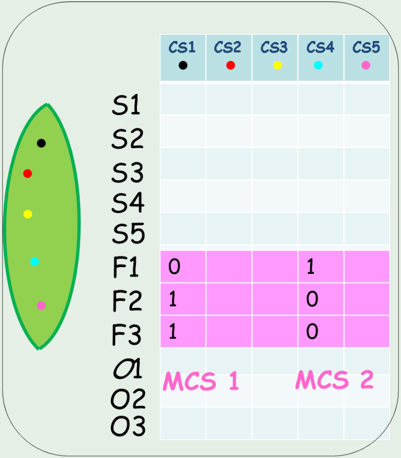
	CS1	CS2	CS3	CS4	CS5
S1					
S2					
S3					
S4					
S5					
F1					
F2					
F3					
O1	0			1	
O2	1			0	
O3	1			0	

FTA: BDD-based Forward Algorithm

function FTA-Forward ($\mathcal{M}$, *Tle*)

$\mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
$Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
$Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
while $(Front \neq \emptyset)$ **do**
    $temp := Reach$;
    $Reach := Reach \cup$
        $fwd\_img(\mathcal{M}, Front)$;
    $Front := Reach \setminus temp$;
end while;
$CS := Project(\underline{o}, Reach \cap Tle)$;
$MCS := Minimize(CS)$;
return $Map_{\underline{o} \rightarrow \underline{f}}(MCS)$;

FTA-Forward

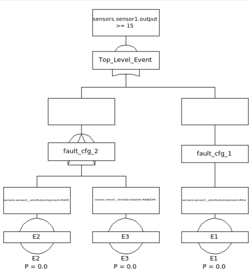


Fault Tree Analysis (FTA)

Dynamic FTs

- Dynamic FTs extend FTs by considering dynamic aspects, such as: ordering constraints, functional dependencies, spares
- Dynamic FTs in COMPASS:
  - Ordering constraints between basic events can be analyzed
  - Priority AND gate (PAND) to display order

Dynamic Fault Tree



References

- FTA (Fault Tree Handbook, U.S. Nuclear Regulatory Commission, 1981)
- FTA (Fault Tree Handbook, NASA 2002)
- Formal FTA (Bozzano, Villafiorita, Taylor & Francis 2010)
- Algorithms for FTA (Bozzano et. al, ATVA 2007)

Outline

Failure Mode and Effects Analysis (FMEA)

Main Features

- Inductive technique (bottom-up)
- Tabled representation of the effects of faults on a set of system properties
- Widespread use in aerospace, avionics, and other domains

FMEA Table

Ref. No.	*Item*	*Failure Mode*	*Failure Cause*	*Local Effects*	*System Effects*	*Detection Means*	*Severity*	*Corrective Actions*
1	Pump	Fails to operate	Comp. broken	Coolant temperature increases	Reactor temperature increases	Temperature alarm	Major	Start secondary pump
			No input flow					Switch to secondary circuit
2	Valve	Stuck closed	Comp. broken	Excess liquid	Reactor pressure increases	Coolant level sensor	Critical	Open release valve
3		Stuck open	Comp. broken	Insufficient liquid	Reactor temperature increases	Coolant level sensor	Critical	Open tank valve

Failure Mode and Effects Analysis (FMEA)

FMEA Table

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $FC_j \subseteq \mathcal{F}$ for $j = 1, \ldots, n$ be a set of fault configurations, and $E_l \in \mathcal{P}$ for $l = 1, \ldots, m$. An FMEA table for $\mathcal{M}$ is the set of pairs $\{(FC_j, E_l) \mid cs(FC_j, E_l)\}$.

Failure Mode and Effects Analysis (FMEA)

FMEA Table

Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L} \rangle$ be a Kripke structure with a set of failure mode variables $\mathcal{F} \subseteq \mathcal{P}$, let $FC_j \subseteq \mathcal{F}$ for $j = 1, \ldots, n$ be a set of fault configurations, and $E_l \in \mathcal{P}$ for $l = 1, \ldots, m$. An FMEA table for $\mathcal{M}$ is the set of pairs $\{(FC_j, E_l) \mid cs(FC_j, E_l)\}$.

Cardinality of FMEA Tables

- FMEA table of cardinality k includes fault configurations of cardinality up to k

Compaction of FMEA Tables

FMEA tables may be “redundant”

Compaction of FMEA tables improves readability

- Idea: remove entries with cardinality k that are “subsumed” by other entries of cardinality less than k

Compaction of FMEA Tables

FMEA tables may be "redundant"

Compaction of FMEA tables improves readability

- Idea: remove entries with cardinality k that are "subsumed" by other entries of cardinality less than k

An Example

- Set of faults: $\{F_1, F_2, F_3, F_4, F_5\}$
- Set of events: $\{E\}$

Compaction of FMEA Tables

FMEA tables may be "redundant"

Compaction of FMEA tables improves readability

- Idea: remove entries with cardinality k that are "subsumed" by other entries of cardinality less than k

An Example

- Set of faults: $\{F_1, F_2, F_3, F_4, F_5\}$
- Set of events: $\{E\}$

An Example (ctd)

FMEA Table of Cardinality 2:

- Fault Configurations of order 1: $\{F_i\}$ for all $i = 1, \ldots, 5$
- Fault Configurations of order 2: $\{F_i, F_j\}$ for all $i, j = 1, \ldots, 5$ with $(i \neq j)$

Compaction of FMEA Tables

An Example (ctd)

Suppose that:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$ are in FMEA table T

Compaction of FMEA Tables

An Example (ctd)

Suppose that:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$ are in FMEA table T

An Example (ctd)

Typically T contains also:

- $(\{F_1, F_i\}, E)$ for $i = 2, 3, 4, 5$
- $(\{F_2, F_i\}, E)$ for $i = 3, 4, 5$
- $(\{F_3, F_i\}, E)$ for $i = 4, 5$

Compaction of FMEA Tables

An Example (ctd)

Suppose that:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$ are in FMEA table T

An Example (ctd)

Typically T contains also:

- $(\{F_1, F_i\}, E)$ for $i = 2, 3, 4, 5$
- $(\{F_2, F_i\}, E)$ for $i = 3, 4, 5$
- $(\{F_3, F_i\}, E)$ for $i = 4, 5$

An Example (ctd)

Also suppose that:

- $(\{F_4, F_5\}, E)$ is in T

Compaction of FMEA Tables

An Example (ctd)

Complete FMEA Table:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$
- $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_1, F_4\}, E)$, $(\{F_1, F_5\}, E)$, $(\{F_2, F_3\}, E)$, $(\{F_2, F_4\}, E)$, $(\{F_2, F_5\}, E)$, $(\{F_3, F_4\}, E)$, $(\{F_3, F_5\}, E)$, $(\{F_4, F_5\}, E)$

Compaction of FMEA Tables

An Example (ctd)

Complete FMEA Table:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$
- $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_1, F_4\}, E)$, $(\{F_1, F_5\}, E)$, $(\{F_2, F_3\}, E)$, $(\{F_2, F_4\}, E)$, $(\{F_2, F_5\}, E)$, $(\{F_3, F_4\}, E)$, $(\{F_3, F_5\}, E)$, $(\{F_4, F_5\}, E)$

An Example (ctd)

We want to preserve only:

- those pairs such that single faults have an effect on event E: $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_2, F_3\}, E)$
  - Intuition: e.g. $(\{F_1, F_4\}, E)$ is redundant, because F_4 has no effect on E (E is explained by F_1 alone)
- "genuine" pairs (no subset of faults in T): $(\{F_4, F_5\}, E)$

Compaction of FMEA Tables

An Example (ctd)

Compact FMEA Table:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$
- $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_2, F_3\}, E)$
- $(\{F_4, F_5\}, E)$

Compaction of FMEA Tables

An Example (ctd)

Compact FMEA Table:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$
- $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_2, F_3\}, E)$
- $(\{F_4, F_5\}, E)$

An Example (ctd)

- 6 entries out of 13 have been removed

Compaction of FMEA Tables

An Example (ctd)

Compact FMEA Table:

- $(\{F_1\}, E)$, $(\{F_2\}, E)$, $(\{F_3\}, E)$
- $(\{F_1, F_2\}, E)$, $(\{F_1, F_3\}, E)$, $(\{F_2, F_3\}, E)$
- $(\{F_4, F_5\}, E)$

An Example (ctd)

- 6 entries out of 13 have been removed

An Example (ctd)

- This idea can be generalized to FMEA tables of arbitrary cardinality and arbitrary number of events:
  - Definition is by induction on the cardinality of the table
  - Compact FMEA tables are defined independently for each event E_j

References

- FMEA (Fault Tree Handbook, U.S. Nuclear Regulatory Commission, 1981)
- FMEA (Fault Tree Handbook with Aerospace Applications, NASA 2002)
- Formal FMEA (Bozzano, Villafiorita, Taylor & Francis 2010)

Outline

Ongoing Activities

Compositional FTA

- Build system-level FT from FTs of sub-components
- Reduce workload in FT generation
- Fits into contract-based system development and verification

Ongoing Activities

Compositional FTA

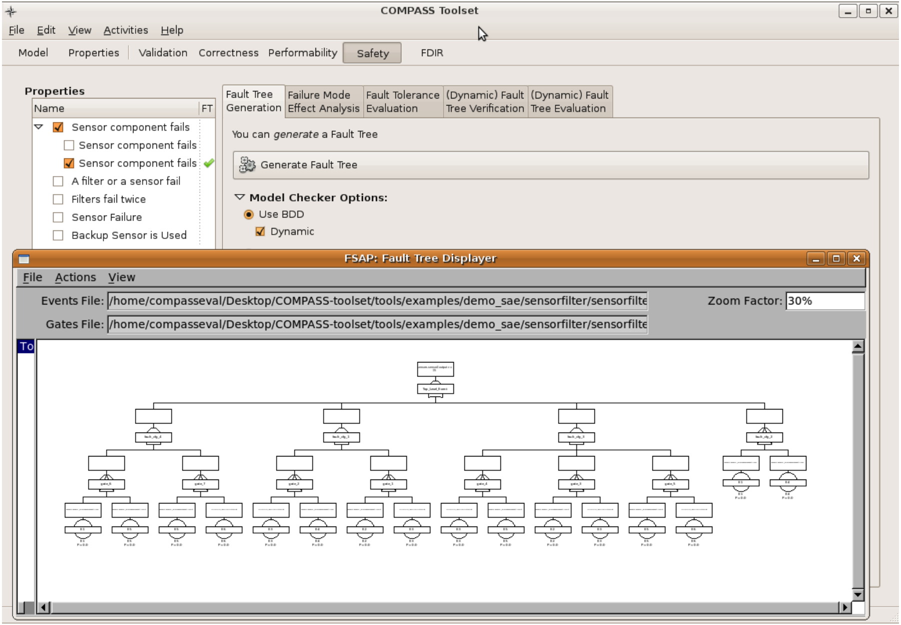
- Build system-level FT from FTs of sub-components
- Reduce workload in FT generation
- Fits into contract-based system development and verification

Hierarchical FTs

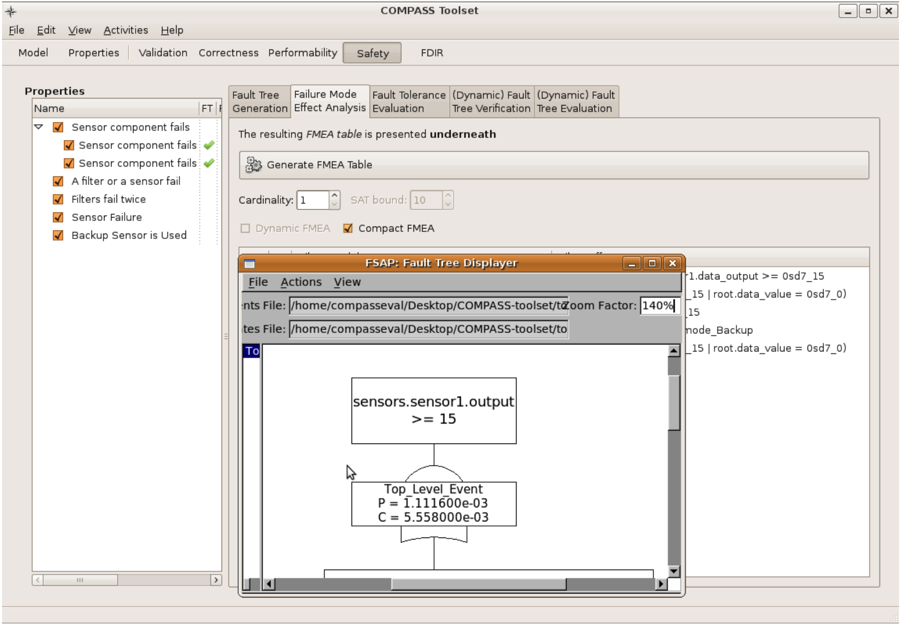
- Generate multi-level FTs
- Improve readability and avoid MCSs enumeration
- FT structure based upon system structure
- Can be integrated with compositional generation of FTs

Outline

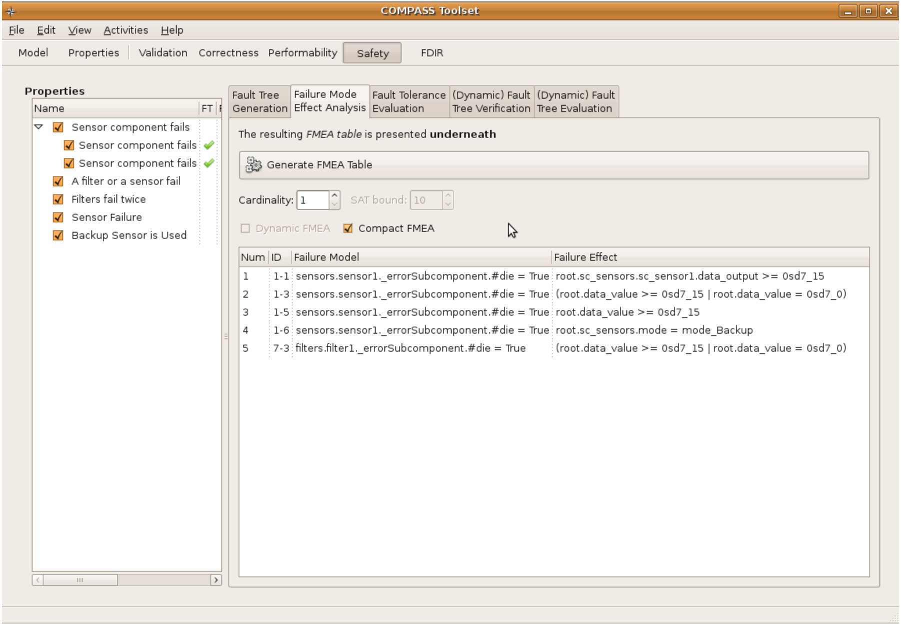
Fault Tree Analysis



Probabilistic Risk Assessment Tree Analysis



Failure Modes and Effects Analysis



Num	ID	Failure Model	Failure Effect
1	1-1	sensors.sensor1._errorSubcomponent.#die = True	root.sc_sensors.sc_sensor1.data_output >= 0sd7_15
2	1-3	sensors.sensor1._errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 | root.data_value = 0sd7_0)
3	1-5	sensors.sensor1._errorSubcomponent.#die = True	root.data_value >= 0sd7_15
4	1-6	sensors.sensor1._errorSubcomponent.#die = True	root.sc_sensors.mode = mode_Backup
5	7-3	filters.filter1._errorSubcomponent.#die = True	(root.data_value >= 0sd7_15 | root.data_value = 0sd7_0)