Safety Assessment in COMPASS

Fondazione Bruno Kessler, Trento, Italy
Marco Bozzano

Future of COMPASS; October 22, 2015; ESA-ESTEC, Noordwijk

Outline

Outline

Overview

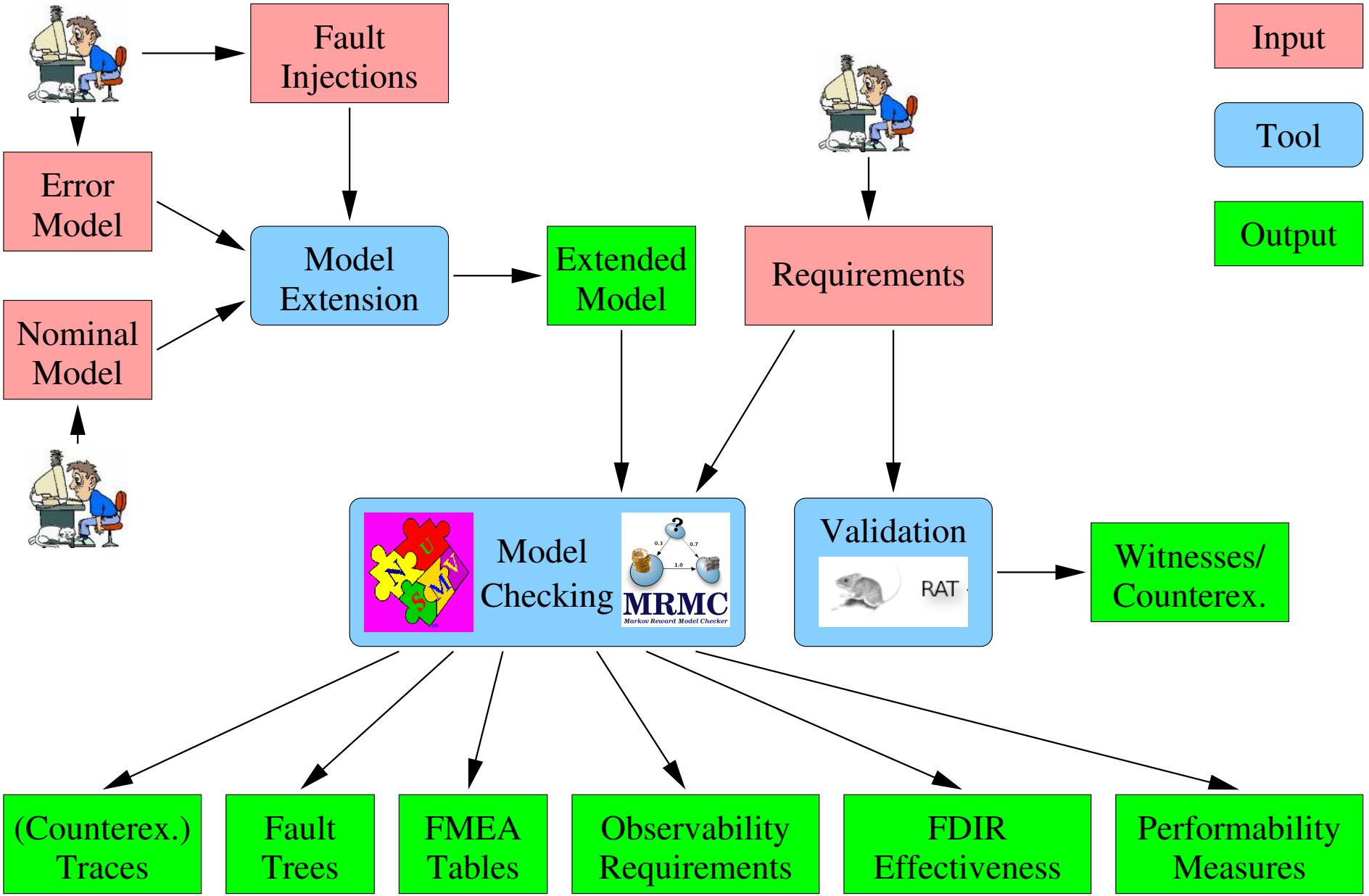
Model-Based Design and Safety Assessment

- Formal model of the system
- Decoupling modeling of *nominal* and *faulty* behavior (automatic model extension facilities)
- Formalized requirements
- Formal, systematic, tool-supported analyses encompassing functional verification, safety assessment, FDIR effectiveness, performance, etc.

Advantages

- Increase comprehension of the system under investigation
- Sharing of models between different stakeholders
- Ensure completeness and consistency of the analyses
- Automate tedious and error-prone tasks
- Support certification

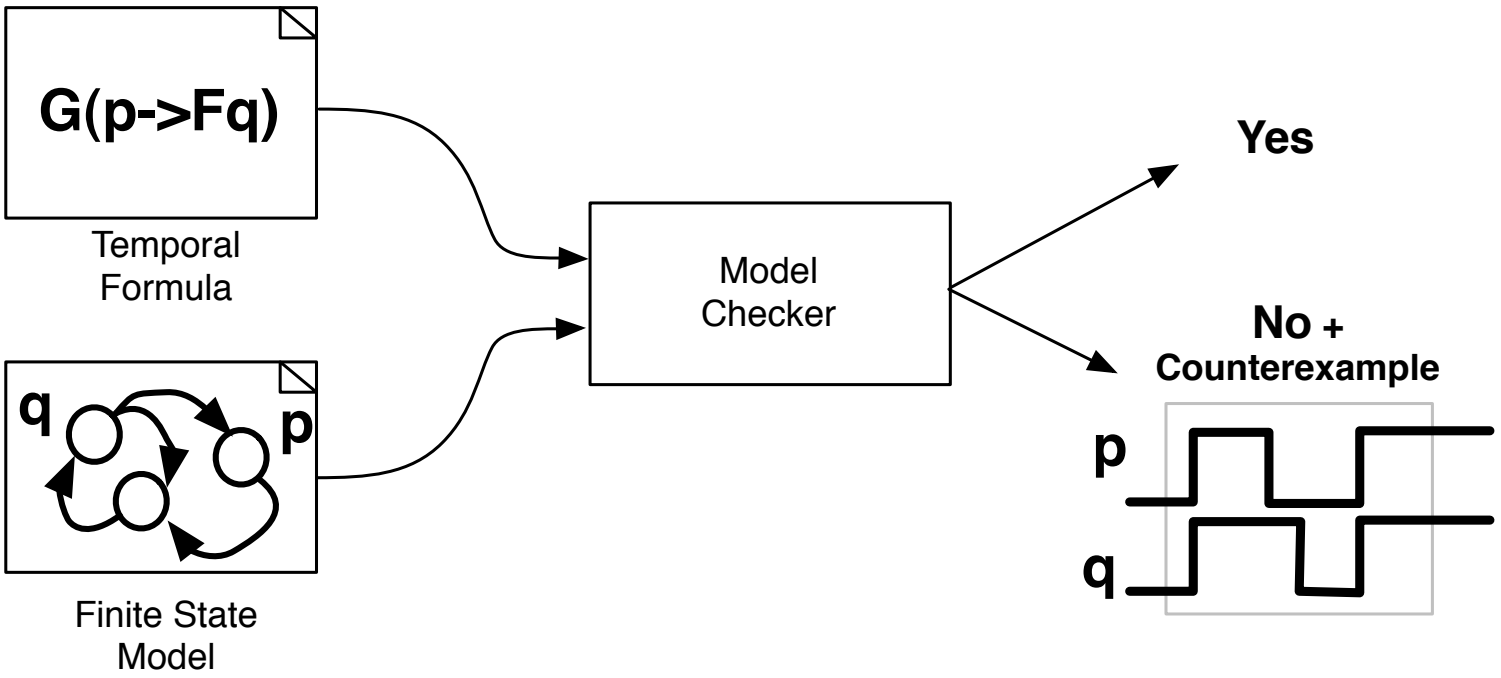
COMPASS Methodology



COMPASS Technology: Model Checking

Model Checker

A pictorial view of a model checker



Safety Assessment

COMPASS builds upon model checking technology to generate safety artifacts

Safety Assessment

Objectives

- Analyse system behaviour in presence of malfunctions
- Determine the conditions under which safety hazards can occur
- Ensure that a system meets safety requirements

Some example properties:

- "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"
- "The probability of a total loss of hydraulic power is less than 10^{-7}"

Patterns

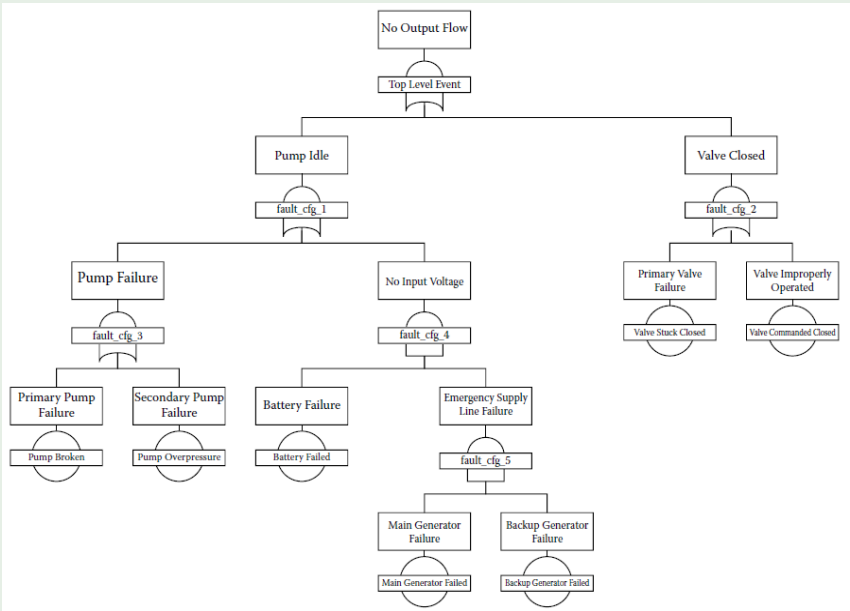
- COMPASS provides patterns to specify properties – they are automatically translated into temporal properties

Safety Assessment in COMPASS

Safety Assessment Techniques

- COMPASS supports safety assessment techniques such as:
  - 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

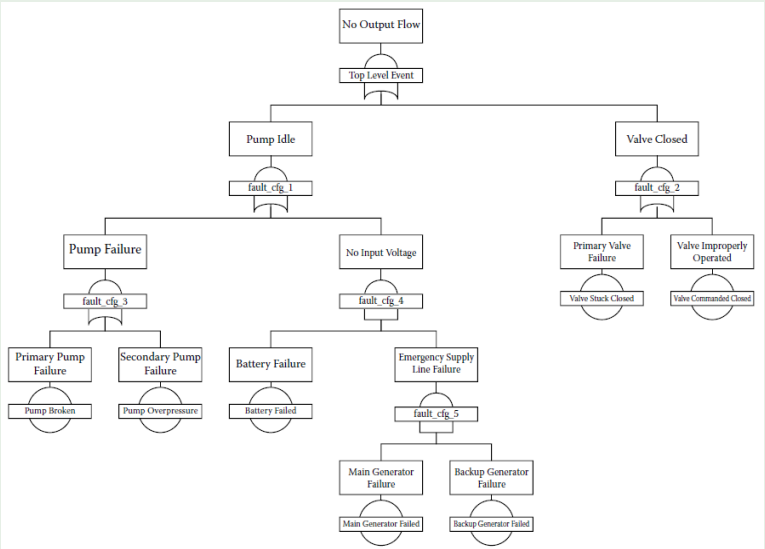
Outline

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)

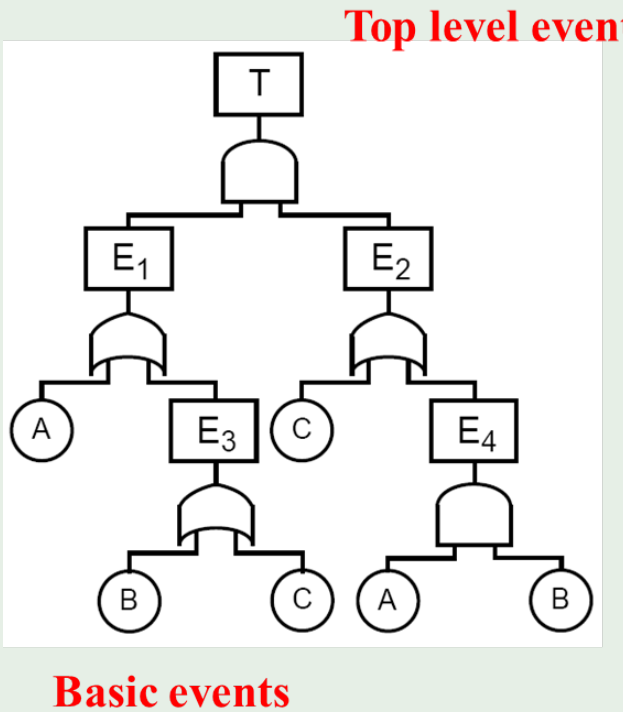
FTA requires:

- Specifying a Top Level Event (TLE) representing an undesired condition
- Find all possible chains of basic events (faults) 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

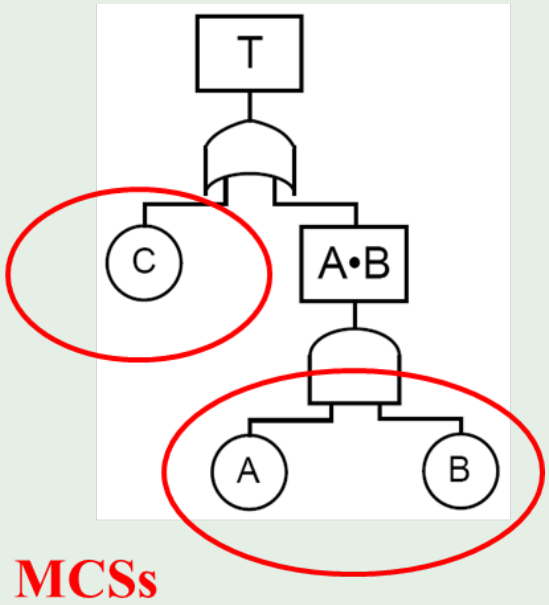


Fault Tree Analysis (FTA)

Minimal Cut Sets (MCSs)

- FT shape 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



Algorithms for FTA

Symbolic Algorithms for FTA

Several algoritmhs:

- BDD-based algorithms
  - Forward algorithm
  - Backward algorithm
- SAT-based algorithms
- . . . SMT- and IC3-based ones (ongoing work)

Several Algorithmic Optimizations

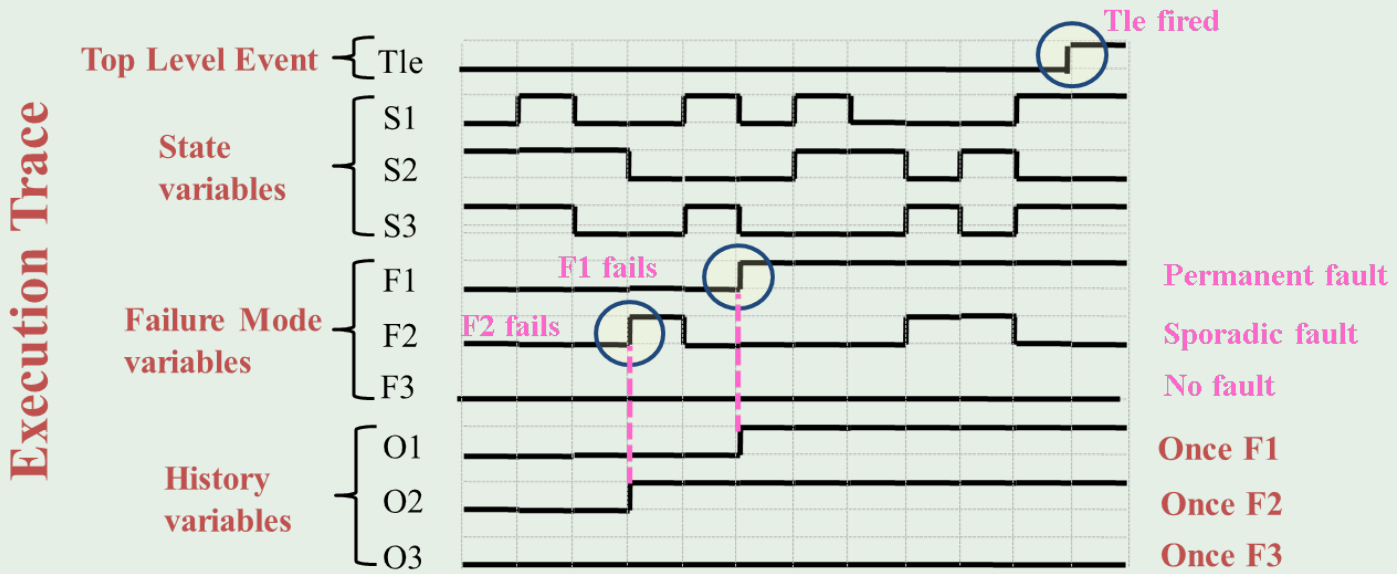
- Dynamic Pruning
- Backward algorithm with DCOI (Dynamic Cone of Influence)
- . . .

An Example

- BDD-based forward algorithm

FTA: BDD-based Forward Algorithm

Trace-based Definition of Cut Set



- $F_1 \wedge F_2$ is a cut set
- History variables remember past failure events:
  O_i is true if and only if F_i is true at some point in the past:

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

FTA: BDD-based Forward Algorithm

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

1
2
3
4
5
6

7
8
9
10
11

FTA-Forward

FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2
3
4
5
6

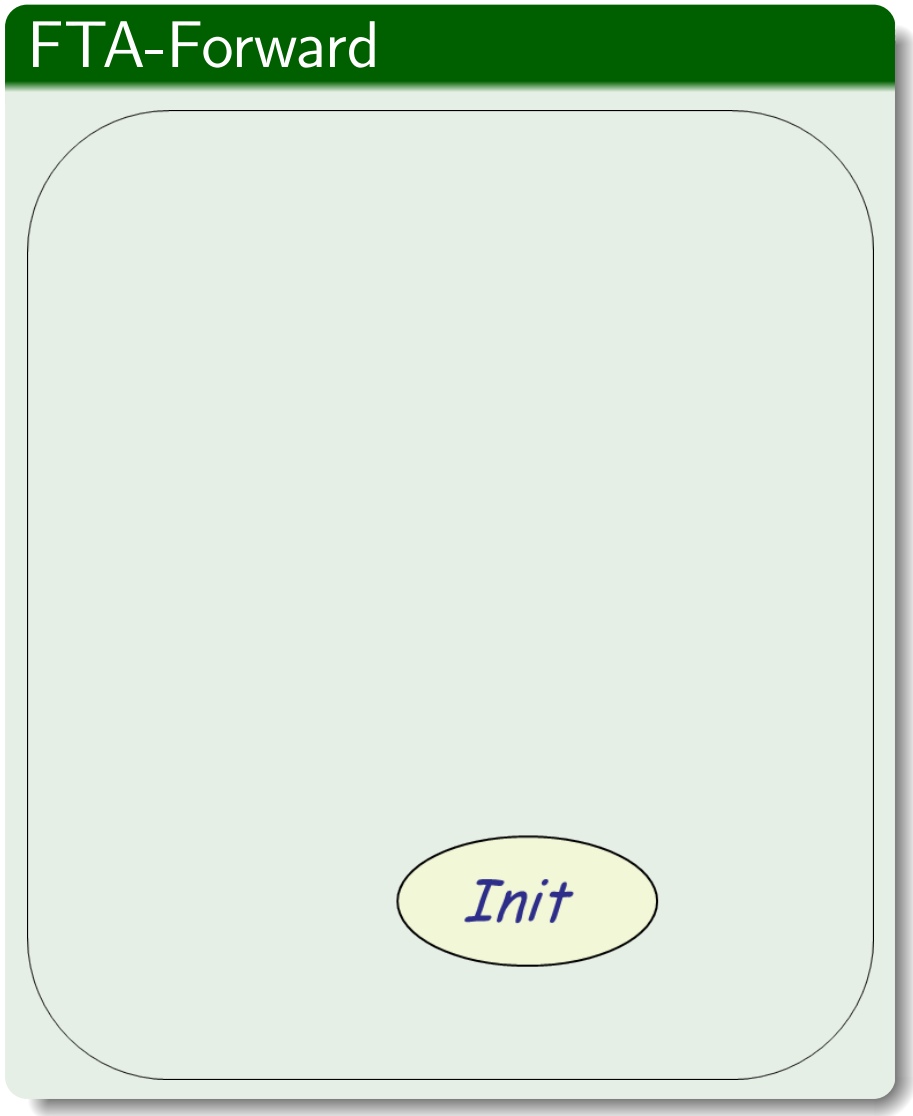
7
8
9
10
11

FTA-Forward

FTA: BDD-based Forward Algorithm

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

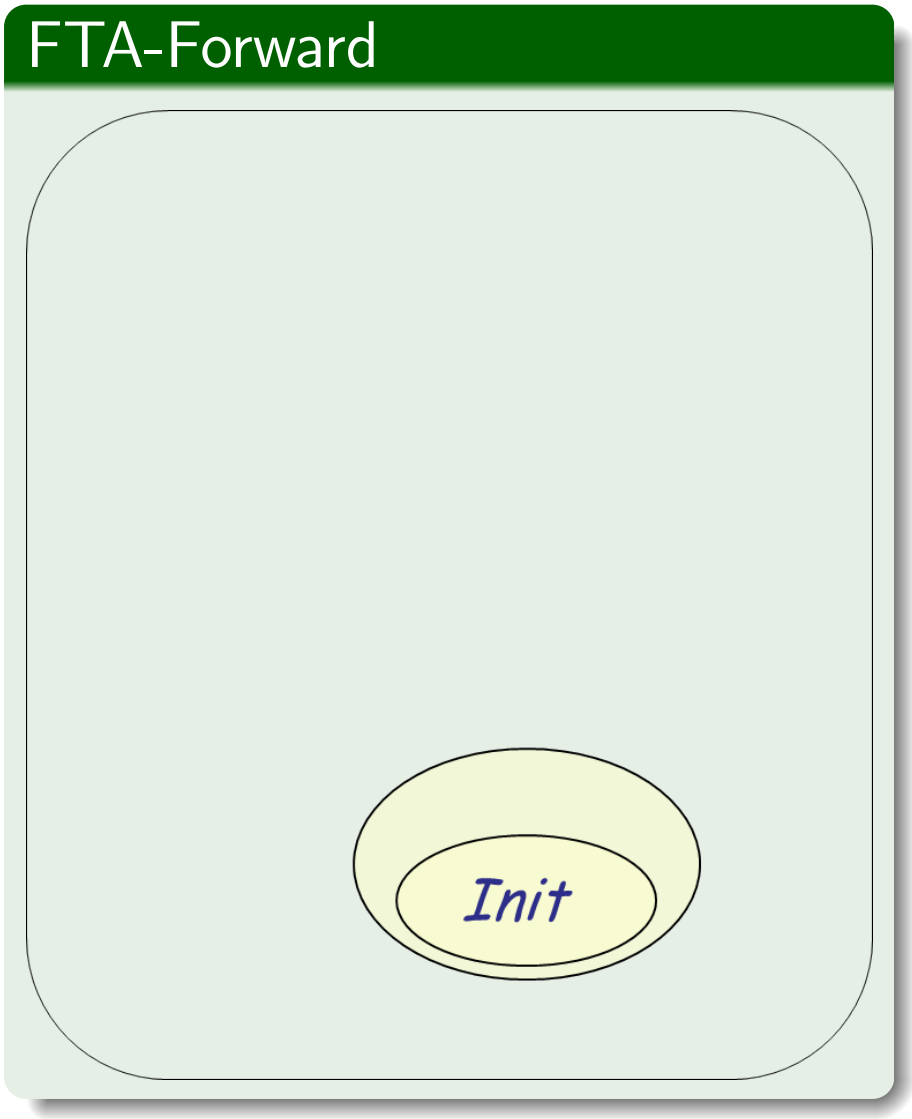
1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4
5
6
7
8
9
10
11



FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(\mathit{Front} \neq \emptyset)$ **do**
5 $\quad\quad \mathit{temp} := \mathit{Reach}$;
6 $\quad\quad \mathit{Reach} := \mathit{Reach} \cup \mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
7 $\quad\quad \mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
8 $\quad$ **end while**;
9
10
11

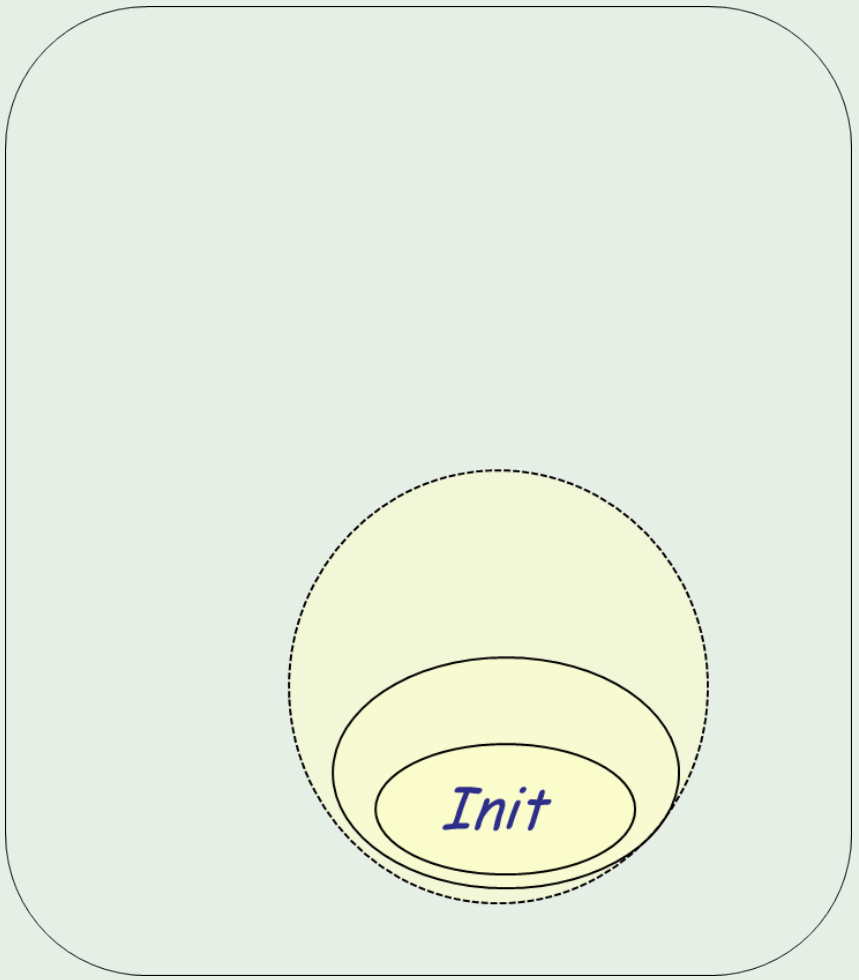


FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(\mathit{Front} \neq \emptyset)$ **do**
5 $\quad\quad \mathit{temp} := \mathit{Reach}$;
6 $\quad\quad \mathit{Reach} := \mathit{Reach} \cup \mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
7 $\quad\quad \mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
8 $\quad$ **end while**;
9
10
11

FTA-Forward

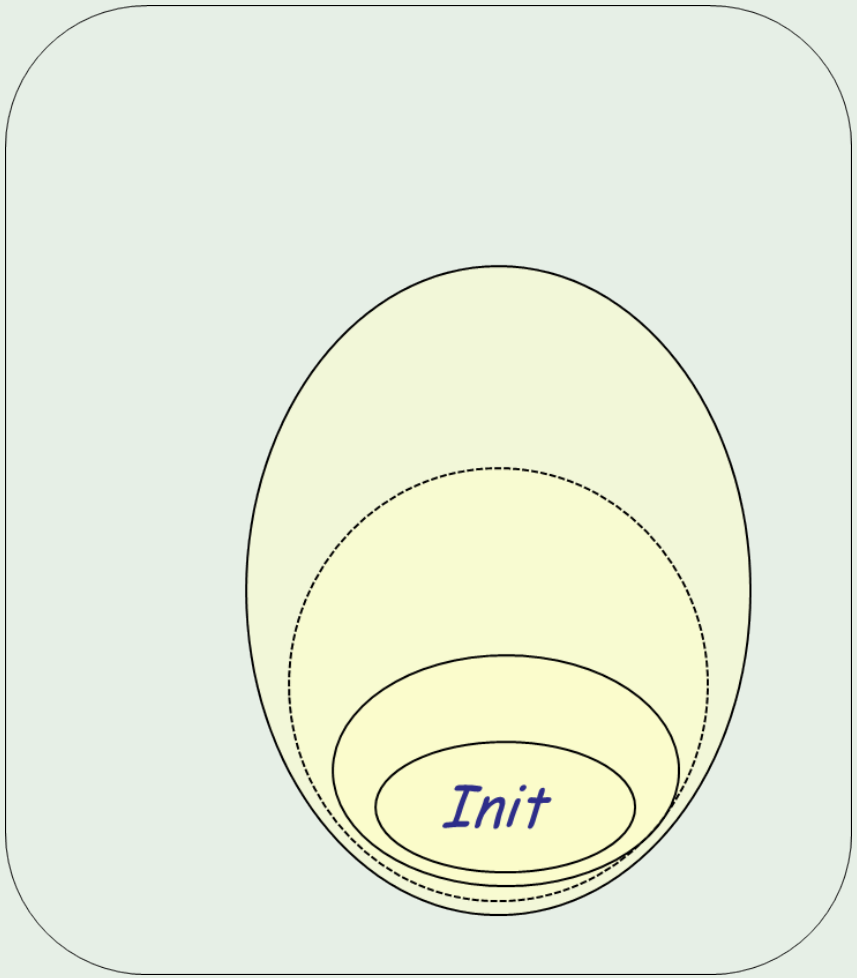


FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(\mathit{Front} \neq \emptyset)$ **do**
5 $\quad\quad \mathit{temp} := \mathit{Reach}$;
6 $\quad\quad \mathit{Reach} := \mathit{Reach} \cup \mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
7 $\quad\quad \mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
8 $\quad$ **end while**;
9
10
11

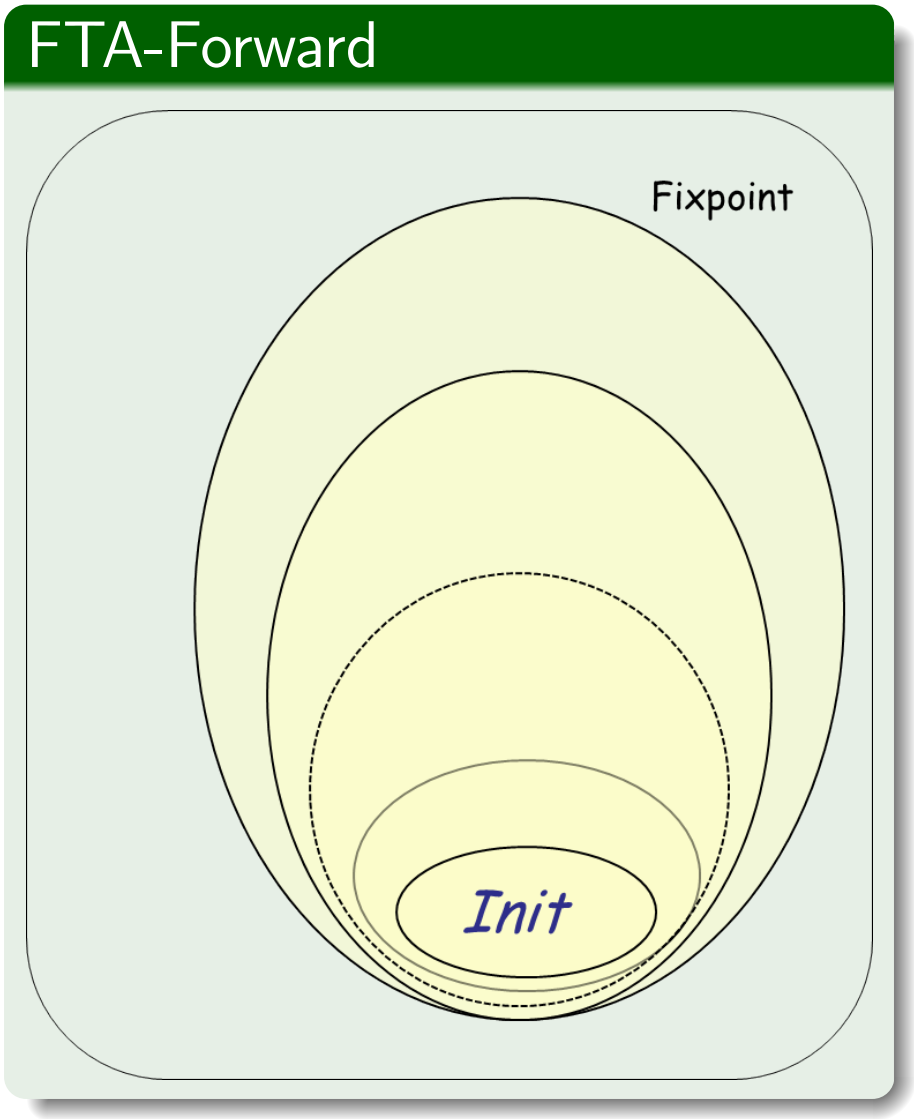
FTA-Forward



FTA: BDD-based Forward Algorithm

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

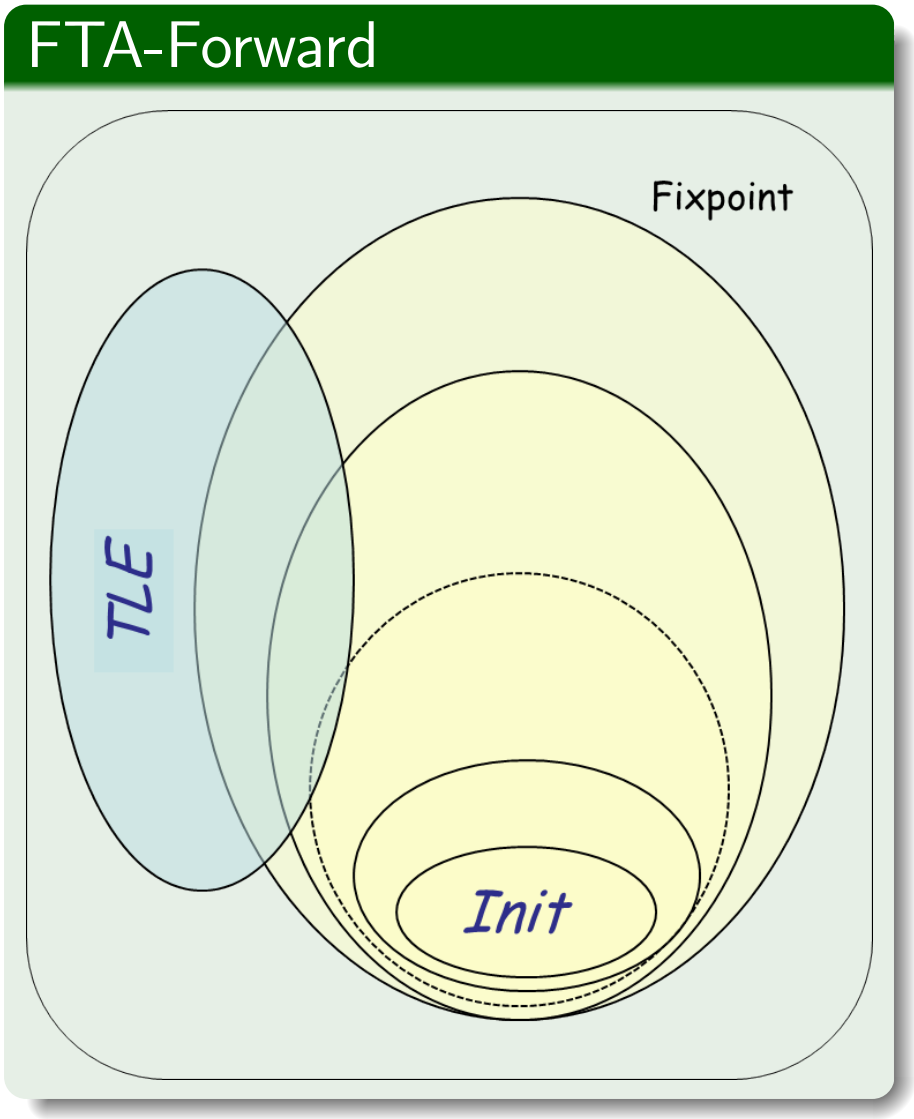
1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(\mathit{Front} \neq \emptyset)$ **do**
5 $\quad\quad \mathit{temp} := \mathit{Reach}$;
6 $\quad\quad \mathit{Reach} := \mathit{Reach} \cup \mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
7 $\quad\quad \mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
8 $\quad$ **end while**;
9
10
11



FTA: BDD-based Forward Algorithm

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

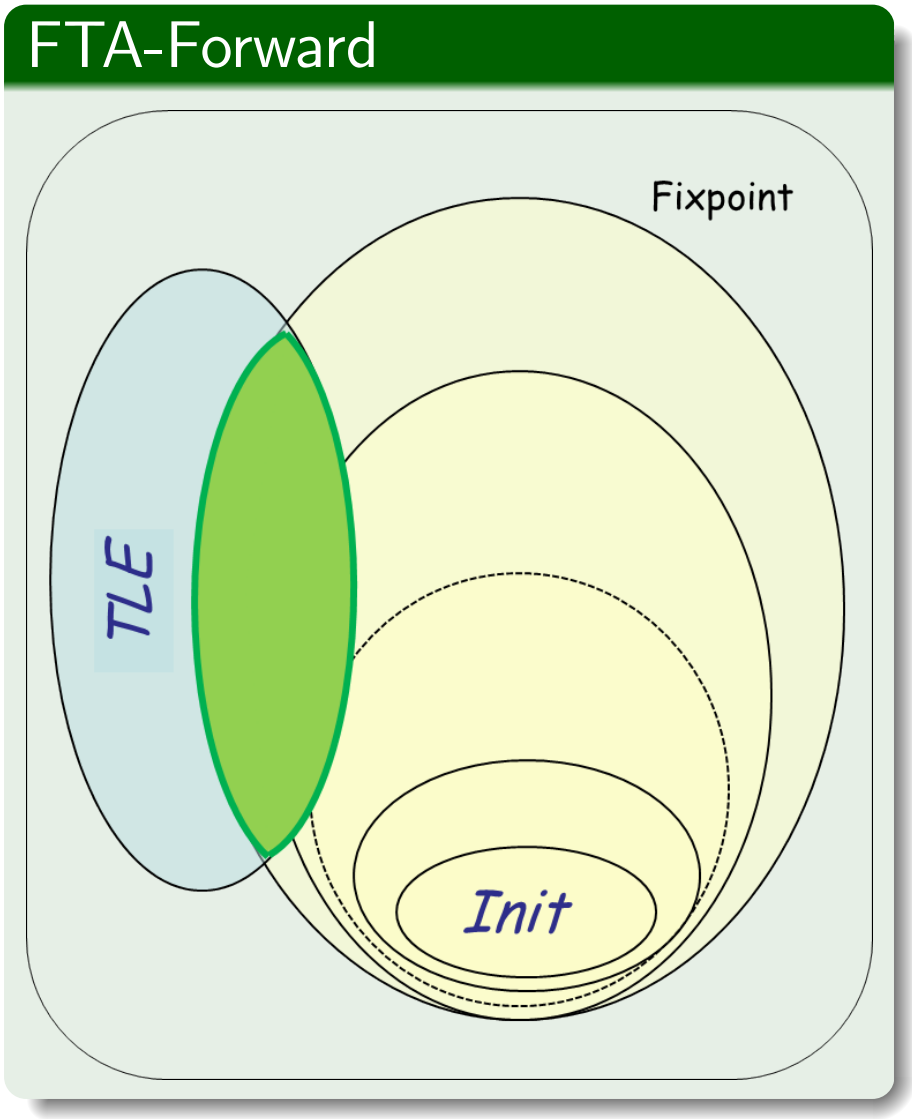
1 $\quad \mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(Front \neq \emptyset)$ **do**
5 $\quad\quad temp := Reach$;
6 $\quad\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
7 $\quad\quad Front := Reach \setminus temp$;
8 $\quad$ **end while**;
9 $\quad CS := Project(\underline{o}, Reach \cap Tle)$;
10
11



FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := \mathit{Extend}(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad \mathit{Reach} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad \mathit{Front} := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(\mathit{Front} \neq \emptyset)$ **do**
5 $\quad\quad \mathit{temp} := \mathit{Reach}$;
6 $\quad\quad \mathit{Reach} := \mathit{Reach} \cup \mathit{fwd\_img}(\mathcal{M}, \mathit{Front})$;
7 $\quad\quad \mathit{Front} := \mathit{Reach} \setminus \mathit{temp}$;
8 $\quad$ **end while**;
9 $\quad \mathit{CS} := \mathit{Project}(\underline{o}, \mathit{Reach} \cap \mathit{Tle})$;
10
11

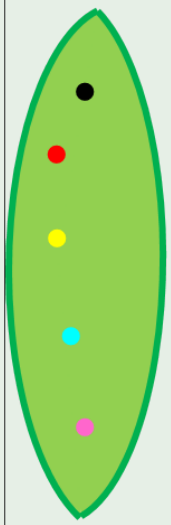


FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(Front \neq \emptyset)$ **do**
5 $\quad\quad temp := Reach$;
6 $\quad\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
7 $\quad\quad Front := Reach \setminus temp$;
8 $\quad$ **end while**;
9 $\quad CS := Project(\underline{o}, Reach \cap Tle)$;
10
11

FTA-Forward



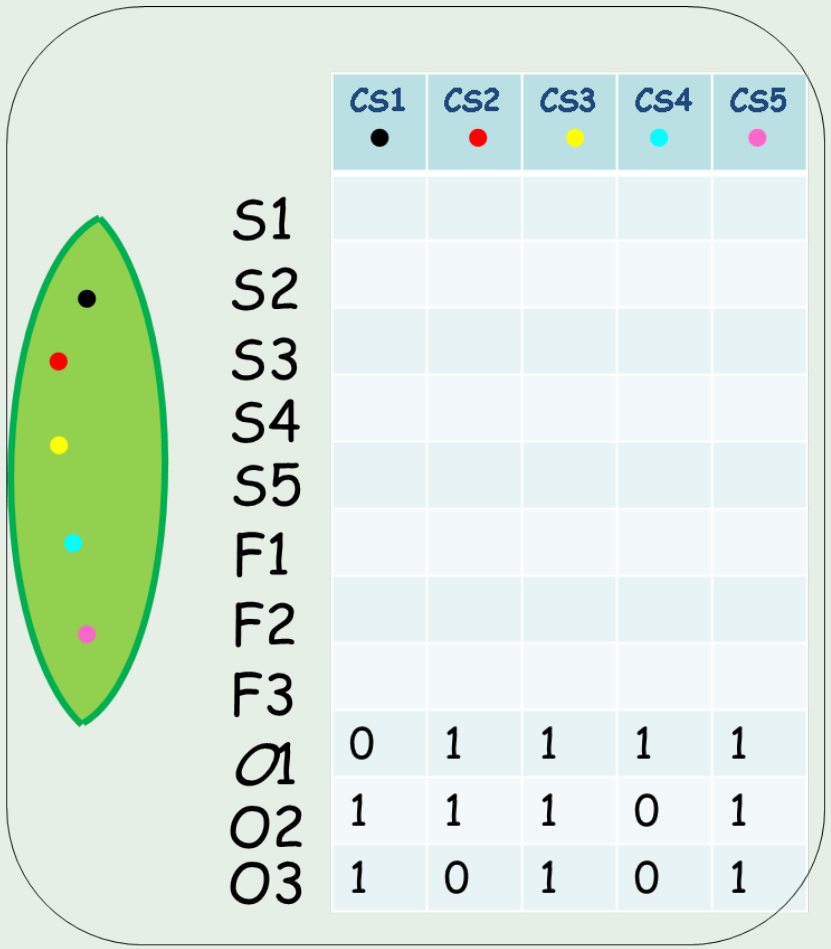
	CS1 ●	CS2 ●	CS3 ●	CS4 ●	CS5 ●
S1	0	1	1	0	1
S2	1	1	0	1	1
S3	1	0	1	0	1
S4	1	0	1	1	1
S5	0	1	1	0	0
F1	0	1	1	1	0
F2	0	0	1	0	1
F3	0	0	1	0	0
O1	0	1	1	1	1
O2	1	1	1	0	1
O3	1	0	1	0	1

FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(Front \neq \emptyset)$ **do**
5 $\quad\quad temp := Reach$;
6 $\quad\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
7 $\quad\quad Front := Reach \setminus temp$;
8 $\quad$ **end while**;
9 $\quad CS := Project(\underline{o}, Reach \cap Tle)$;
10
11

FTA-Forward

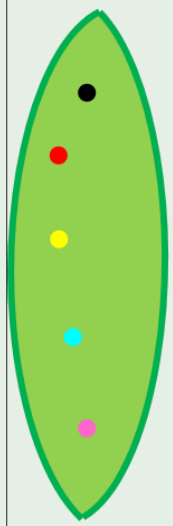


FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(Front \neq \emptyset)$ **do**
5 $\quad\quad temp := Reach$;
6 $\quad\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
7 $\quad\quad Front := Reach \setminus temp$;
8 $\quad$ **end while**;
9 $\quad CS := Project(\underline{o}, Reach \cap Tle)$;
10 $\quad MCS := Minimize(CS)$;
11

FTA-Forward



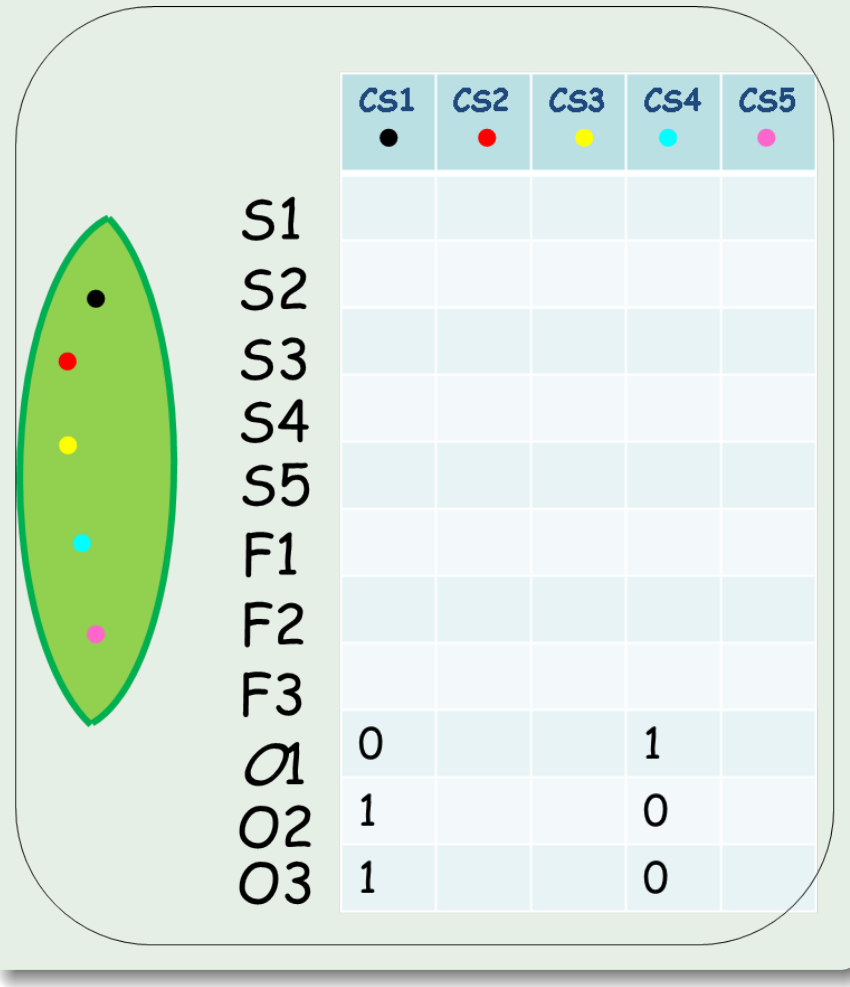
	CS1 ●	CS2 ●	CS3 ●	CS4 ●	CS5 ●
S1					
S2					
S3					
S4					
S5					
F1					
F2					
F3					
O1	0	1	1	1	1
O2	1	1	1	0	1
O3	1	0	1	0	1

FTA: BDD-based Forward Algorithm

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

```
1    M := Extend(M, R°);
2    Reach := I ∩ (o = f);
3    Front := I ∩ (o = f);
4    while (Front ≠ ∅) do
5        temp := Reach;
6        Reach := Reach ∪
                  fwd_img(M, Front);
7        Front := Reach \ temp;
8    end while;
9    CS := Project(o, Reach ∩ Tle);
10   MCS := Minimize(CS);
11
```

FTA-Forward

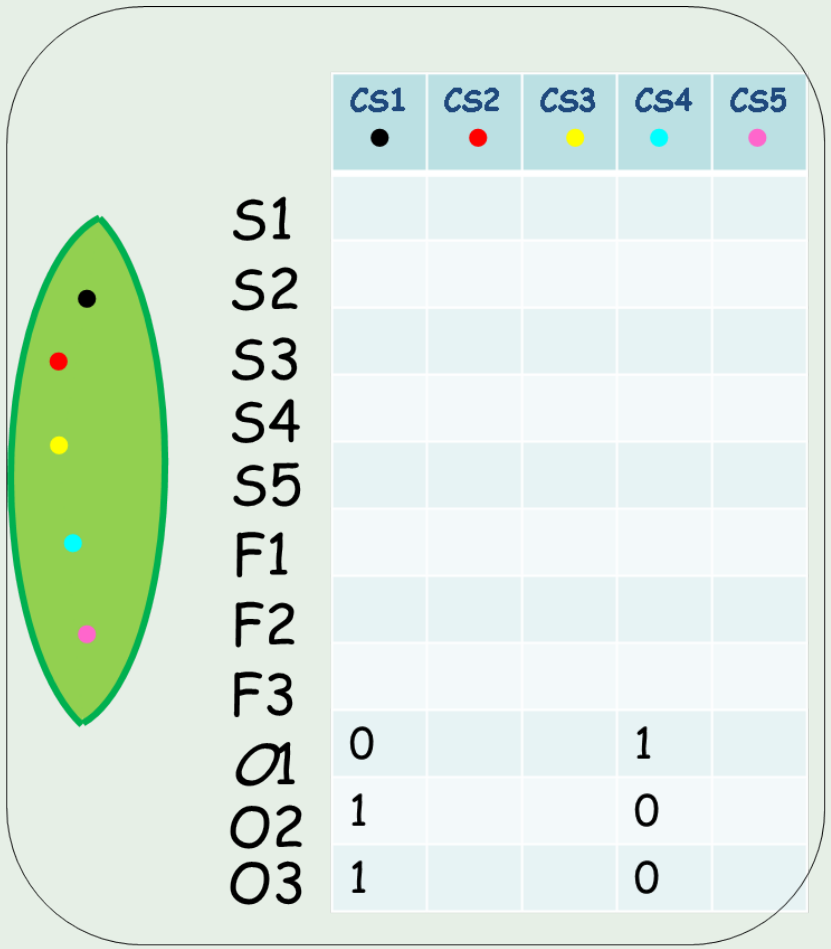


FTA: BDD-based Forward Algorithm

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

```
1    M := Extend(M, R°);
2    Reach := I ∩ (o = f);
3    Front := I ∩ (o = f);
4    while (Front ≠ ∅) do
5        temp := Reach;
6        Reach := Reach ∪
                  fwd_img(M, Front);
7        Front := Reach \ temp;
8    end while;
9    CS := Project(o, Reach ∩ Tle);
10   MCS := Minimize(CS);
11   return Map_{o→f}(MCS);
```

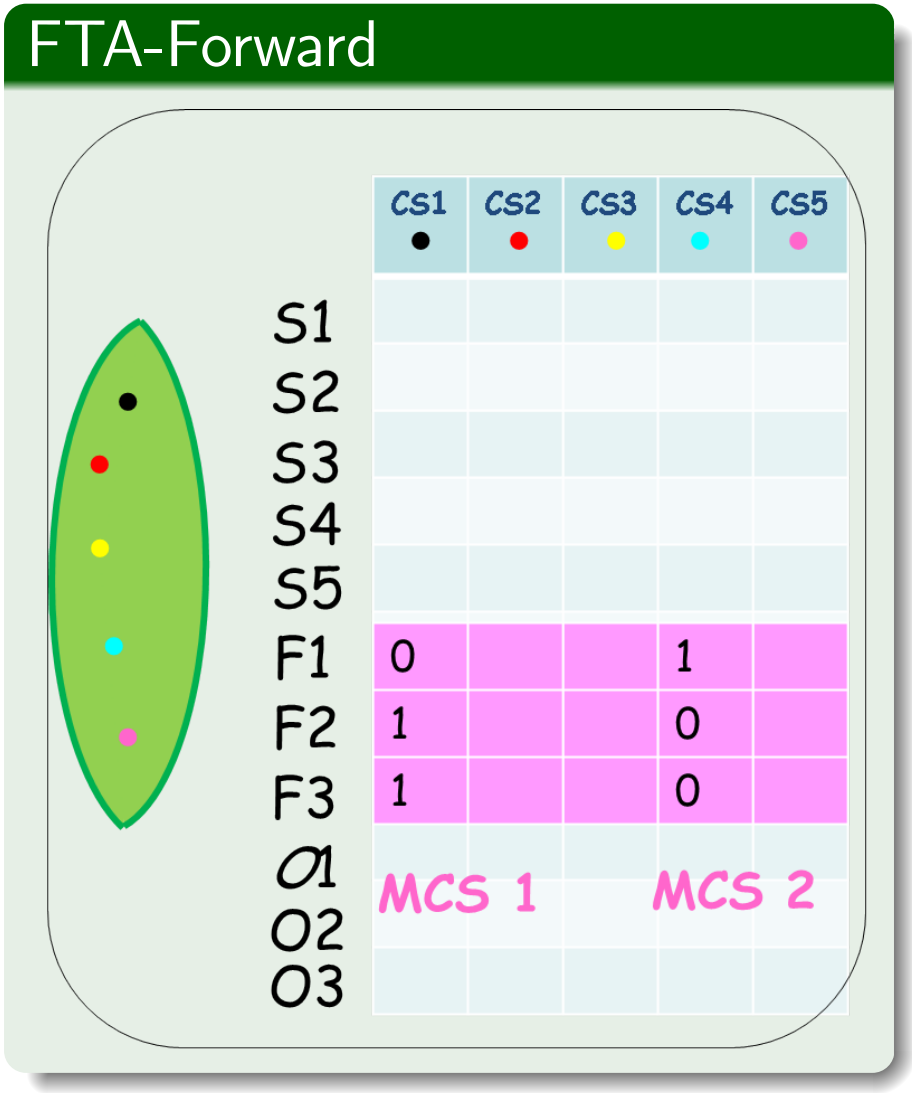
FTA-Forward



FTA: BDD-based Forward Algorithm

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

1 $\quad \mathcal{M} := Extend(\mathcal{M}, \mathcal{R}^o)$;
2 $\quad Reach := \mathcal{I} \cap (\underline{o} = \underline{f})$;
3 $\quad Front := \mathcal{I} \cap (\underline{o} = \underline{f})$;
4 $\quad$ **while** $(Front \neq \emptyset)$ **do**
5 $\quad\quad temp := Reach$;
6 $\quad\quad Reach := Reach \cup fwd\_img(\mathcal{M}, Front)$;
7 $\quad\quad Front := Reach \setminus temp$;
8 $\quad$ **end while**;
9 $\quad CS := Project(\underline{o}, Reach \cap Tle)$;
10 $\quad MCS := Minimize(CS)$;
11 $\quad$ **return** $Map_{\underline{o} \to \underline{f}}(MCS)$;

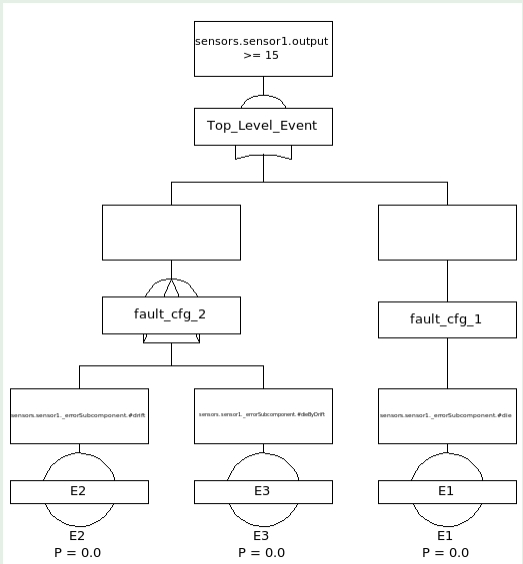


Fault Tree Analysis (FTA)

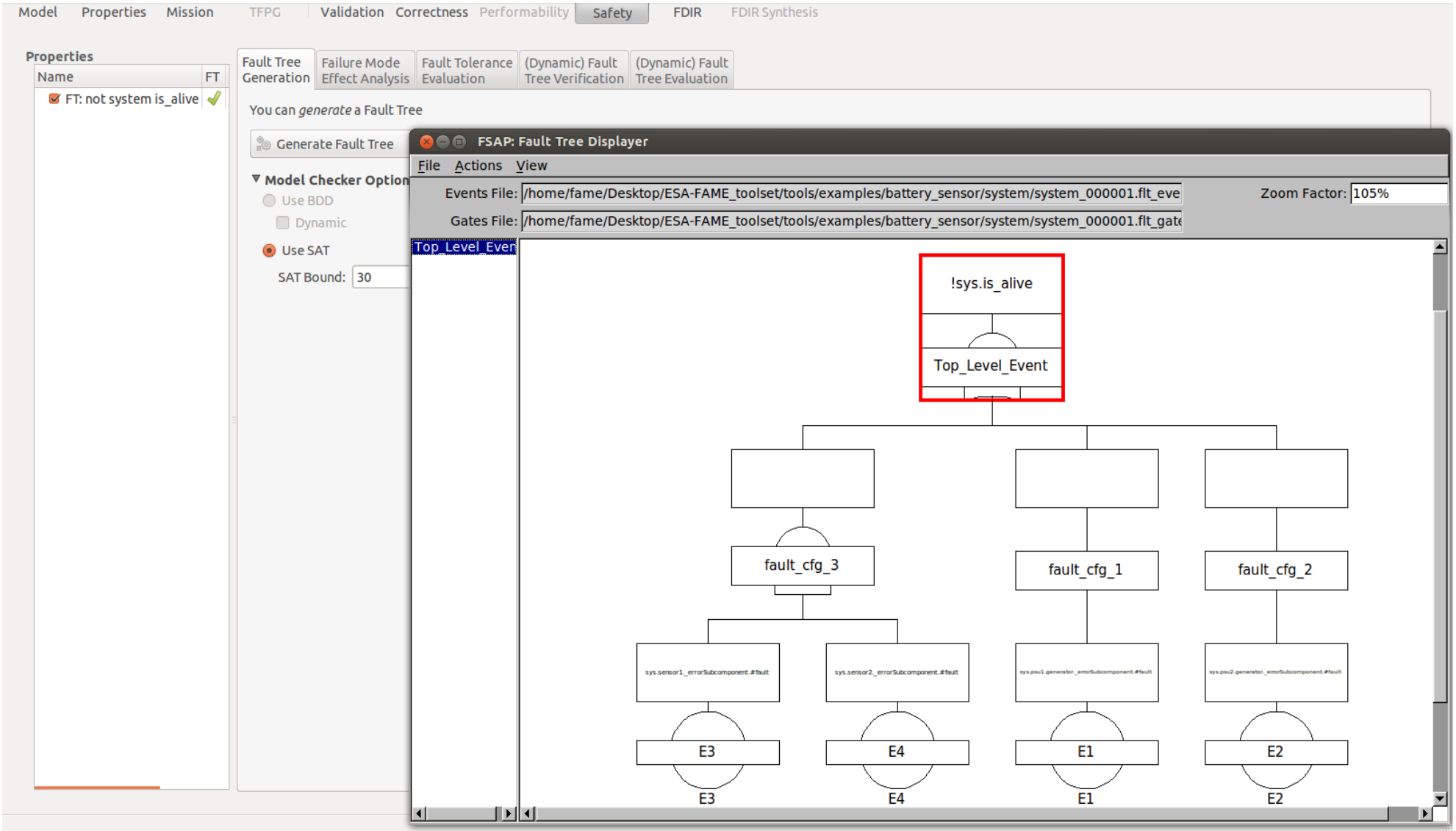
Dynamic FTs

- 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

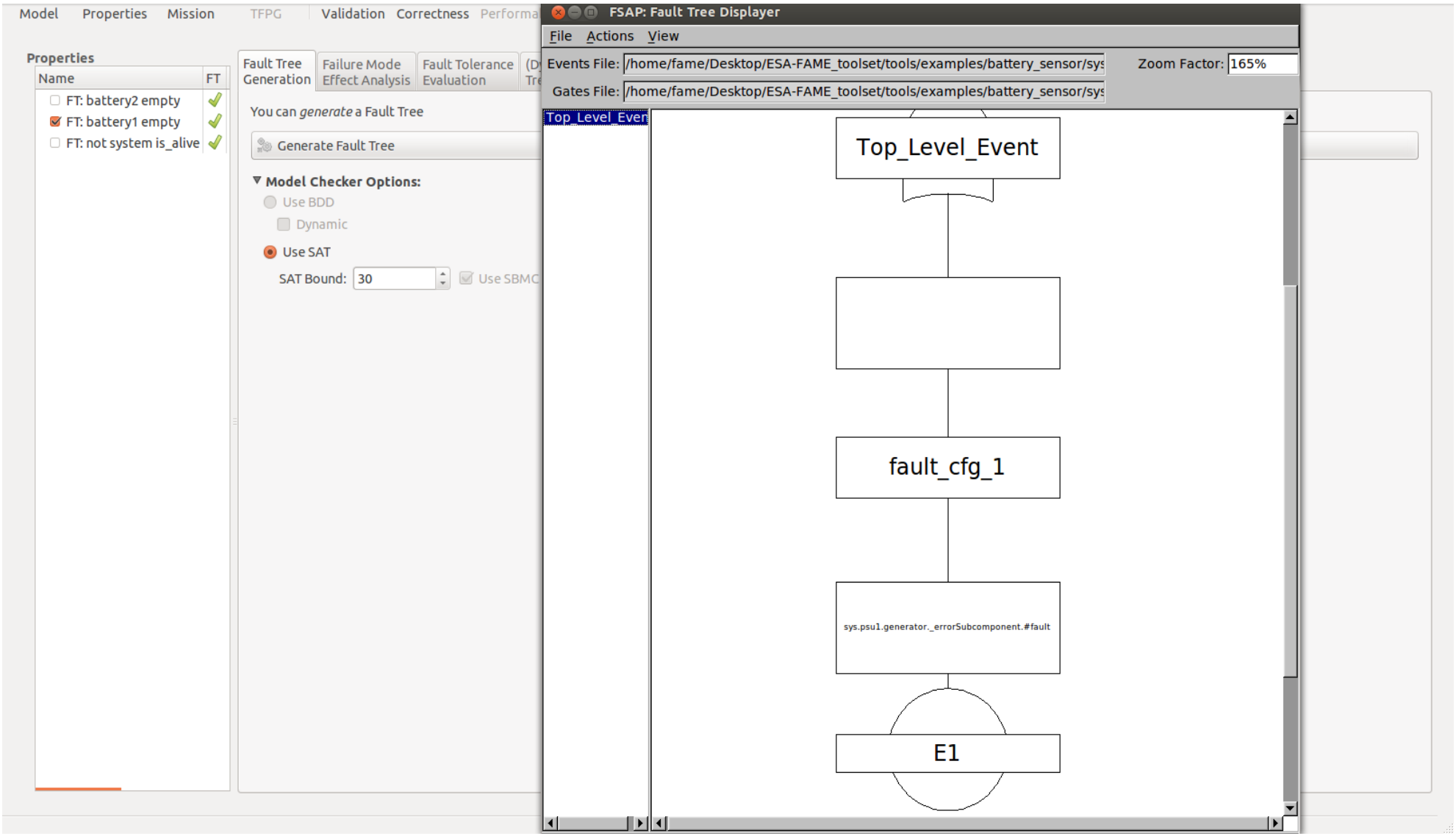
Dynamic Fault Tree



Screenshot: Fault Tree Analysis



Screenshot: Fault Tree Analysis



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 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

Failure Mode and Effects Analysis (FMEA)

FMEA Table

An FMEA table is a set of pairs linking a fault configuration (set of faults) FC_i to a property P_j. That is, P_j may be violated when the system fails according to FC_i.

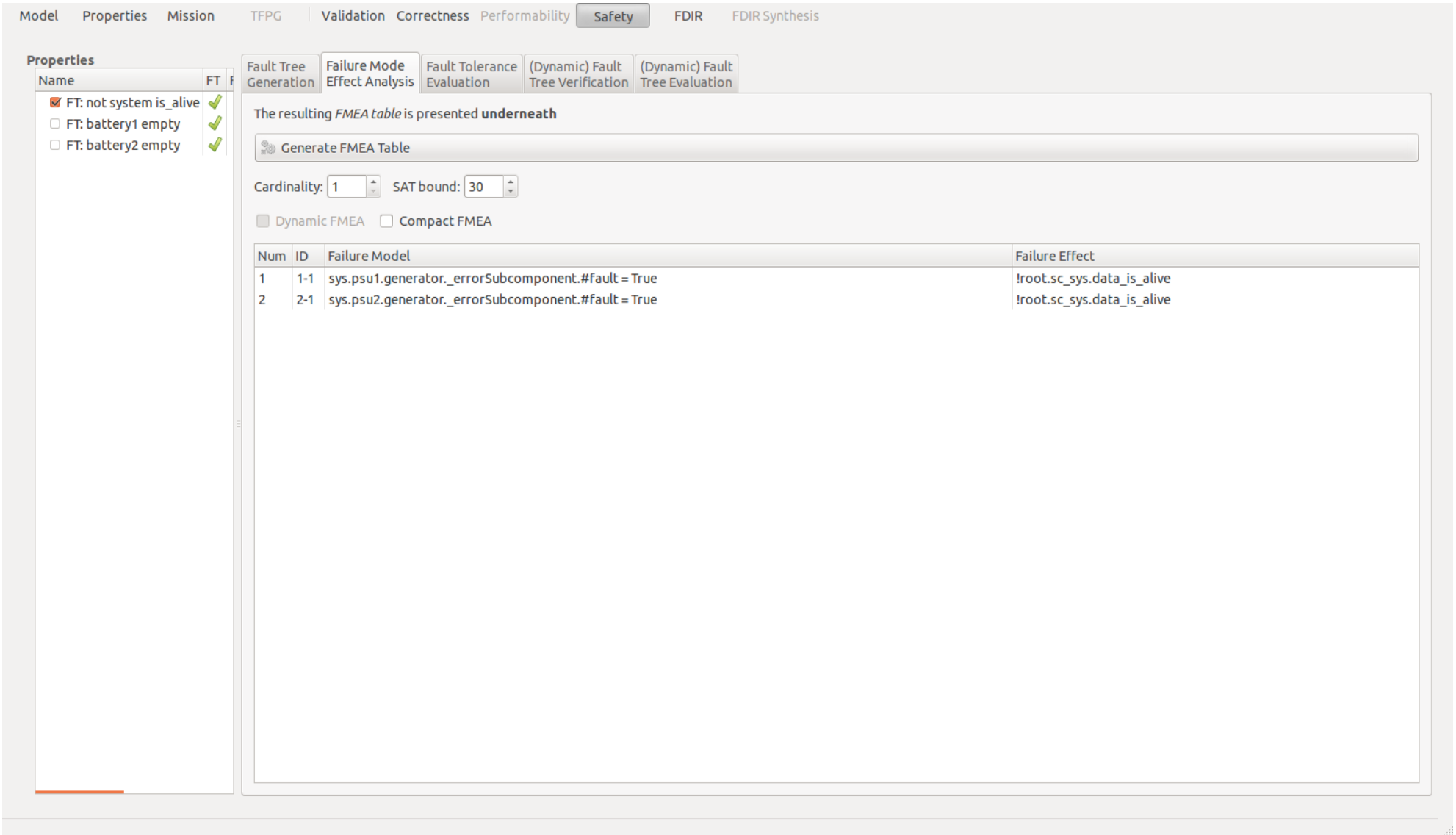
Cardinality of FMEA Tables

- Often, cardinality one (single fault) is considered
- Generalizes to FMEA table of cardinality k (including fault configurations of cardinality up to k)

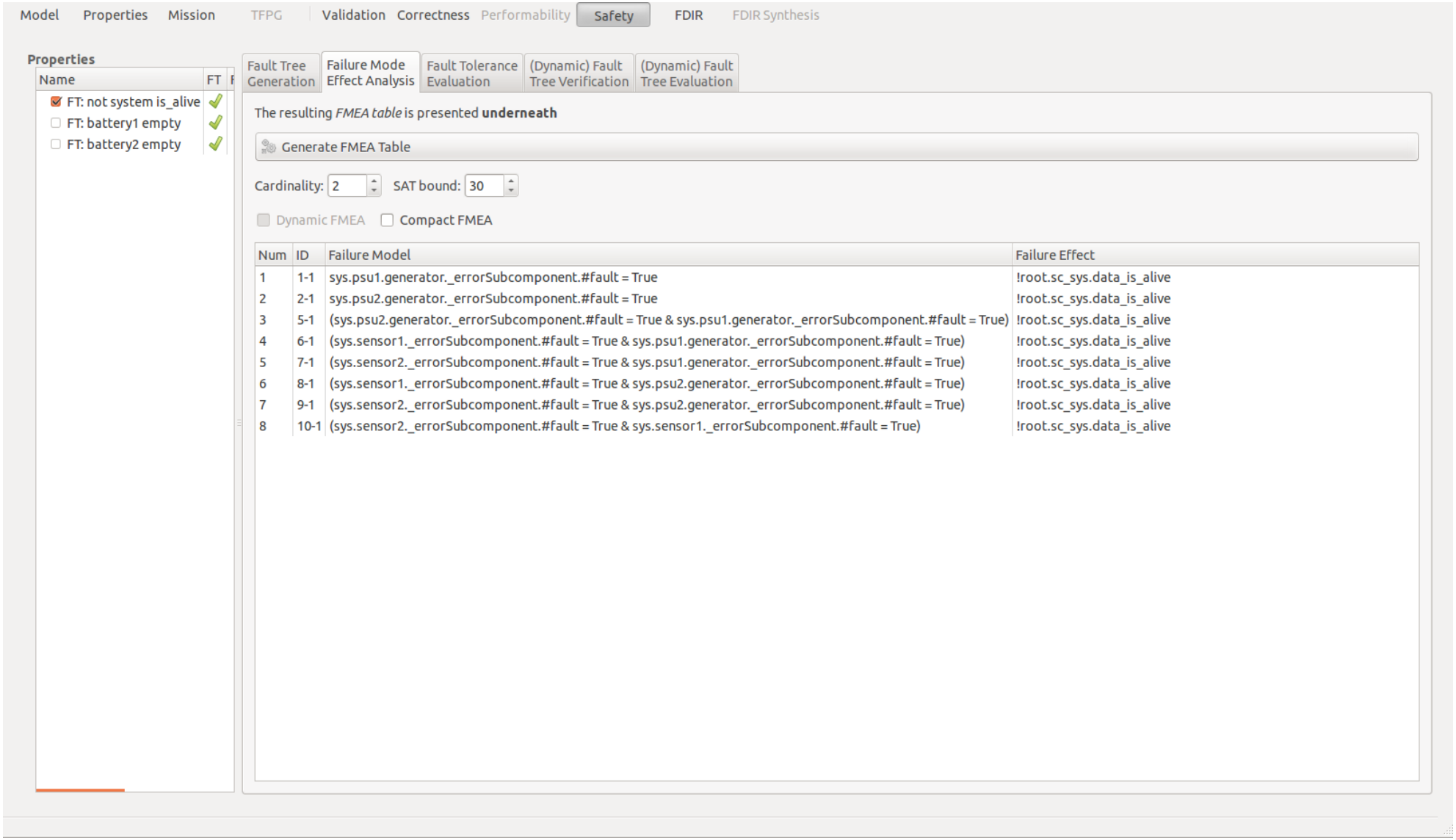
Dynamic FMEA Tables

- Order between fault events may be imposed

Screenshot: Failure Modes and Effects Analysis



Screenshot: Failure Modes and Effects Analysis



Outline

Fault Detection, Isolation and Recovery

FDIR

- Fault Detection, Isolation and Recovery (FDIR) is an essential block of safety-critical systems – needed to ensure fault tolerance and prevent safety hazards

Goals of FDIR

- Fault detection: identify malfunctions
- Fault isolation: precisely identify the fault responsible for a malfunction
- Fault recovery: recover after a fault has occurred, e.g. reconfiguring the system or switching operational mode

FDIR in COMPASS

FDIR Effectiveness Analysis

- Evaluate the effectiveness of an existing FDIR

Diagnosability Analysis

- Check if there exists a diagnoser that can infer at run-time accurate and sufficient information on the behavior of the plant

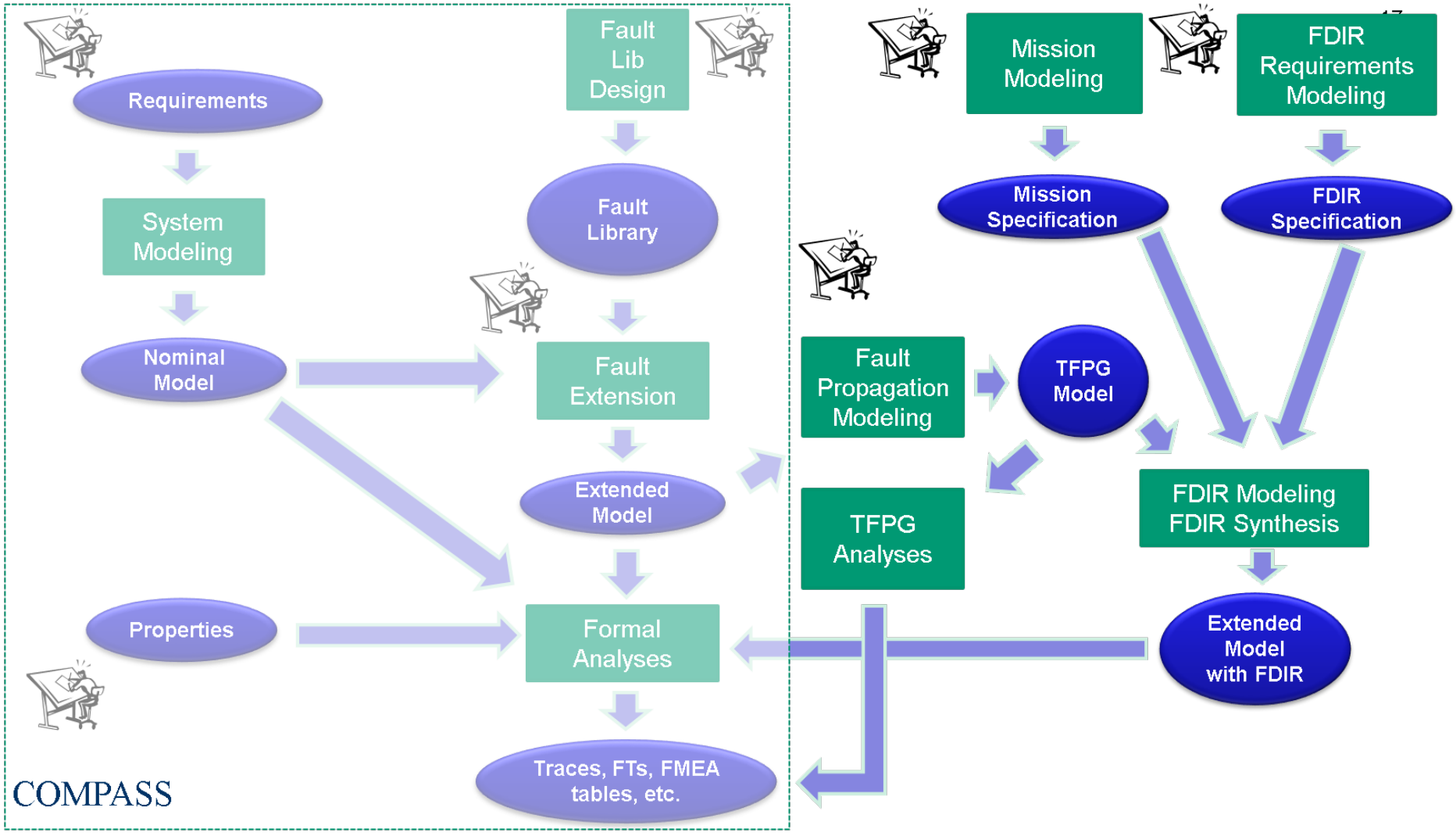
FDIR Synthesis

- Automatically synthesize an FDIR component

Fault Propagation Analysis

- See presentation on "Fault propagation modeling and analysis via TFPG" by Benjamin Bittner

The FAME Environment



Outline

Ongoing Activities

Efficient routines for FTA

- Efficient IC3-based routines, based on parameter synthesis
- Compute MCSs by layers of increasing cardinality
- Anytime techniques: compute lower and upper bound of TLE probabilities, at each intermediate layer

Contract-Based Safety Assessment (CBSA)

- Tight integration between architectural design and safety assessment
- Based on iterative refinement
- Generation of Hierarchical Fault Trees
- See presentation on "Contract-based verification of AADL models" by Stefano Tonetta

FDIR

- Failure Propagation Analysis – validation and synthesis of TFPGs

References

- Safety Critical Systems (Storey, Addison-Wesley 1996)
- System Safety (Leveson, Addison-Wesley 1995)
- Formal Safety Assessment (Bozzano, Villafiorita, Taylor & Francis 2010)
- FTA (Fault Tree Handbook, NASA 2002)
- Algorithms for FTA (Bozzano et. al, ATVA 2007, CAV 2015)
- Contract-Based Safety Assessment (Bozzano et. al, ATVA 2014)
- Formal Framework for FDI (Bozzano et. al, TACAS 2014)
- FAME (Bittner et. al, IMBSA 2014)