A Modern Perspective on Fault Tree Analysis

Joost-Pieter Katoen and Matthias Volk

Joint work with: Majdi Ghadhab (BMW), Dennis Guck (TWT),
Sebastian Junges (RWTH), Matthias Kuntz (BMW),
Enno Ruijters (U. Twente) and Mariëlle Stoelinga (U. Twente)

Tutorial MMB 2018, Erlangen, BY

Roadmap of This Tutorial

Part 1. What are Dynamic Fault Trees?

- DFT Elements, Benchmarks, Intricacies, DFTs as Stochastic Petri Nets

Part 2. From DFTs to Markov Models, Compositionally

- Compositional State-Space Minimisation, Non-Determinacy

Part 3. From DFTs to Markov Models, Monolithically

- Symmetry Reduction, Don't Care Propagation

Part 4. DFT Analysis by Model Checking

- Reliability Measures, Core Algorithms, `Storm` Tool

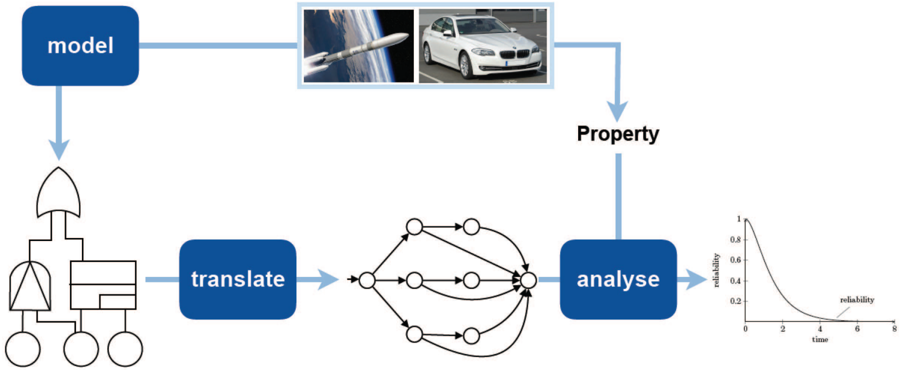
Part 5. Advanced Optimisations

- Graph Rewriting, Partial State-Space Generation

Part 6. Industrial Applications and Outlook

Focus is on conveying intuition and experimental results

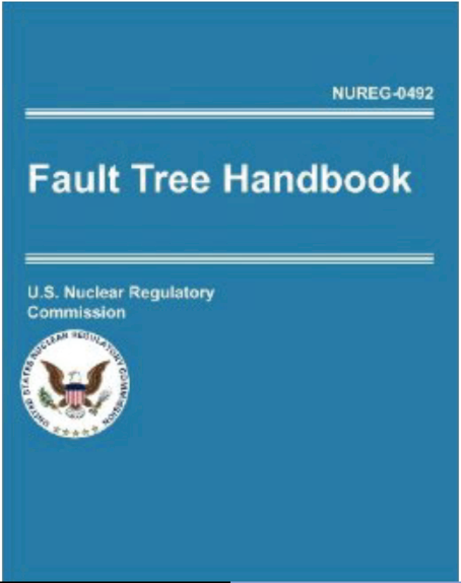
Graphical Overview



Overview Part 1 and 2

Overview

Reliability Engineering



Reliability Engineering

- Risk analysis ensures that critical assets, like medical devices and nuclear power plants, operate in a safe and reliable way.
- Fault tree analysis (FTA) is one of the most prominent techniques.
- Used by a wide range of industries (aerospace, automotive, nuclear, medical, process engineering)
- Used by many companies and institutions: FAA, NASA, ESA, Airbus, Honeywell, etc.
- Industrial standards by the IEC and by ISO for automotive applications

The SpaceEx Falcon-9 Explosion

A launch failure in 2015 resulted in a loss of a quarter billion dollars.

Overview

Fault Trees

- Fault trees (FTs) are a graphical method that model how failures propagate through the system
- They model how do component failures lead to system failures?
- Not all component failures lead to a system failure

  due to redundancy, spare management, etcetera.

Reliability: Static Fault Trees [Watson, 1961–62]

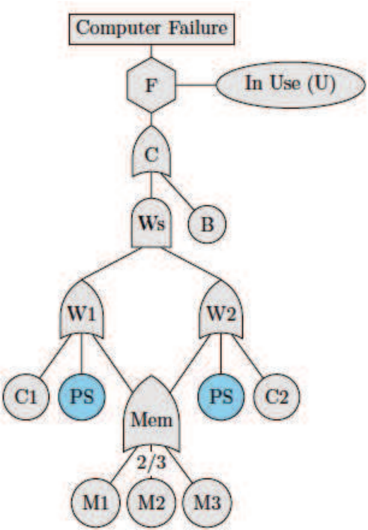
- Fault tree is a directed acyclic graph consisting of two types of nodes: events (depicted as circles) and gates:

(a) AND gate (b) OR gate (c) k/N gate (d) INHIBIT gate

- An event is an occurrence within the system, typically the failure of a component or sub-system.
- Events can be divided into:
  - basic events (BEs), which occur on their own, and
  - intermediate events, which are caused by other events
- The root, called the top level event (TLE), models a system failure

Static Fault Trees

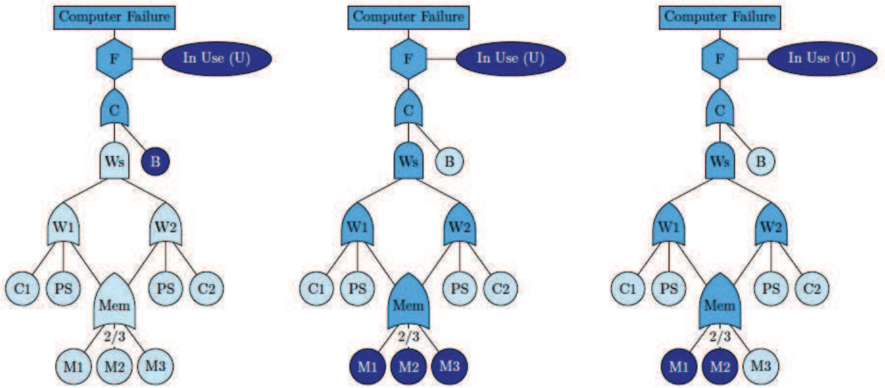
[Watson, 1961–62]



Legend:

F: Computer failure while in use
C: Computer failure
Ws: Failure of both workstations
B: Bus failure
W1: Failure of workstation 1
W2: Failure of workstation 2
C1: Failure of CPU 1
C2: Failure of CPU 2
PS: Failure of power supply
Mem: Failure of memory system
M1: Failure of memory module 1
M2: Failure of memory module 2
M3: Failure of memory module 3

Minimal Cut Sets

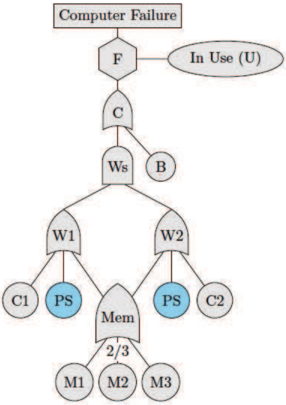


minimal — not minimal — minimal

A cut set is a set of components that together can cause the system to fail.

A minimal cut set is a cut set without proper subset being a cut set.

Boolean Manipulation



- Turn SFT into a propositional Boolean formula
- Top-down: F, $F \wedge C$, $F \wedge (B \vee Ws)$, $(F \wedge B) \vee (F \wedge Ws)$, ...
- Halt when all gates are eliminated
- This yields all MCSs
- Bottom-up: $W1 = C1 \vee PS \vee Mem$,
- $Mem = (M1 \wedge M2) \vee \ldots \vee (M2 \wedge M3)$, ...
- This yields cut sets for all gates

Efficient implementation: using BDDs.

Probability calculations are done on top of the analysis using minimal cut sets.

Methods for Minimal Cut Sets

Author	Method	Remarks	Tool
Vesely et al. [VGRH81]	Top-down	Classic boolean method	MOCUS [FHM74]
Vesely et al. [VGRH81]	Bottom-up	Produces MSC for gates	MICSUP [PSC75]
Coudert and Madre [CM93]	BDD	Usually faster than classic methods	MetaPrime [CM94]
Rauzy [Rau93]	BDD	Only for coherent FTs but faster than [CM93]	Aralia [RD97]
Dutuit and Rauzy [DR96]	Modular BDD	Faster for FTs with independent submodules	DIFTree [DVG97]
Remenyte et al. [RA06, RPA08]	BDD	Comparison of BDD construction methods	-
Codetta-Raiteri [CR06]	BDD	Faster when FT has shared subtrees	-
Xiang et al. [XYM+11]	Minimal Cut Vote	Reduced complexity with large voting gates	CASSI [XYM+11]
Carrasco et al. [Cn99]	CS-Monte Carlo	Less complex for FTs with few MCS	-
Vesely and Narum [VN70]	Monte Carlo	Low memory use, accuracy not guaranteed	PREP [VN70]

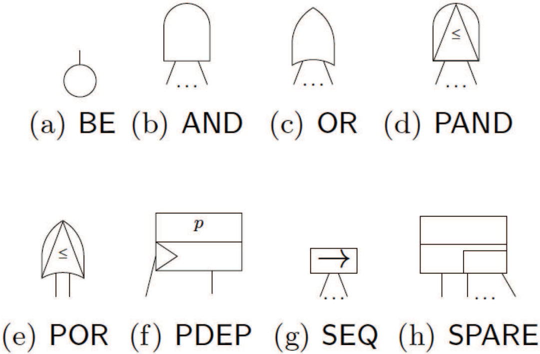
Analysing static fault trees is relatively simple as the ordering of failures is irrelevant.
It only matters whether an event has occurred or not.

Deficiencies of Static Fault Trees

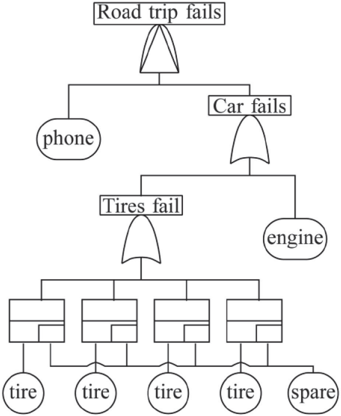
Main limitations of static fault trees:

- Too simple for practical systems
- They lack common dependability patterns, such as:
  - spare management
  - functional dependencies
  - redundancies
- Static behaviour:
  - TLE failure only depends on the set of failed events, not on any temporal ordering of faults

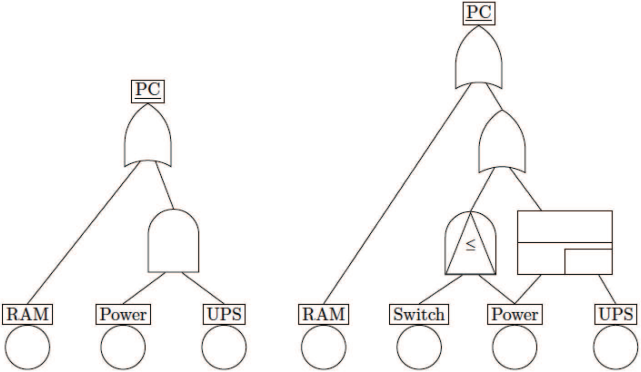
Dynamic Fault Trees [Dugan *et al.*, 1990]



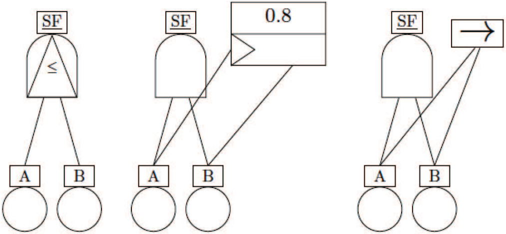
A Simple DFT Example



A Simple DFT Example

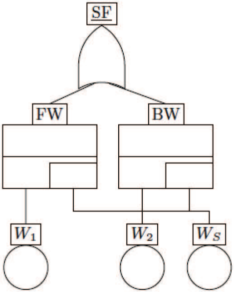


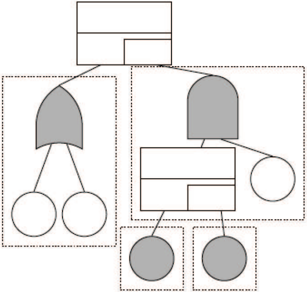
Dynamic Gates: PAND, FDEP, and SEQ



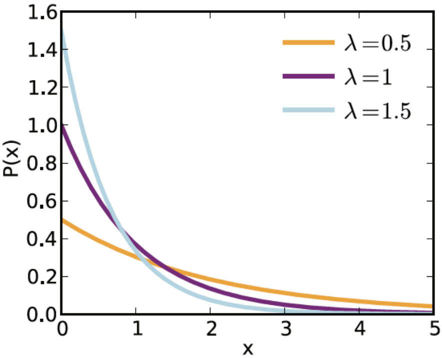
(a) (b) (c)

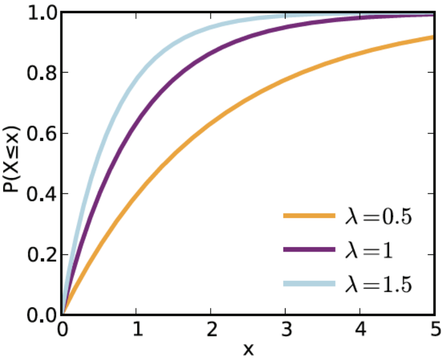
The SPARE Gate





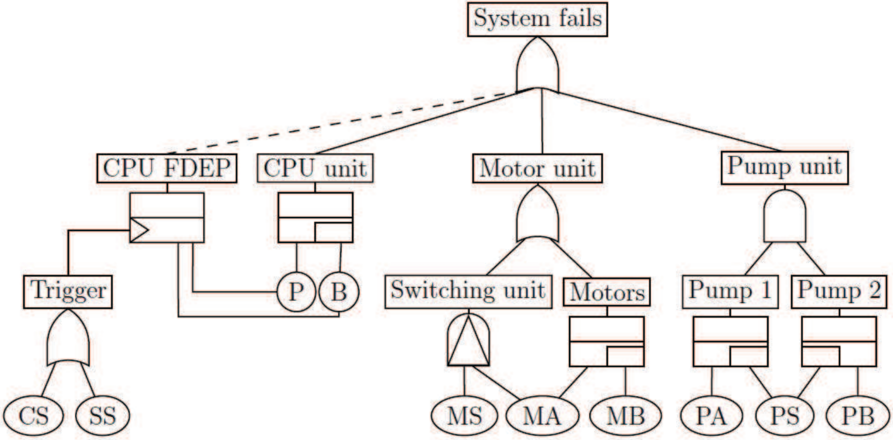
Failures := Exponential Distributions



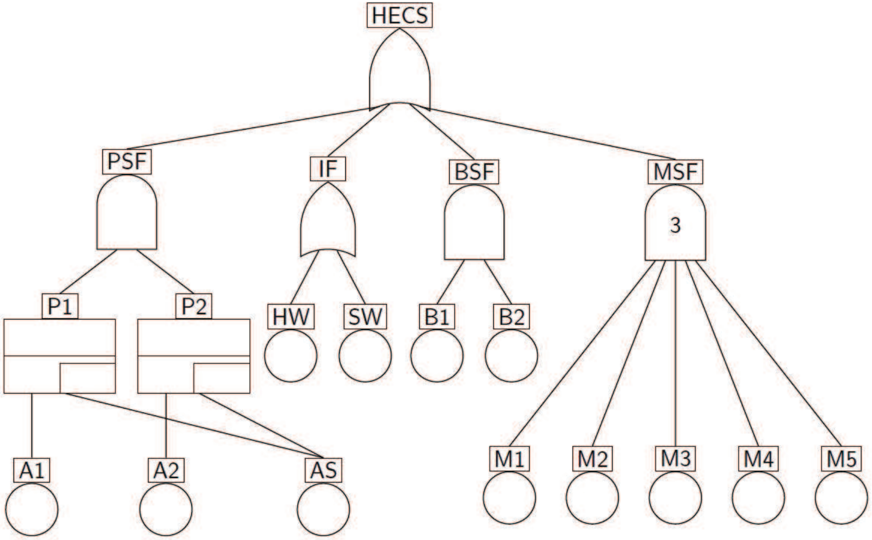


The higher the rate λ, the faster the cdf approaches 1.

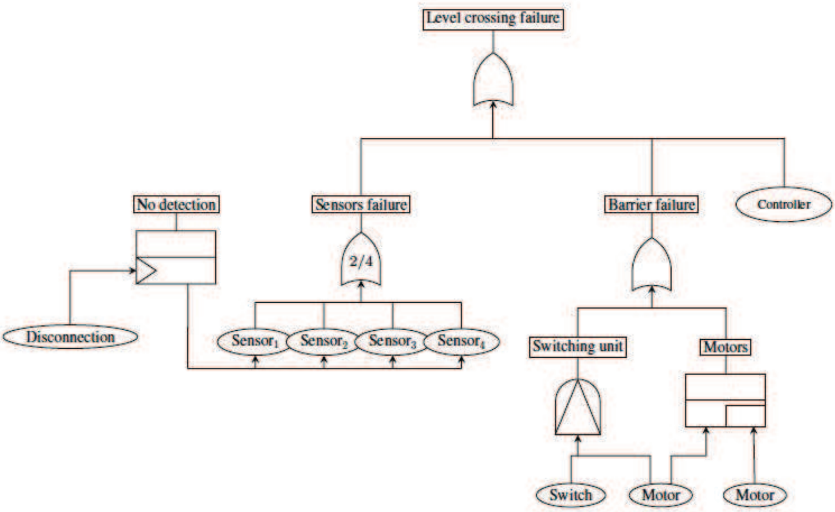
Benchmark DFT: Cardiac Assist System [Boudali & Dugan, 2005]



Benchmark DFT: HECS [NASA Handbook, 1982]

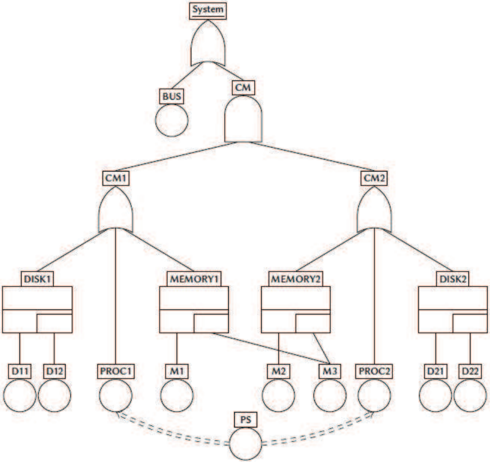


Benchmark DFT: Railway Crossing [Guck *et al.*, 2014]



Benchmark DFT: MCS

[Malhotra & Trivedi, 1995]



Multiprocessor Computing System

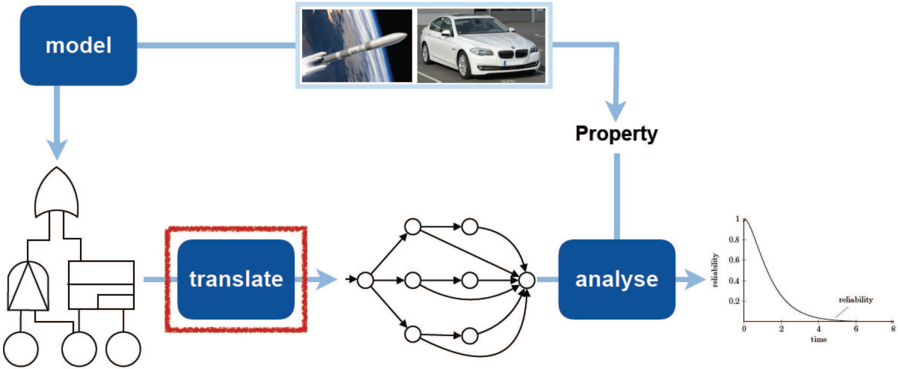
The Price of DFTs

- Analysis can no longer be done using minimal cut sets
  - Generalisations towards cut sequences are insufficient
  - ⇒ The DFT behaviour is history-dependent
- DFT analysis is done by generating a stochastic (decision) process [1]
  - Monolithic approach
  - Compositional approach
  - Approaches via other models (e.g., Bayes' networks or Petri nets)
- Use Markov Chain analysis techniques to obtain quantitative measures
  - We use probabilistic model checking

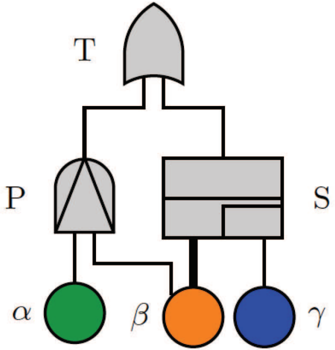
[1] Continuous-Time Markov Chain.

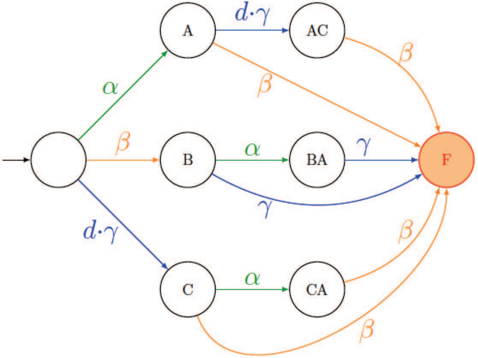
Overview

Graphical Overview



The Markov Model of a DFT





$d :=$ dormancy factor

State-space generation
is a major bottleneck in DFT analysis.

Myths About Dynamic Fault Trees

"Although DFTs are powerful in modeling systems with dynamic failure behaviors, their quantitative analyses are pretty much troublesome, especially for large scale and complex DFTs."

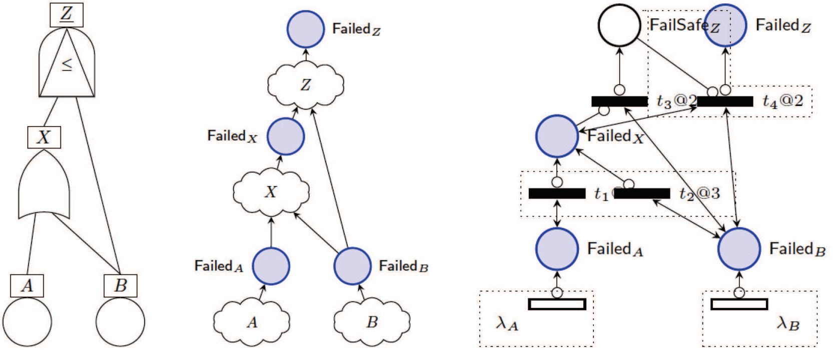
[Ge *et al.*, Rel. Eng. Syst. Safe, 2015]

"Although many extensions of fault trees have been proposed, they suffer from a variety of shortcomings. In particular, even where software tool support exists, these analyses require a lot of manual effort."

[Kabir, Expert Syst. Appl., 2017]

We will show that these are myths. Scalable DFT analysis is possible.

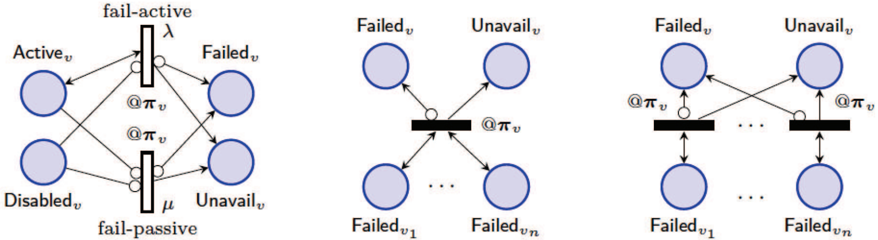
Use Nets to Capture The Meaning of DFTs



(a) DFT (b) Basic scheme (c) Simplified resulting GSPN

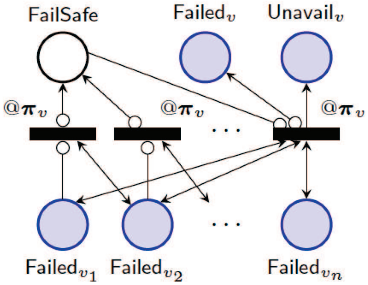
This provides a meaning to DFTs in a compositional manner

Nets for Static Gates

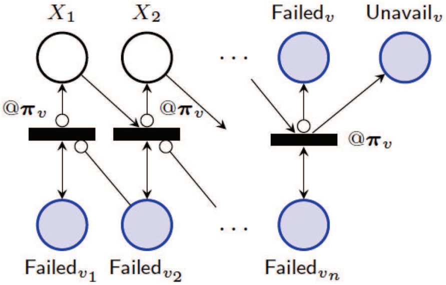


(a) BE (b) AND (c) OR

Nets for PANDs

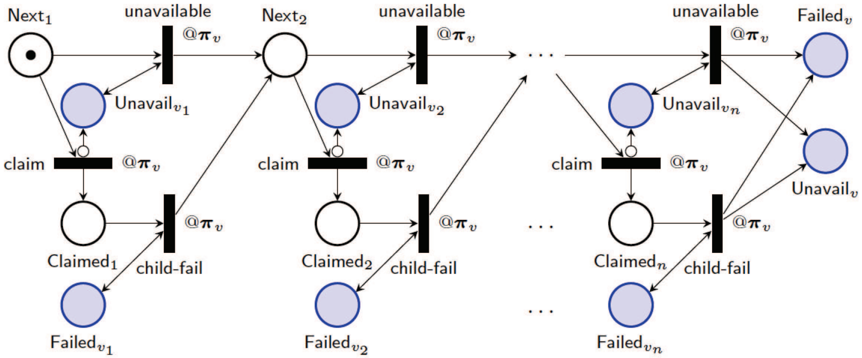


(a) Inclusive PAND$_\leq$



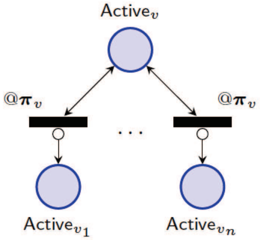
(b) Exclusive PAND$_<$

A Net for the SPARE Claiming Mechanism

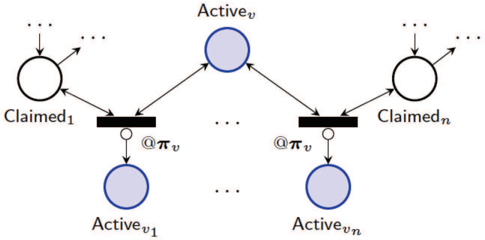


Claiming: a SPARE uses one of its children. If this child fails, the SPARE tries to claim another child (left to right). Only operational children that are not claimed by another SPARE can be claimed. If claiming fails – all spare components have failed – the SPARE fails.

A Net for the SPARE Activation Mechanism



(a) Gate



(b) SPARE

Activation: Nodes outside spare "modules" are disabled by default. For each active SPARE and used child v, the nodes in v's spare module are activated. Active BEs fail with their active failure rate, disabled BEs with their passive failure rate.

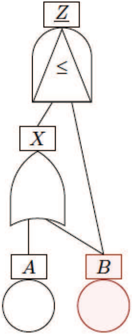
GSPNs for DFTs

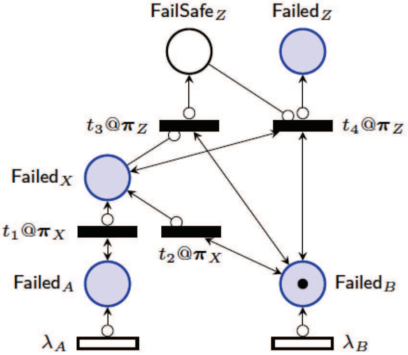
Benchmark	DFT				GSPN		
	#BE	#Dyn.	#Nodes	σ_{max}	#Places	#Timed Trans.	#Immed. Trans.
HECS 5_5_2_np	61	10	107	16	273	122	181
MCS 3_3_3_dp_x	46	21	80	7	246	92	163
RC 15_15_hc	69	33	103	34	376	138	240

σ_{max} is the maximal number of children in the DFT

The size of the GSPN is linearly proportional to the DFT size.
The GSPNs are 2-bounded and have no time traps.

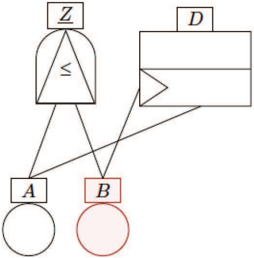
Issue 1: Failure Propagation

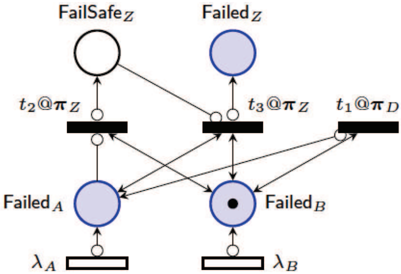




Is B's failure first propagated to gate X, causing PAND Z to fail,
or is B's failure first propagated to gate Z, turning Z fail-safe?

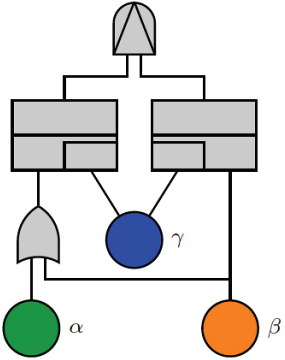
Issue 2: FDEP Failure Forwarding

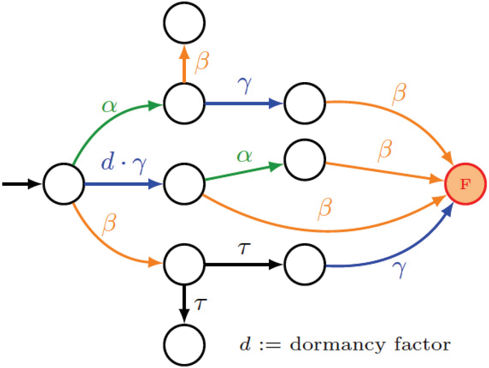




Is B's failure first propagated via D, causing A and Z to fail, or does B's failure first cause Z to become fail-safe before A fails?

Issue 3: Nondeterminism





This phenomenon is called a spare race.

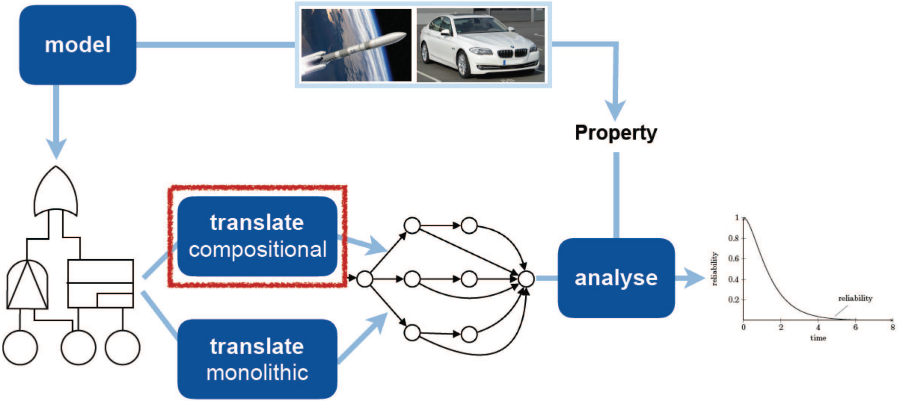
Different Existing DFT Semantics [Volk *et al.*, 2018]

	Monolithic CTMC [11]	IOIMC [12]	Monolithic MA [13]	Orig. GSPN [7]	New GSPN
Tool support	Galileo [17]	DFTCalc [18]	Storm [19]	—	—
Underlying model	CTMC	IMC [15]	MA [14]	GSPN/CTMC [5, 6]	GSPN/MA [16]
Priority gates	$\leq$	$<$	$\leq$	$<$	$\leq$ and $<$
Nested spares	not supported	late claiming	early claiming	not supported	early claiming
Failure propagation	bottom-up	arbitrary	bottom-up	arbitrary	bottom-up
FDEP forwarding	first	interleaved	last	interleaved	first
Non-determinism	uniform	true (everywhere)	true FDEP	uniform	true (PAND, SPARE)

All these different semantics are small twists of the GSPN mapping.

Only the priority mechanism and treatment of nondeterminacy differs.

Graphical Overview



Probabilistic Bisimulation [Larsen & Skou, 1989]

Intuition: transition probabilities for each equivalence class coincide.

Consider a DTMC with state space S and equivalence $R \subseteq S \times S$.
R is a probabilistic bisimulation[2] on S if for any $(s, t) \in R$:

$$L(s) = L(t) \quad \text{and} \quad \mathbf{P}(s, C) = \mathbf{P}(t, C) \text{ for each } C \in S/R$$

where $\mathbf{P}(s, C) = \sum_{s' \in C} \mathbf{P}(s, s')$.

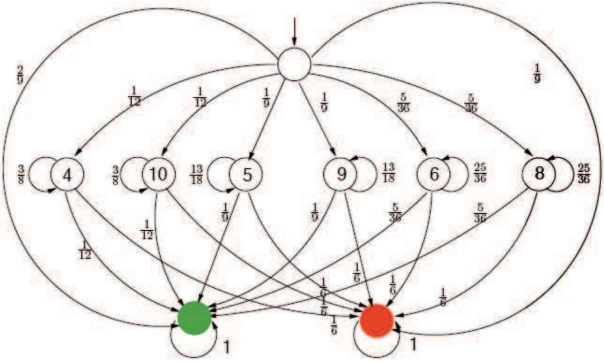
Let $\sim$ denote the largest possible probabilistic bisimulation.

Variants: weak, divergence-sensitive, distribution-based, for CTMC, MDPs, etc.

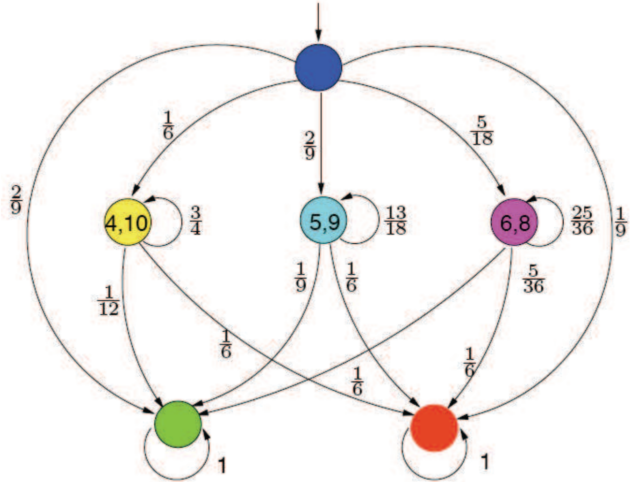
[2] Lumping.

Craps

- Come-out roll:
  - 7 or 11: win
  - 2, 3, or 12: lose
  - else: roll again
- Next roll(s):
  - 7: lose
  - point: win
  - else: roll again



Craps's Bisimulation Quotient



Properties

Quotienting: using partition-refinement in $\mathcal{O}(|\mathbf{P}| \cdot \log |S|)$

Preservation: all relevant measures-of-interest

Congruence: with respect to parallel composition

$$\mathcal{M} \sim \mathcal{N} \quad \text{implies} \quad \mathcal{M} \,\|\, \mathcal{M}' \sim \mathcal{N} \,\|\, \mathcal{M}'$$

Stuttering: weak variants treat internal transitions in special way

Savings: potentially exponentially in time and space

Compositional Minimisation [Hermanns and K., 2000]

- Assume system is given by:

$$\mathcal{M}_1 \,\|\, \ldots \,\|\, \mathcal{M}_i \,\|\, \ldots \,\|\, \mathcal{M}_k$$

  with $\mathcal{M}_j$ a Markov model and parallel composition $\|$

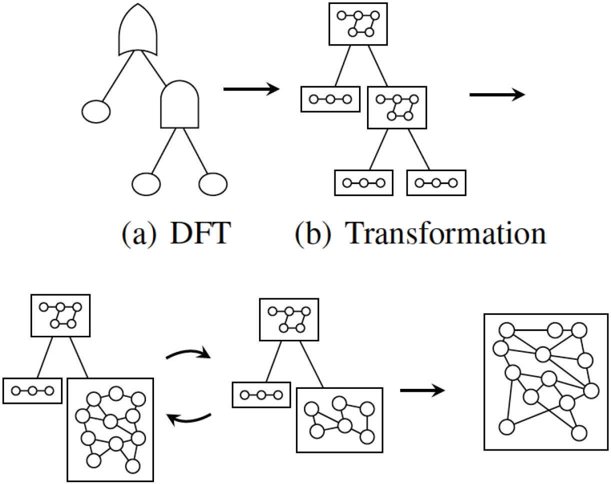
- Recall congruence property:
  $\mathcal{M} \sim \mathcal{N}$ implies $\mathcal{M} \,\|\, \mathcal{M}' \sim \mathcal{N} \,\|\, \mathcal{M}'$

- Component-wise minimisation
  1. Pick process $\mathcal{M}_i$ and consider its quotient $\mathcal{M}_i/\!\sim$ under $\sim$
  2. Yielding $\mathcal{M}_1 \,\|\, \ldots \,\|\, \mathcal{M}_i/\!\sim \,\|\, \ldots \,\|\, \mathcal{M}_k$; repeat 1. and 2.
  3. Once all done, minimise pairs $\mathcal{M}_i/\!\sim \,\|\, \mathcal{M}_{i+1}/\!\sim$ etc.

- Finding optimal ordering to minimise is NP-complete

  Ordering by heuristics [Crouzen *et al.*, 2008]

Compositional DFT Minimisation [Crouzen *et al.*, 2010]



(a) DFT (b) Transformation

(c) Composition (d) Minimisation (e) IMC

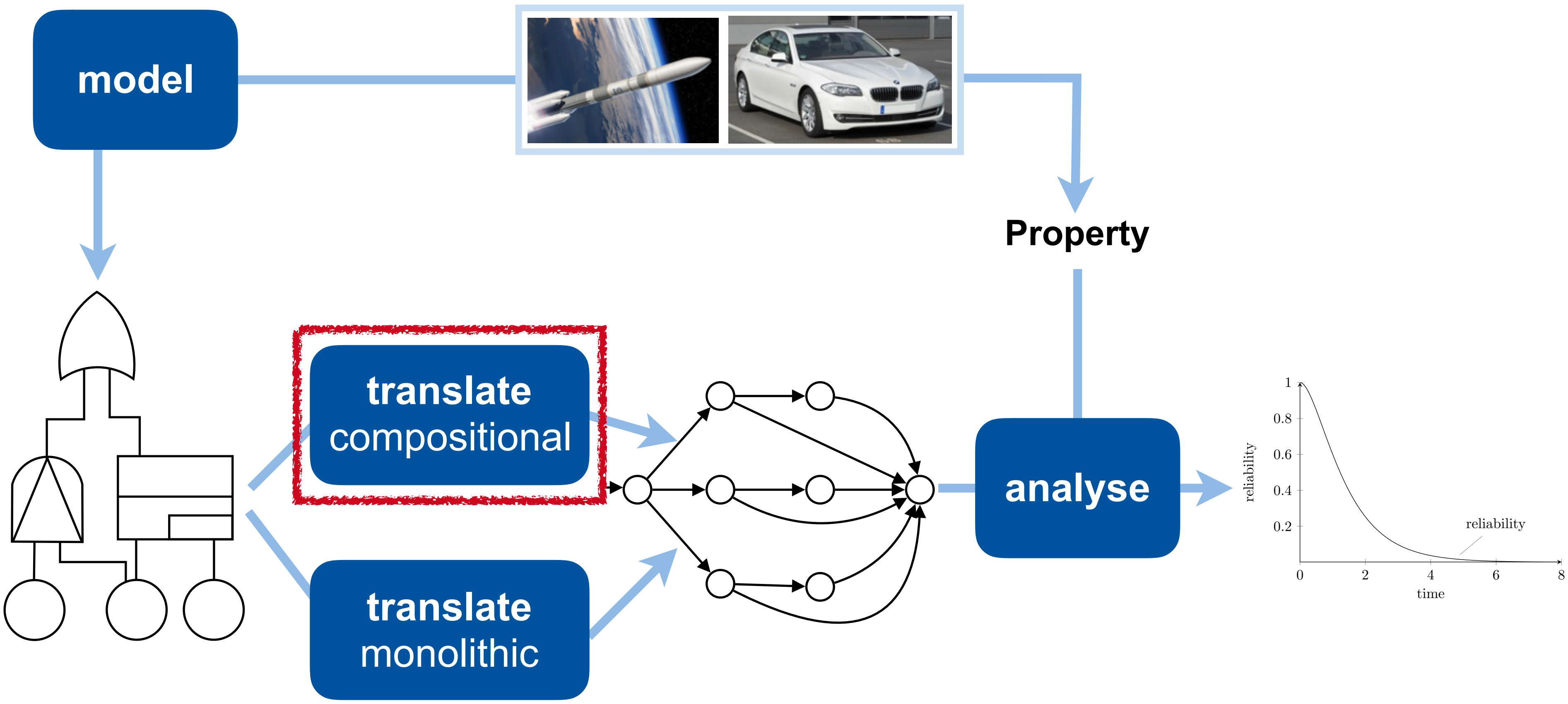
Compositional DFT Minimisation [Crouzen *et al.*, 2010]

case study	peak # states	# transitions	unreliability	time (s)
CPS	4113	24608	.00135	490
CAS	8	10	.65790	1
CAS-PH	x	x	x	x
NDPS	x	x	x	x
FTTP-4	32757	426826	.01922	13111
FTTP-5	MO	MO	MO	MO

CPS	133	465	.00135	67
CAS	36	119	.65790	94
CAS-PH	40052	265442	.112	231
NDPS	61	169	[.00586, .00598]	266
FTTP-4	1325	13642	.01922	65
FTTP-6	11806565	22147378	.00045	1989

Comparing `Galileo DIFTree` (1995, top) to `DFTCalc` (2011, bottom)

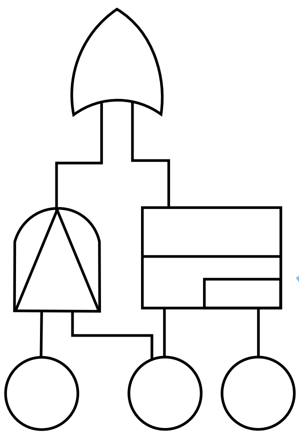
Overview


model
Property
translate compositional
translate monolithic
analyse
reliability
time
1
0.8
0.6
0.4
0.2
0
2
4
6
8

Overview

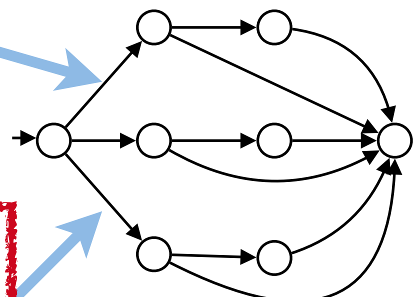
model

Property

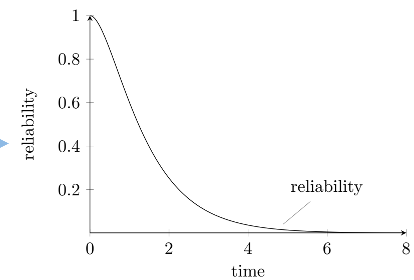


translate
compositional

translate
monolithic



analyse



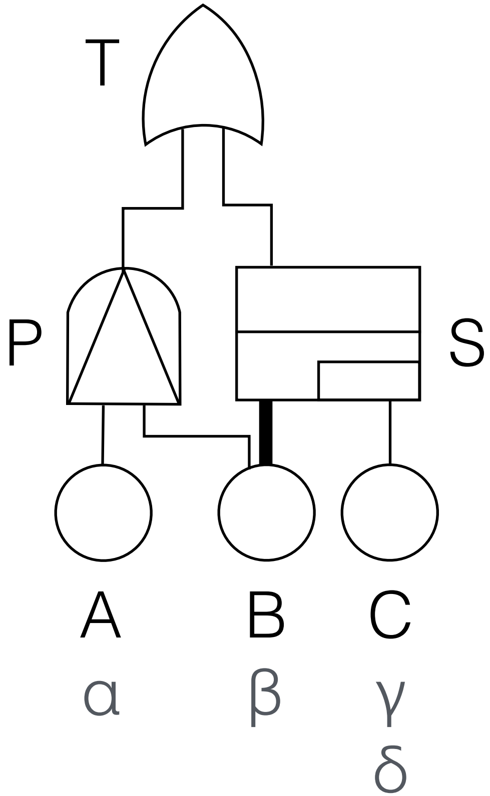
State space exploration

[Sullivan et al, FTCS 1999]
[Volk et al., SAFECOMP 2016]

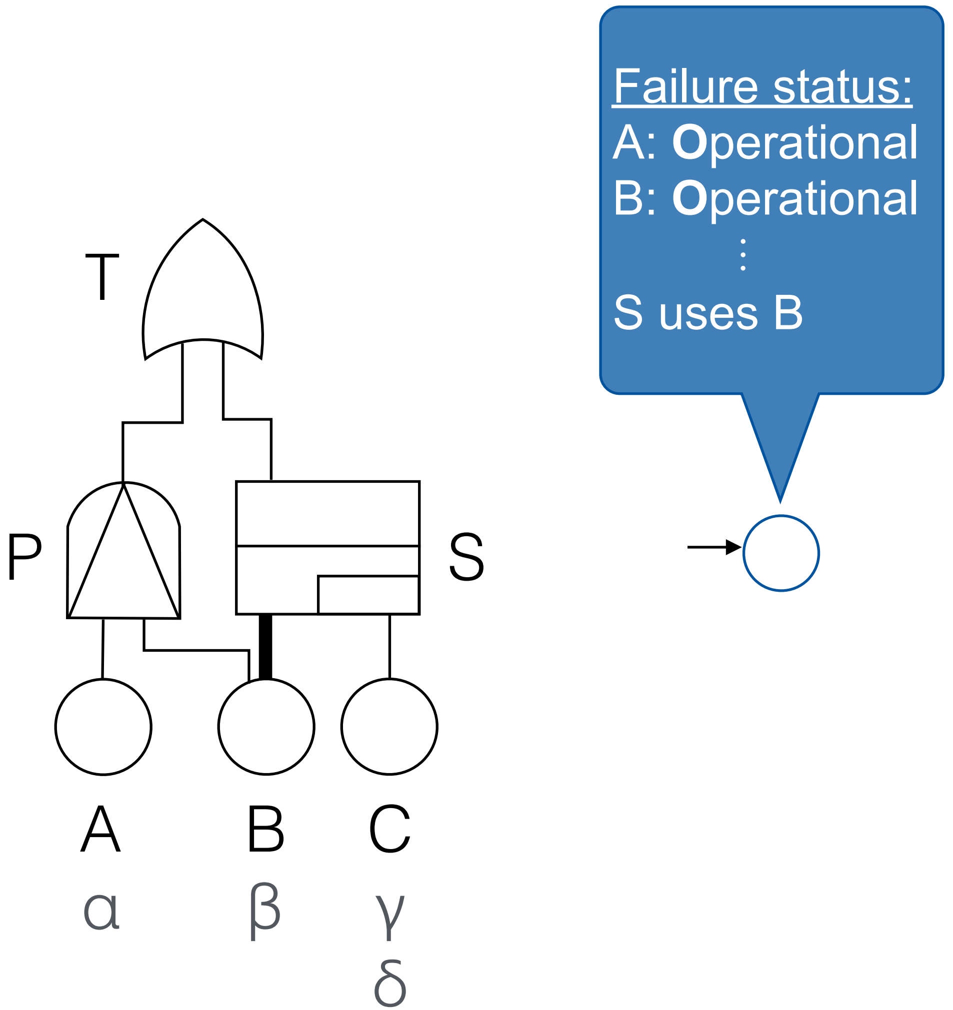
Monolithic idea

- Based on Galileo approach
- Generate states of Markov chain iteratively:
  1. Let basic element in DFT fail
  2. Propagate failure through the DFT
  3. Resulting failure status of DFT is new state in Markov chain

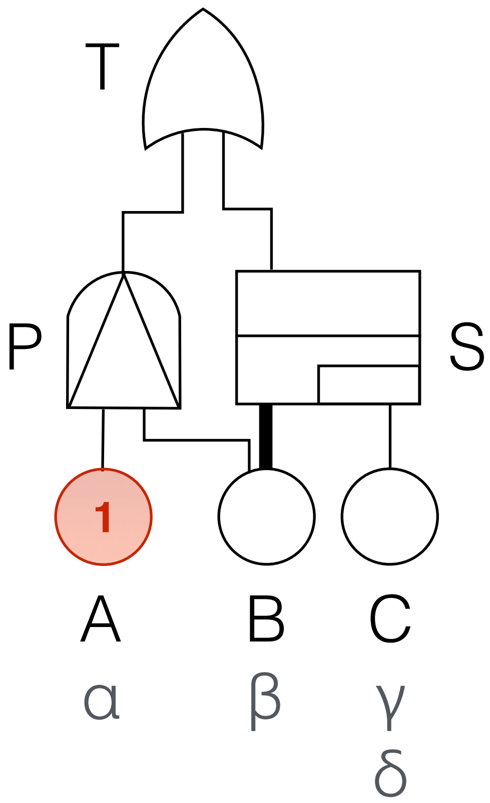
State space exploration

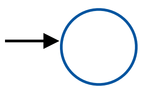


State space exploration

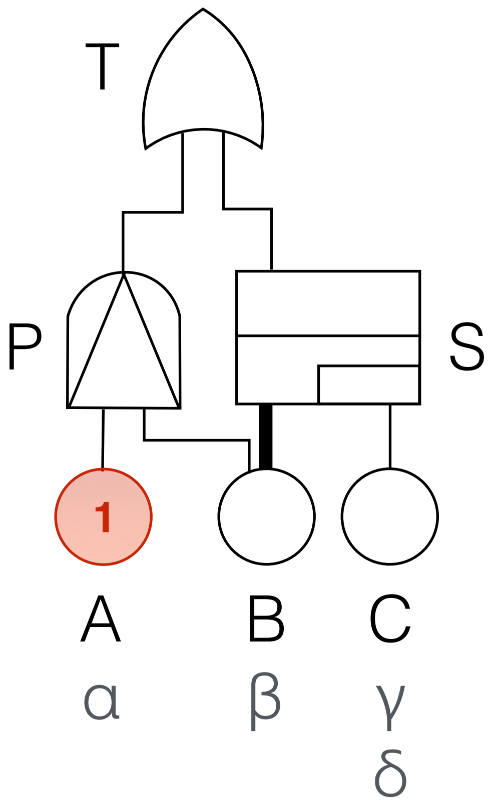


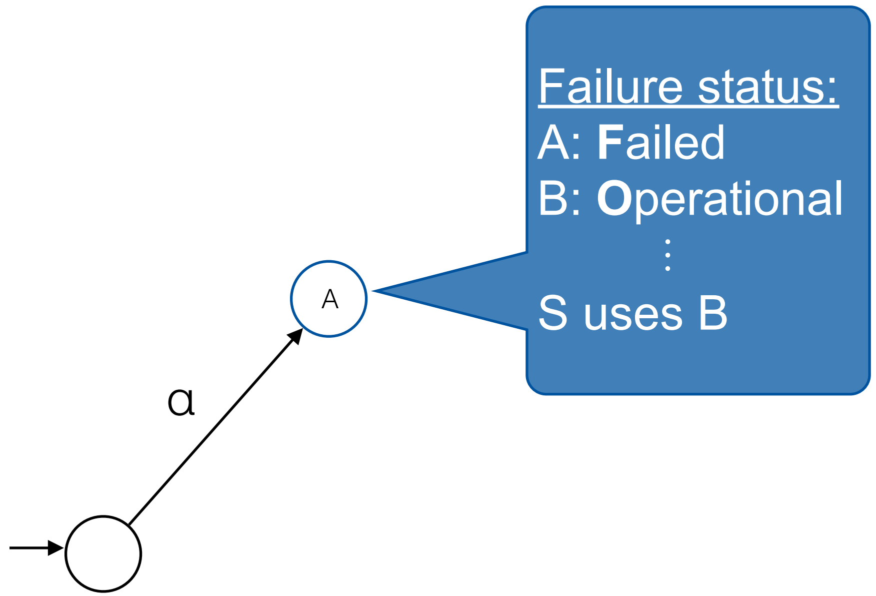
State space exploration



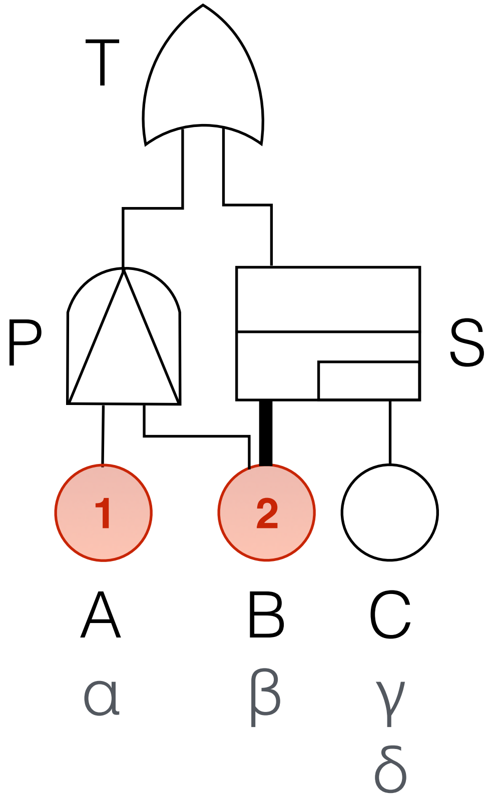


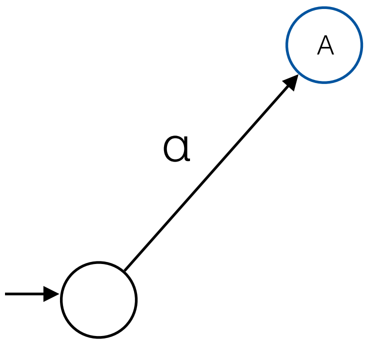
State space exploration



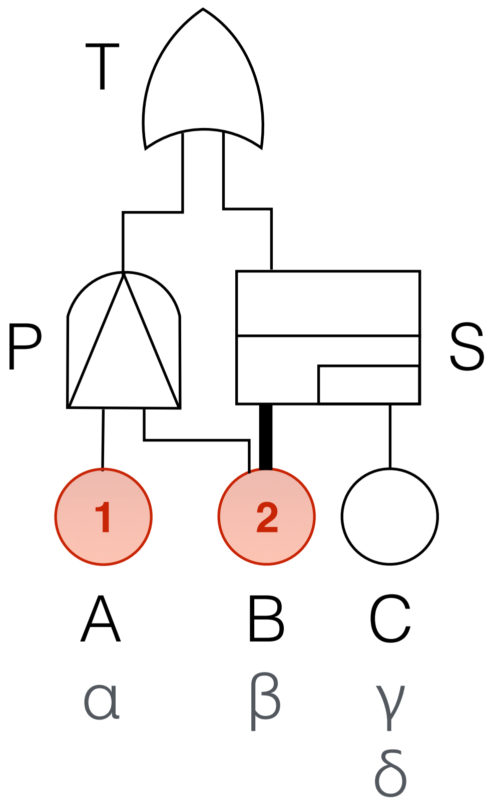


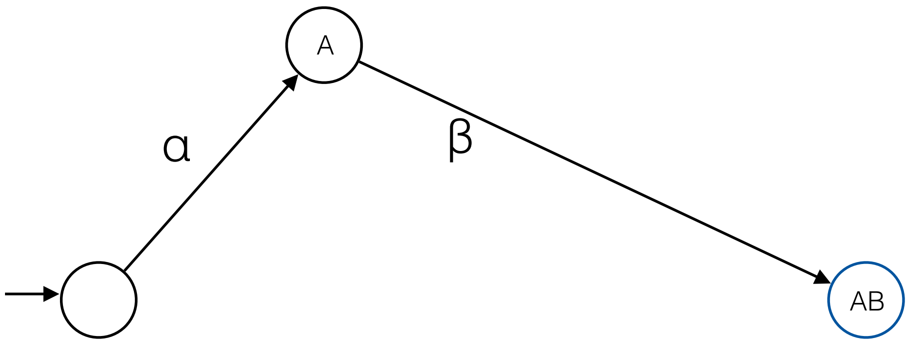
State space exploration



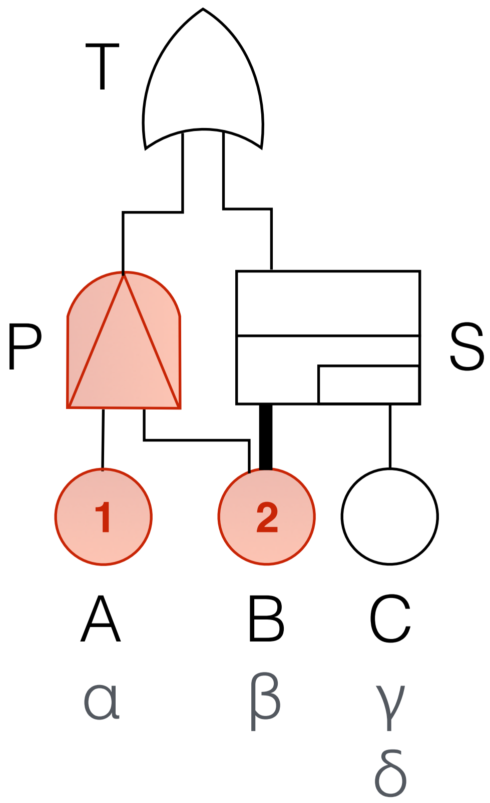


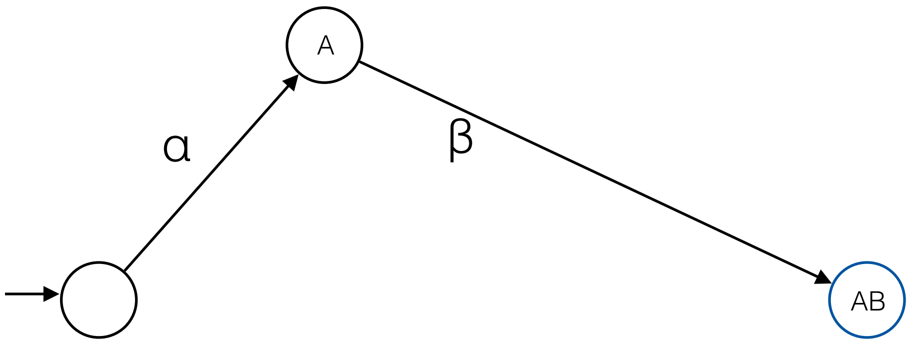
State space exploration



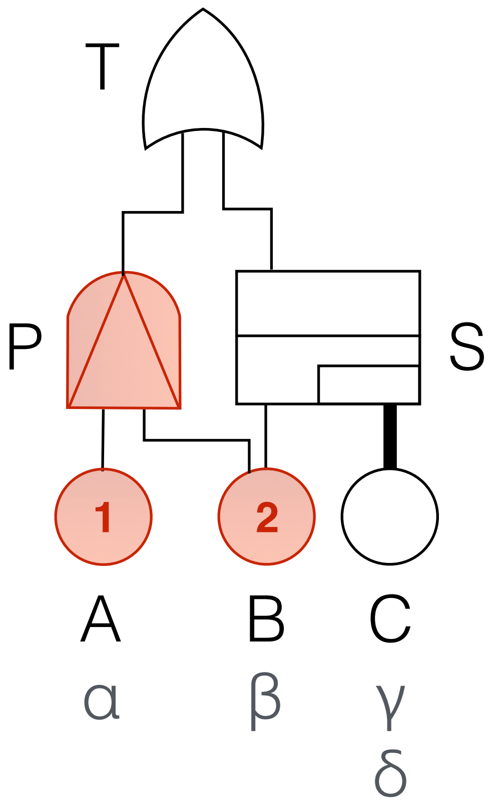


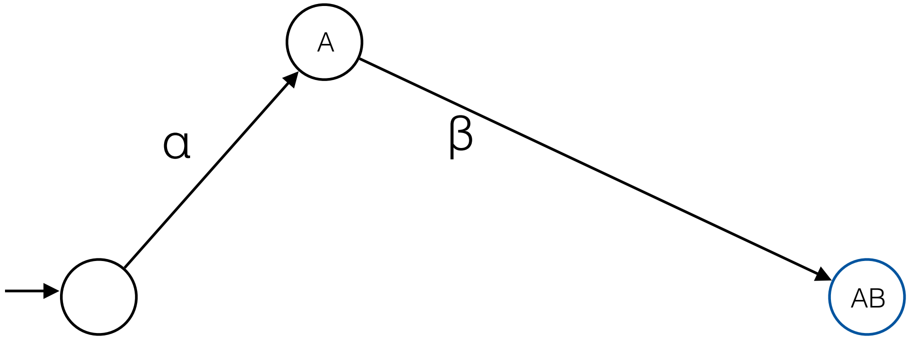
State space exploration



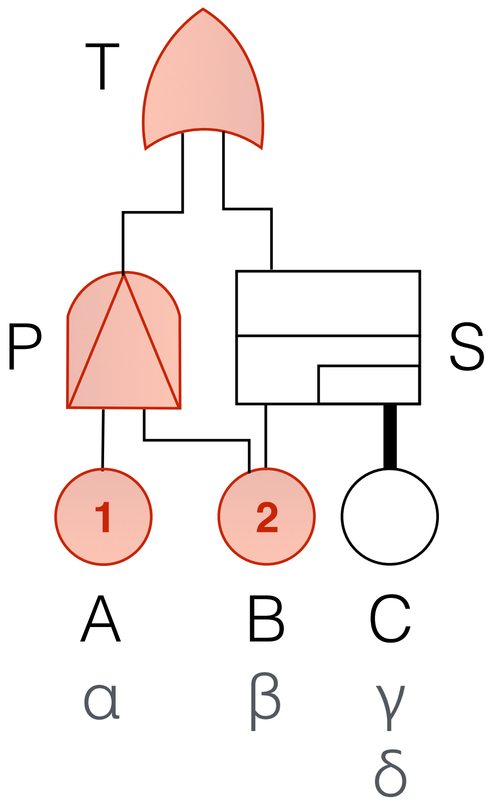


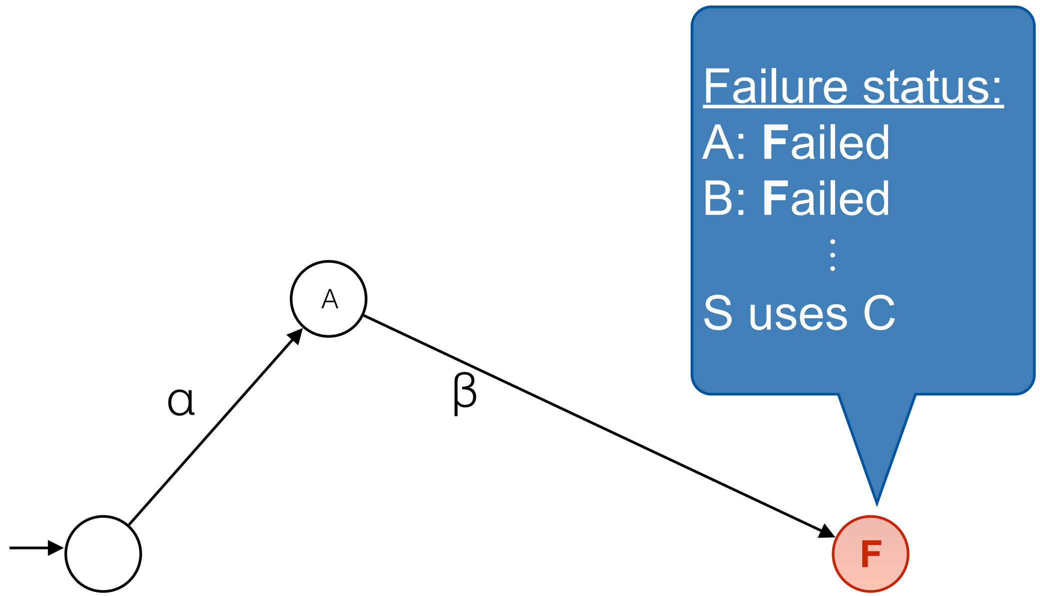
State space exploration



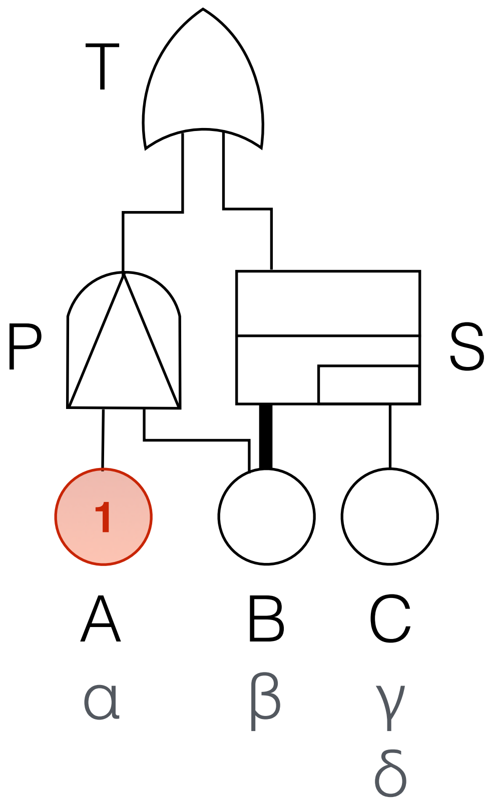


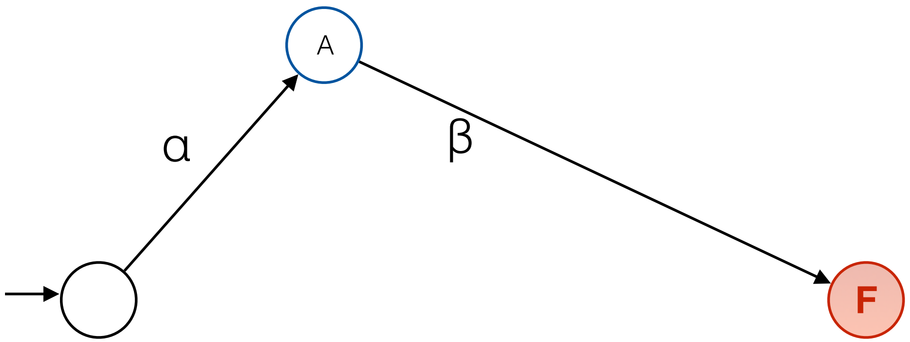
State space exploration



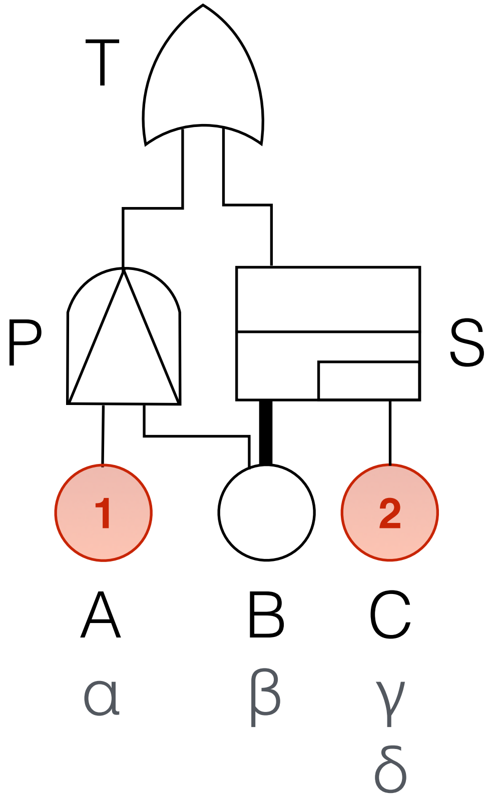


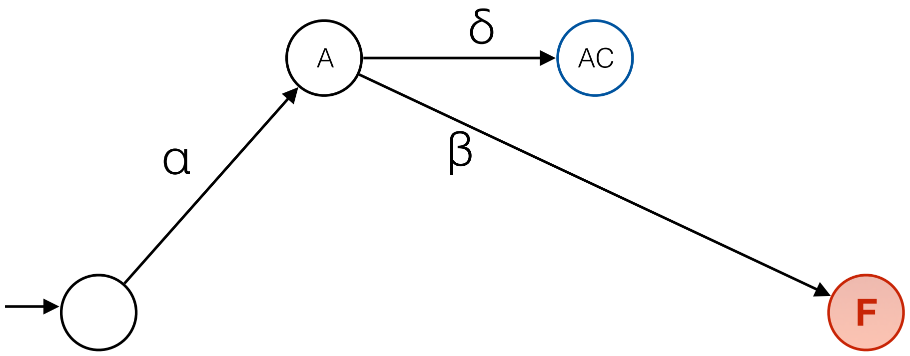
State space exploration



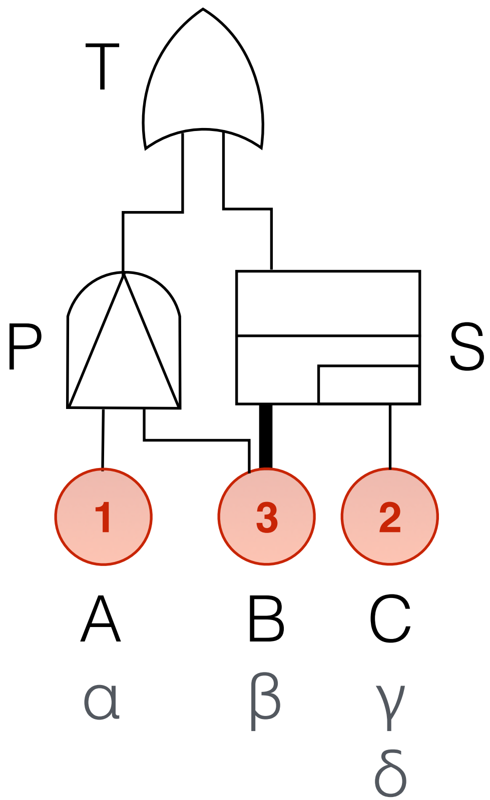


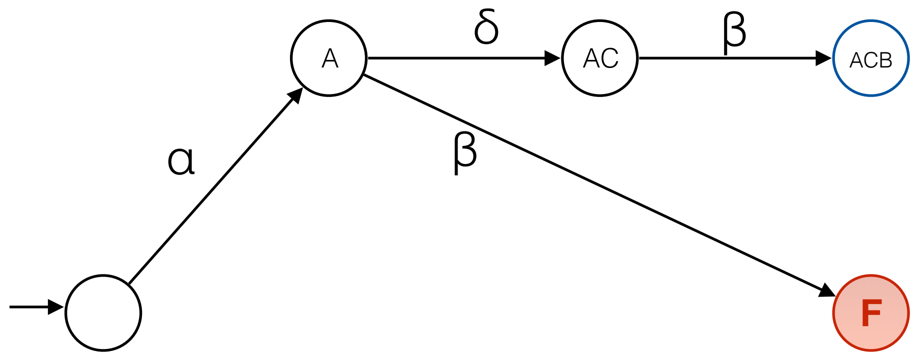
State space exploration



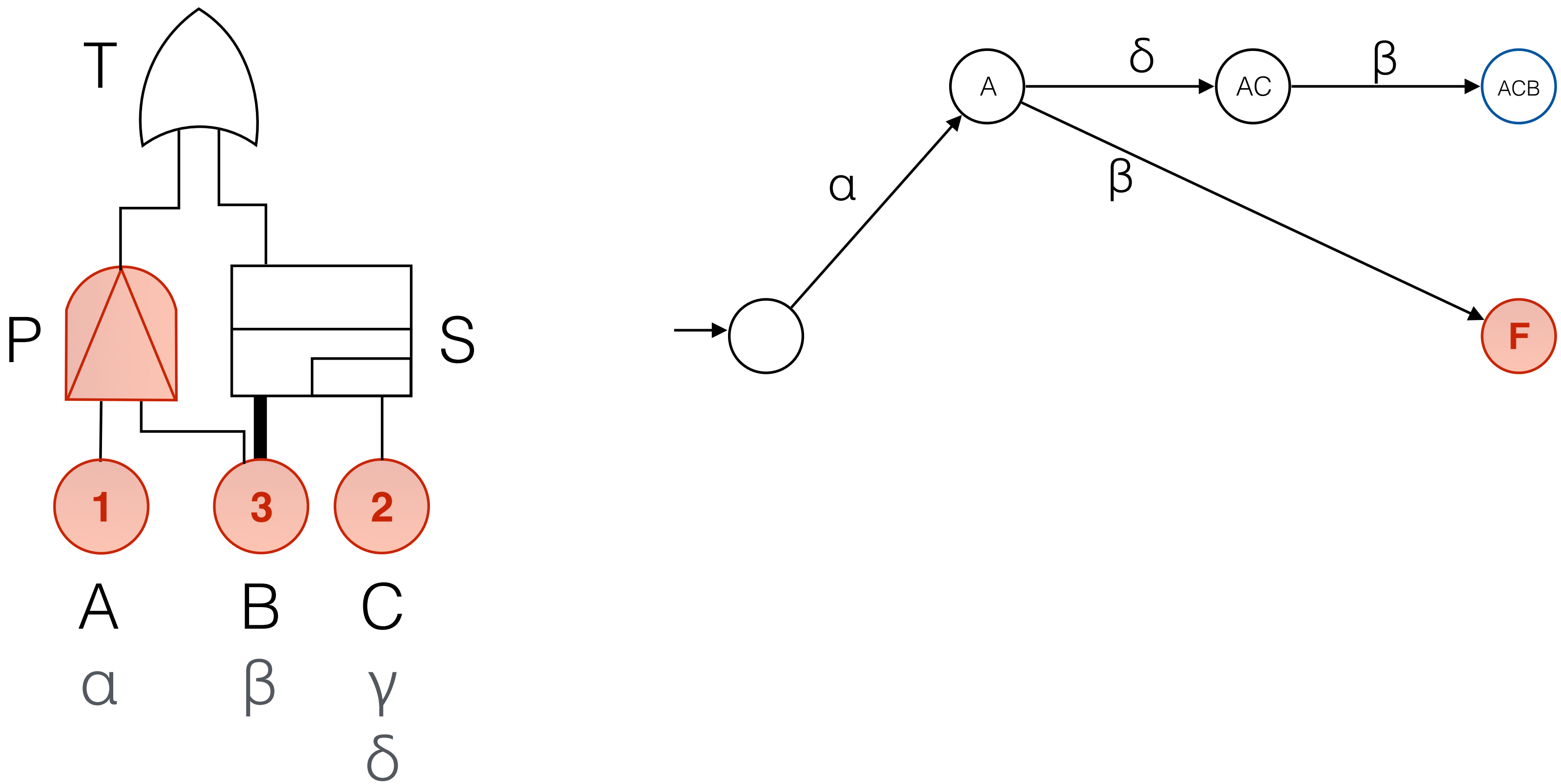


State space exploration

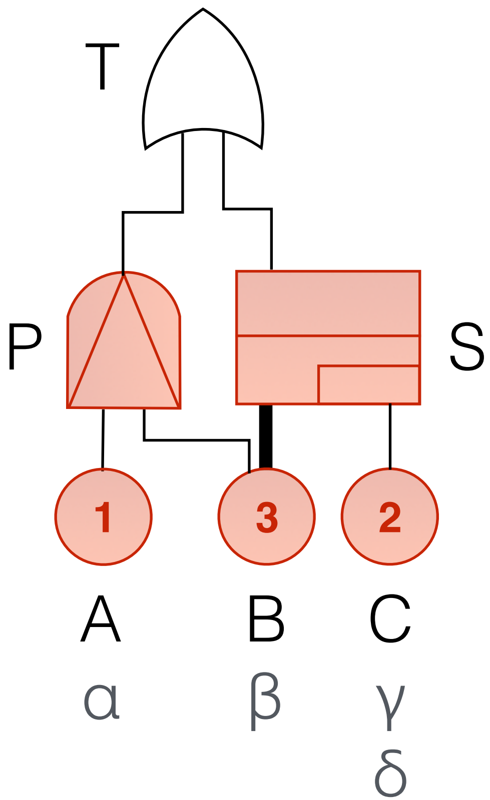


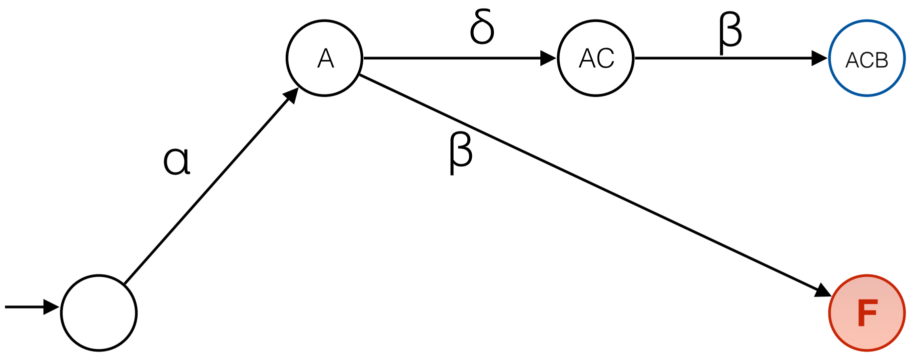


State space exploration

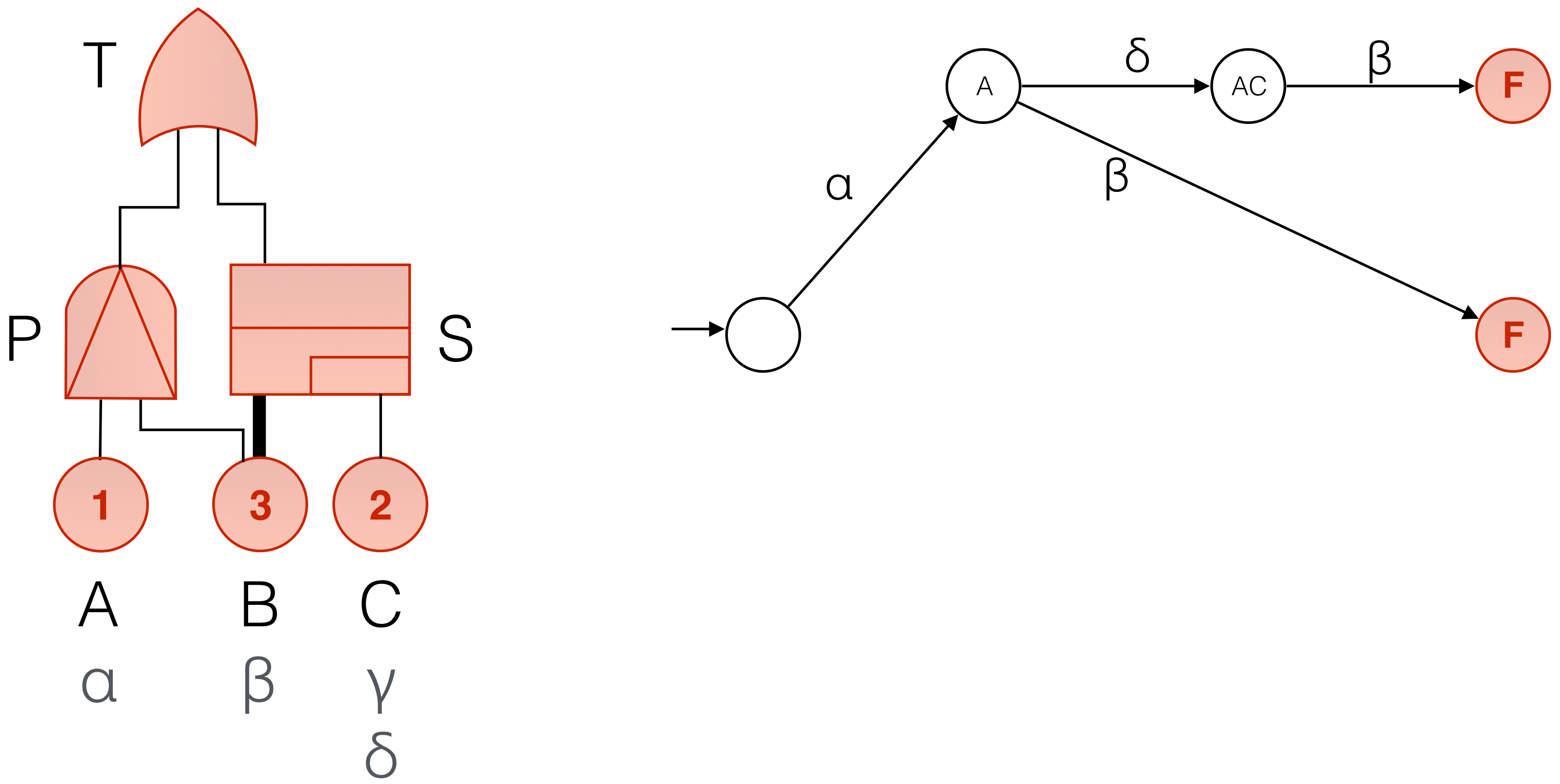


State space exploration

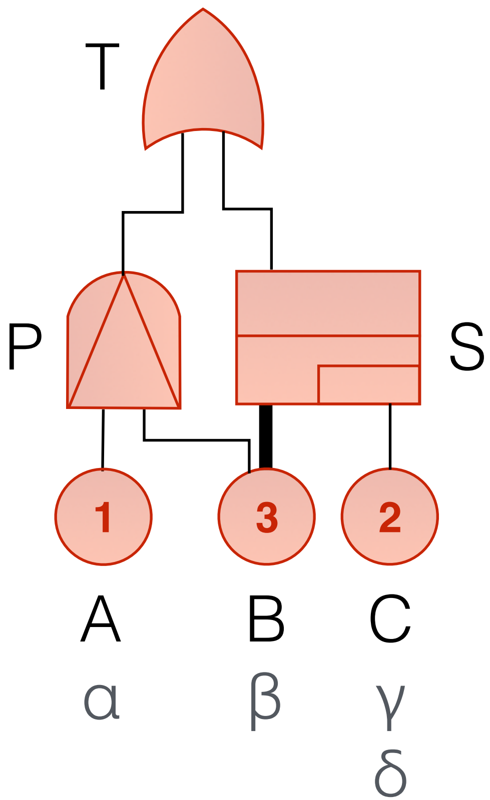


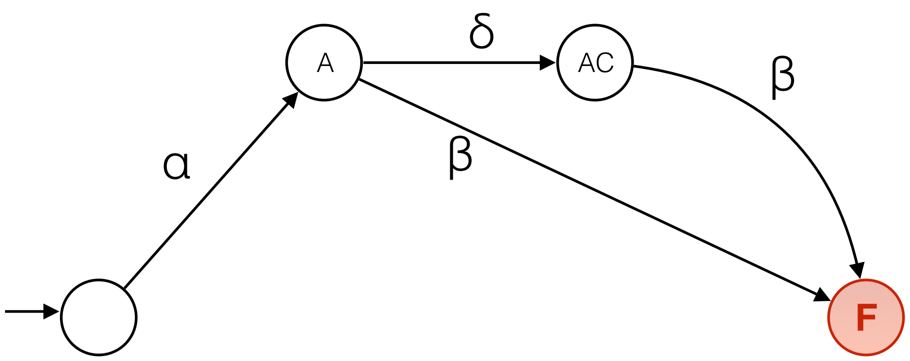


State space exploration

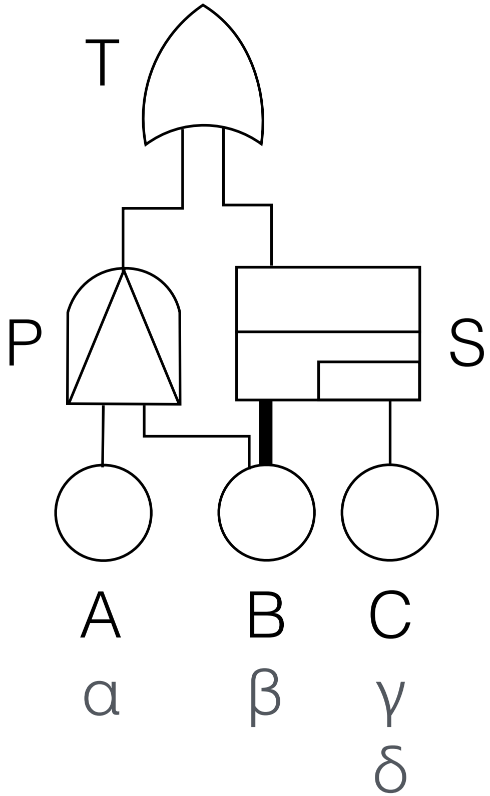


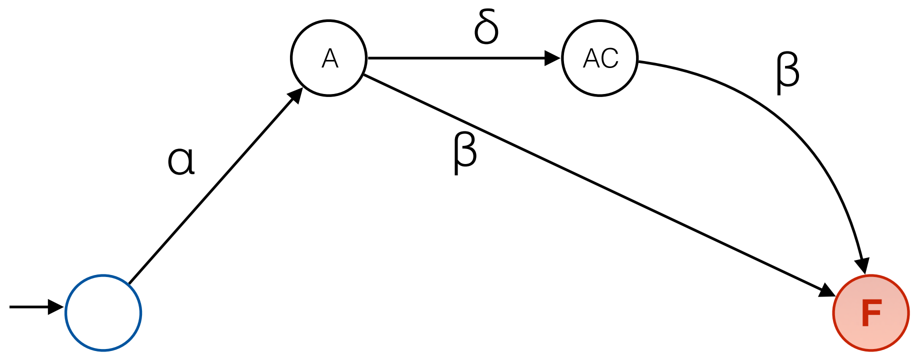
State space exploration



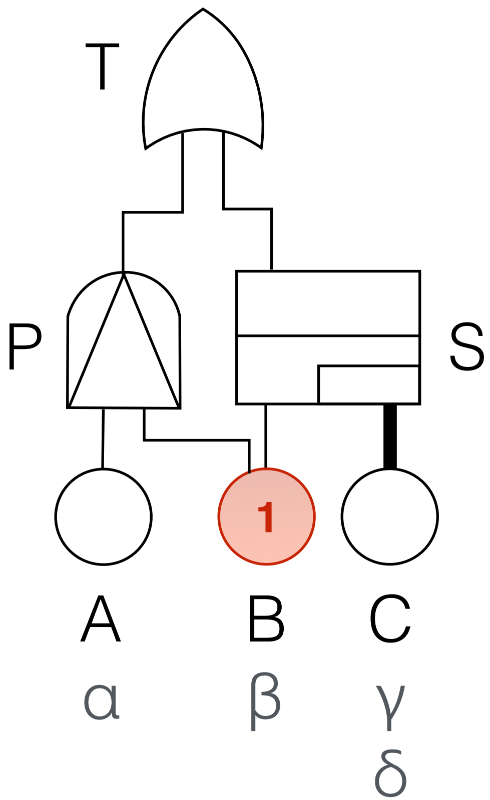


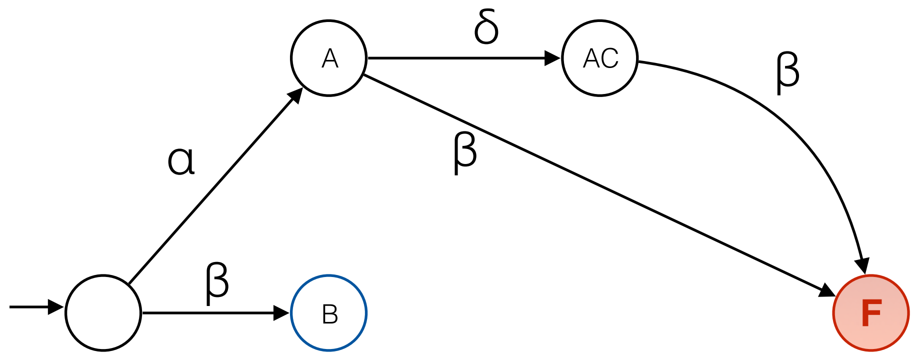
State space exploration



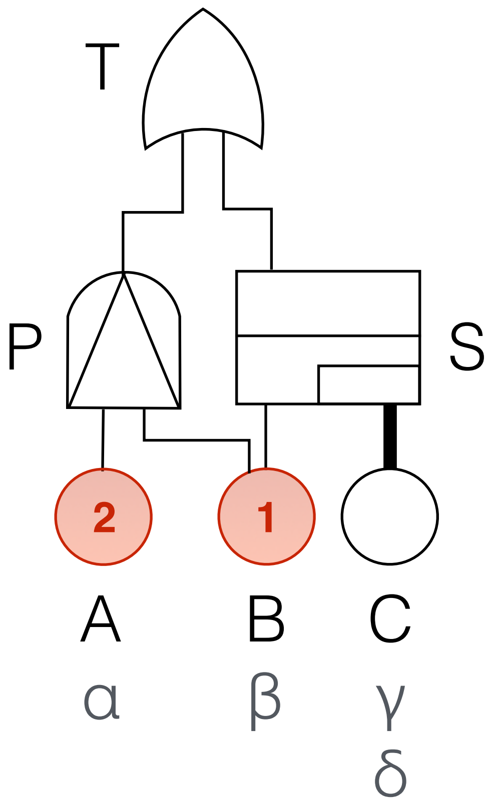


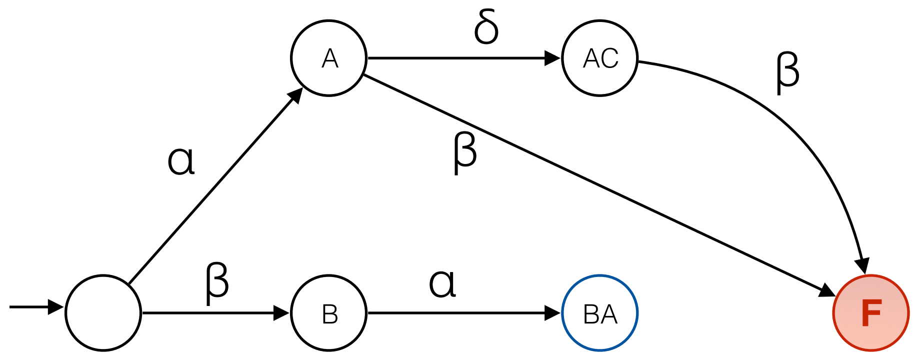
State space exploration



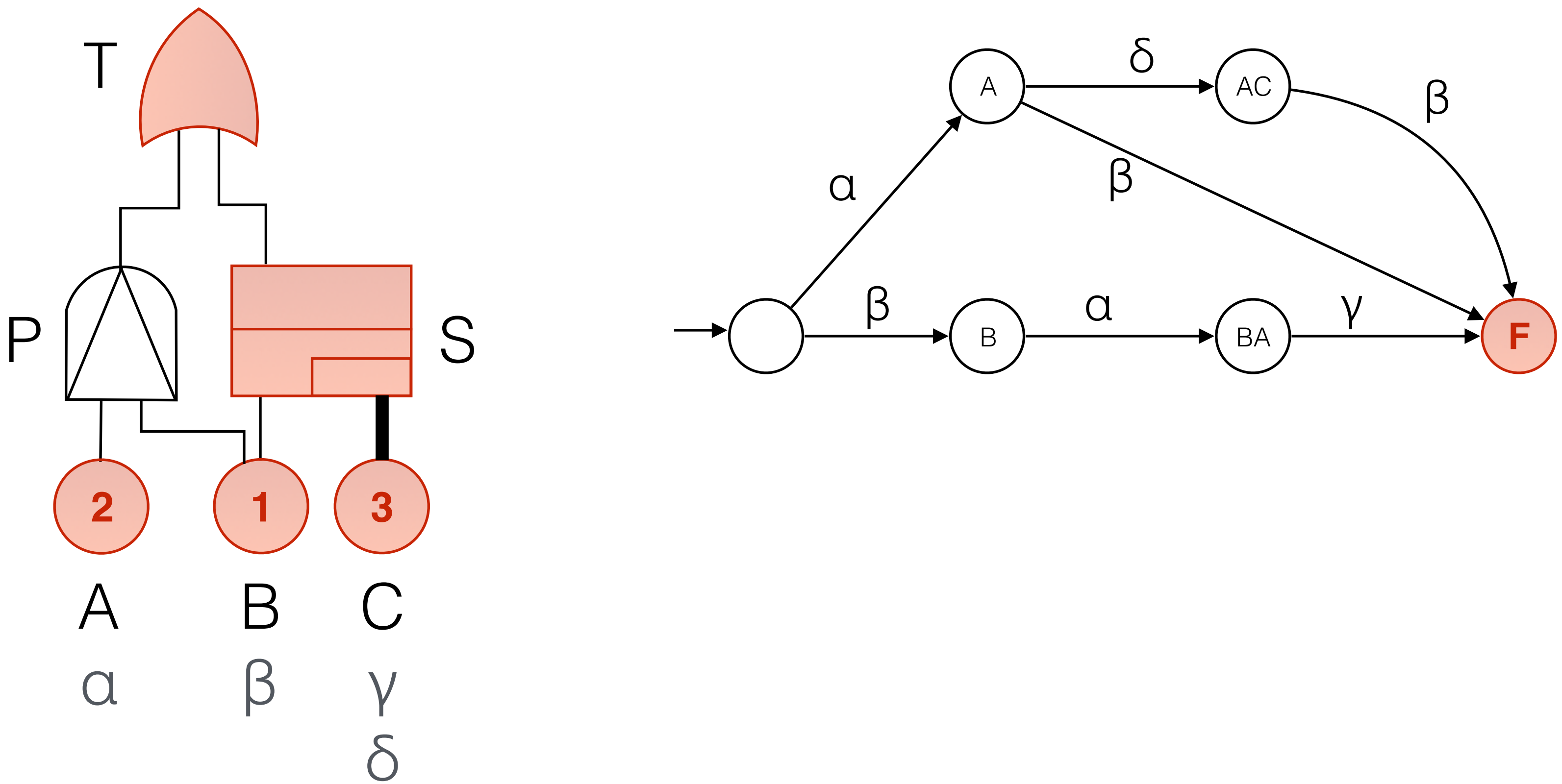


State space exploration

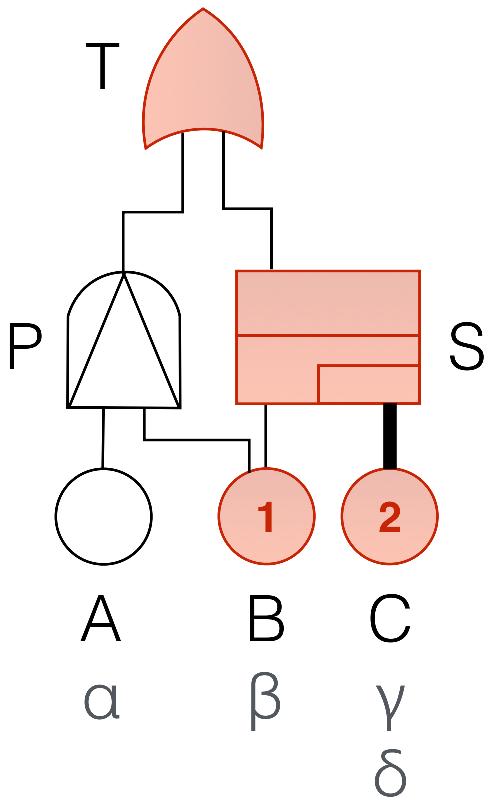


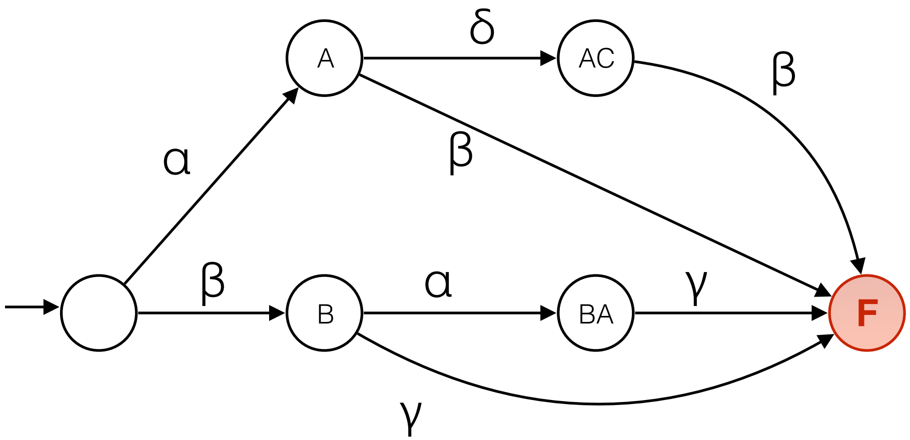


State space exploration

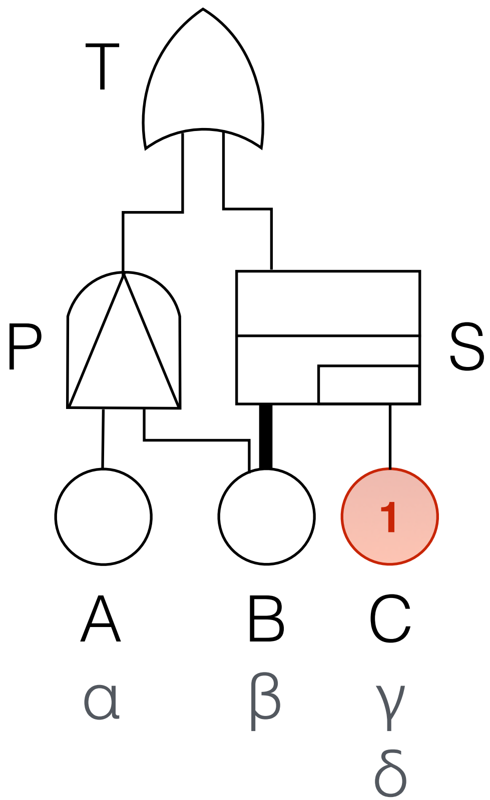


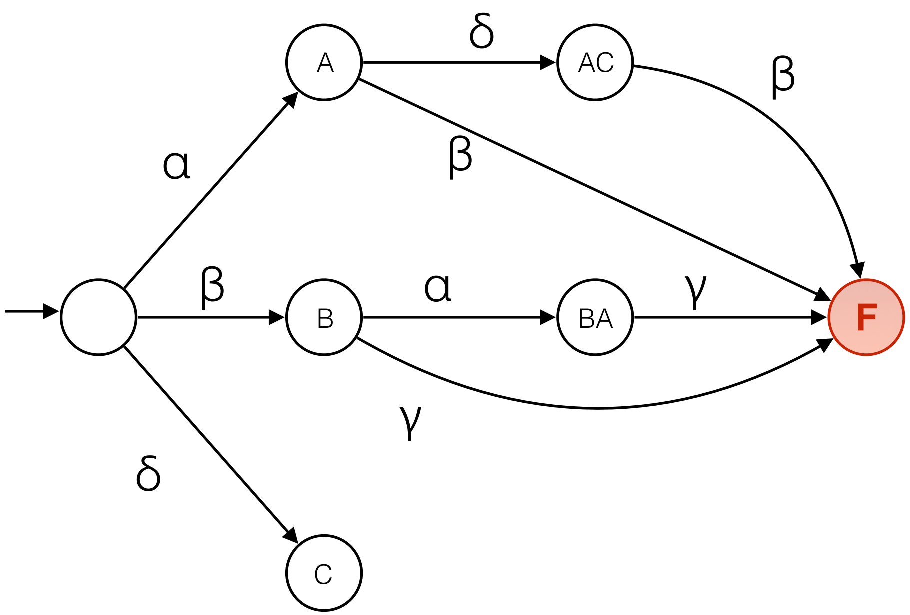
State space exploration



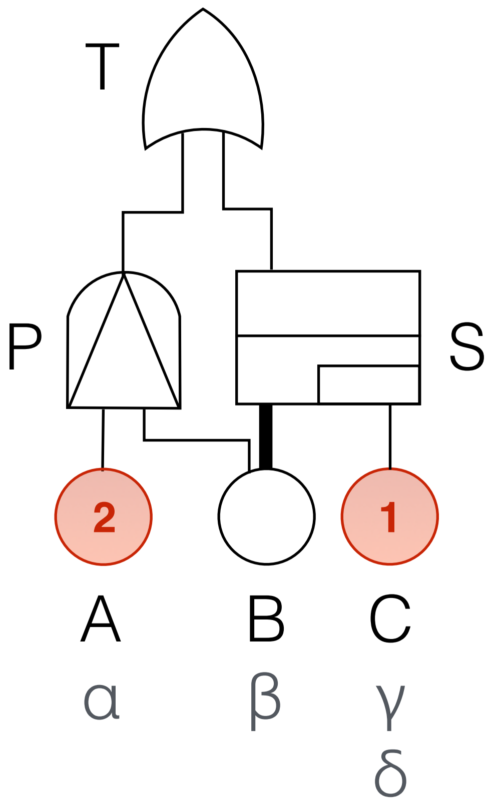


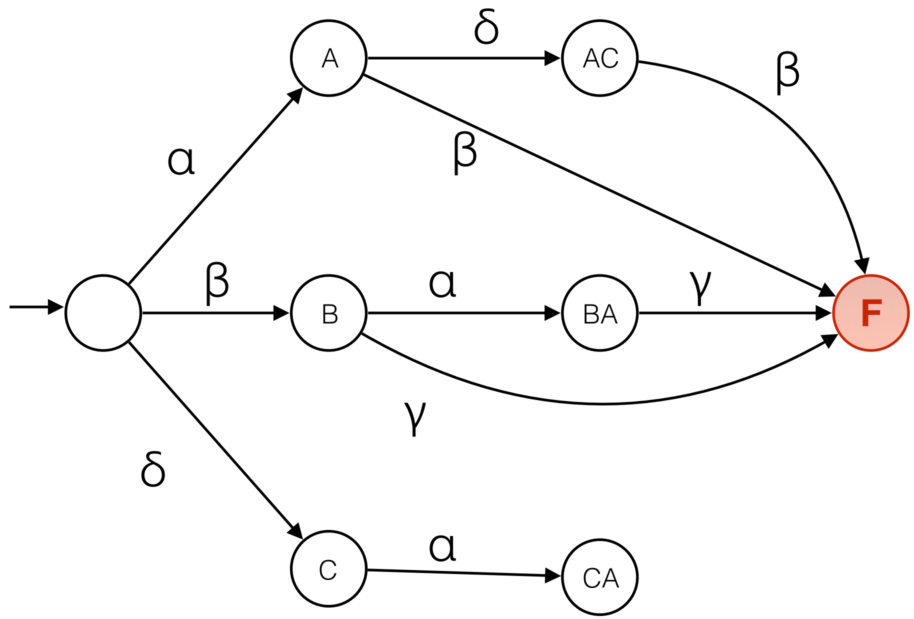
State space exploration



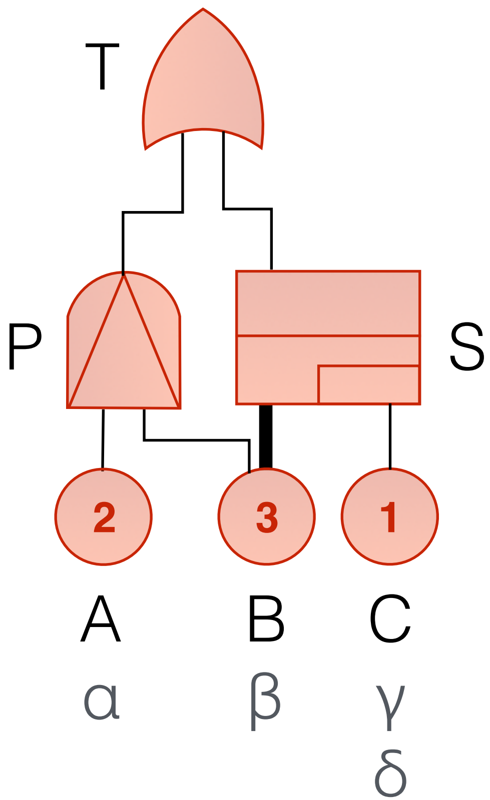


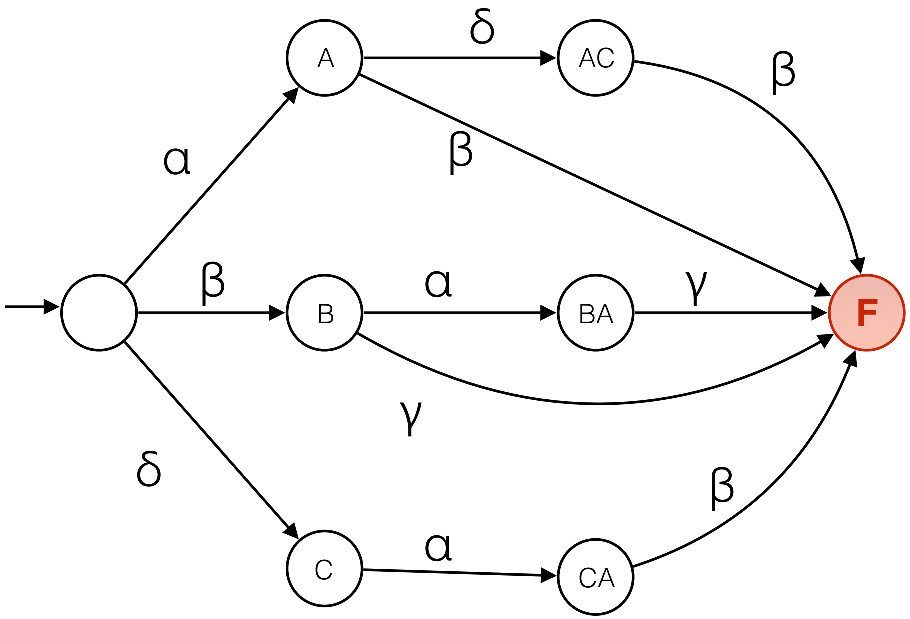
State space exploration



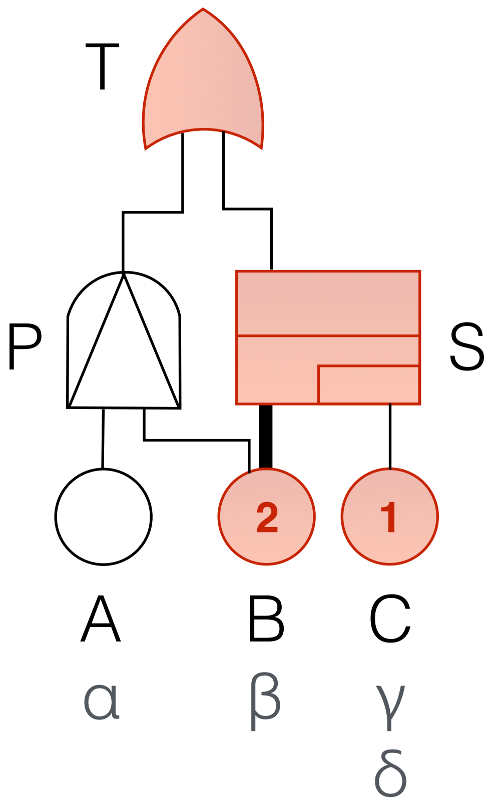


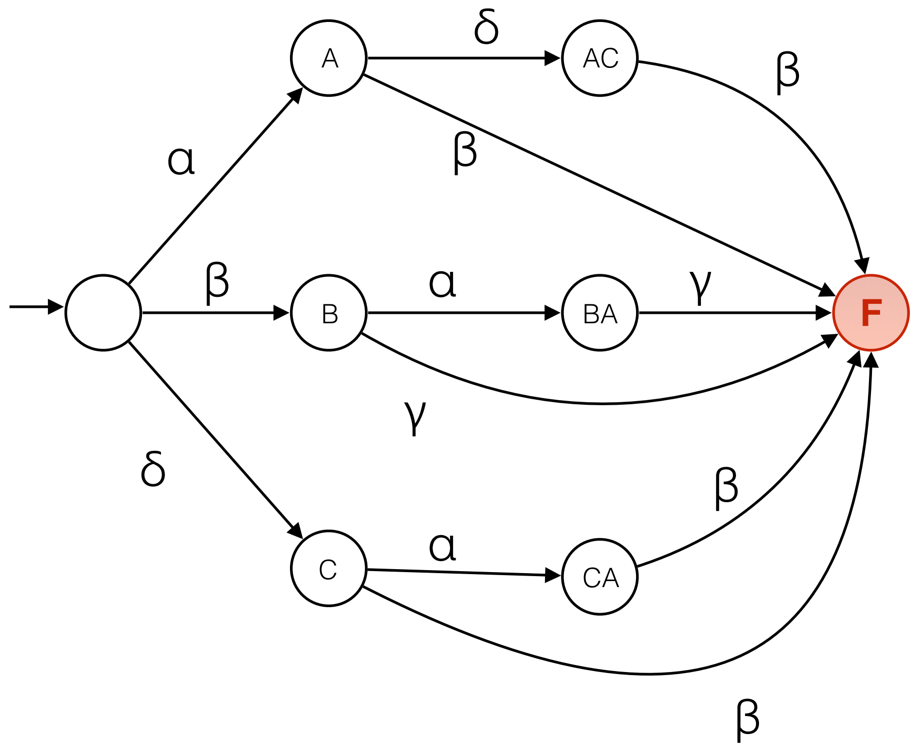
State space exploration



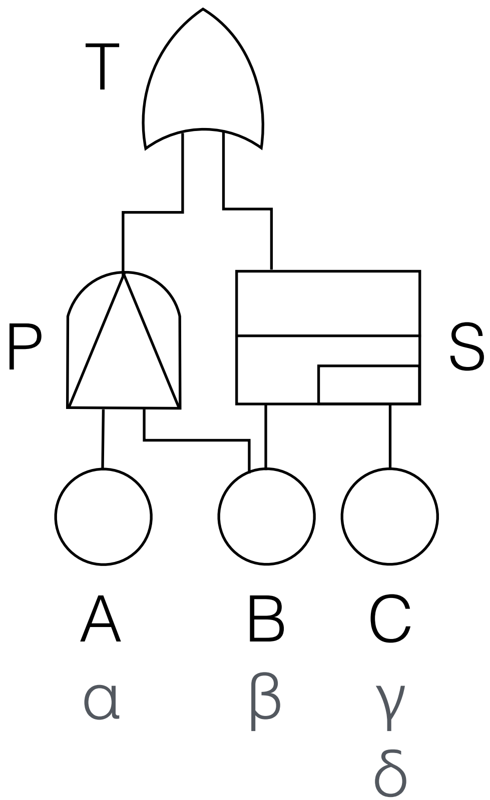


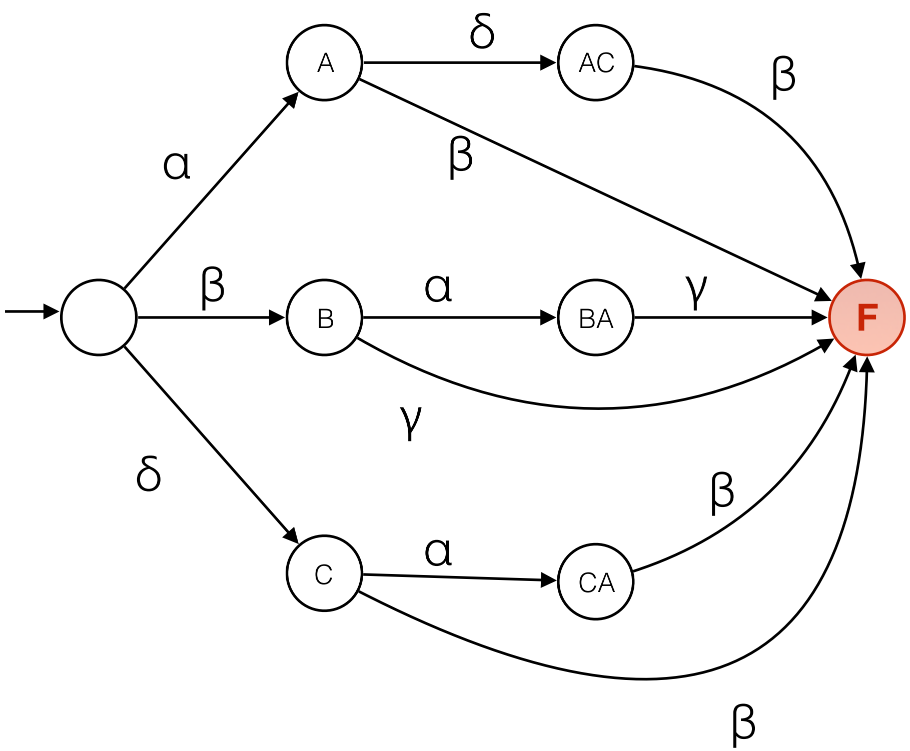
State space exploration



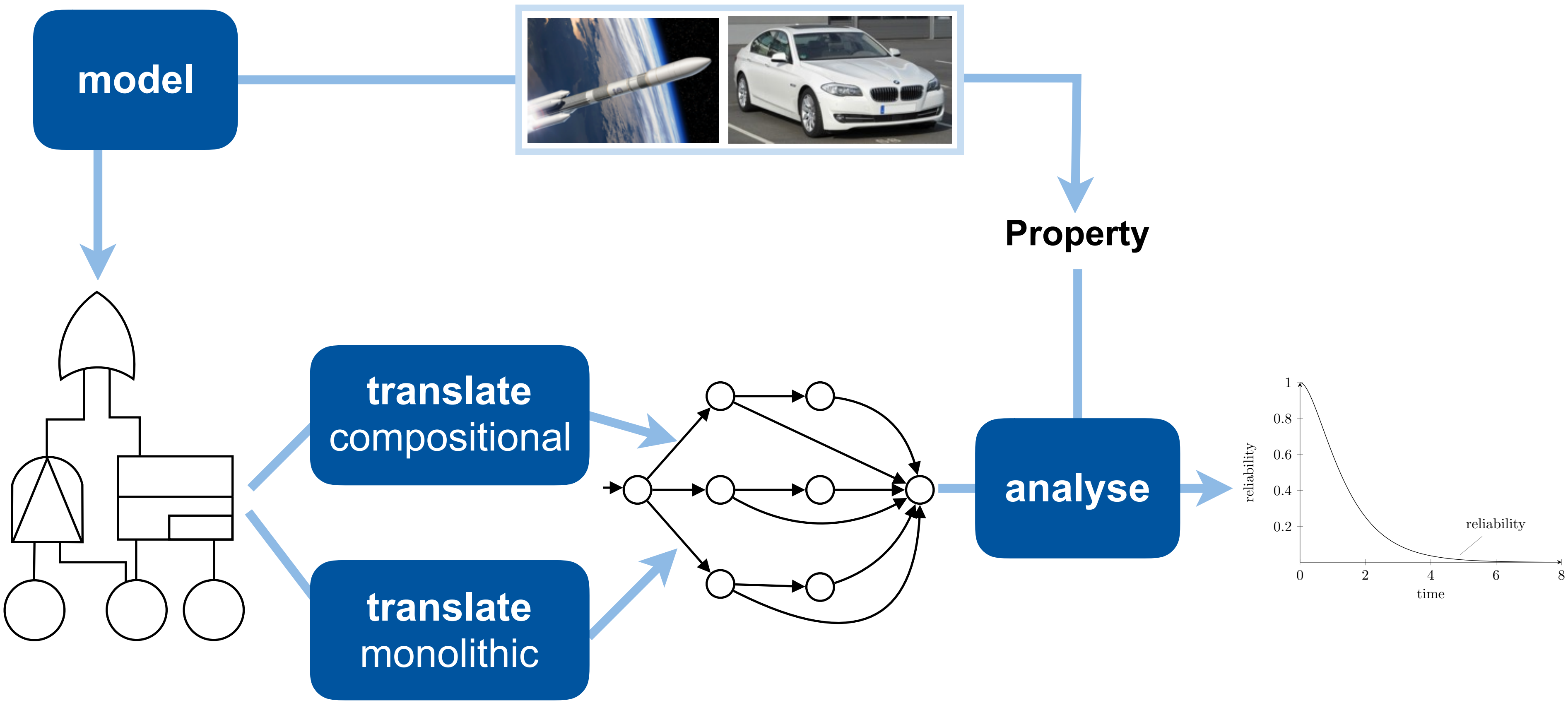


State space exploration





Overview


model
Property
translate compositional
translate monolithic
analyse
1
0.8
0.6
0.4
0.2
reliability
reliability
0
2
4
6
8
time

State space reduction techniques

Problem:
exponential state space size

Solution:
apply **reduction techniques**

State space reduction techniques

[Volk et al., IEEE TII 2018]

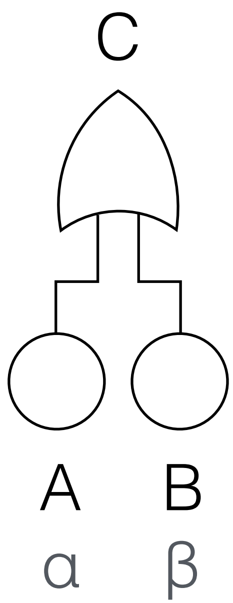
Solution:
apply **reduction techniques**

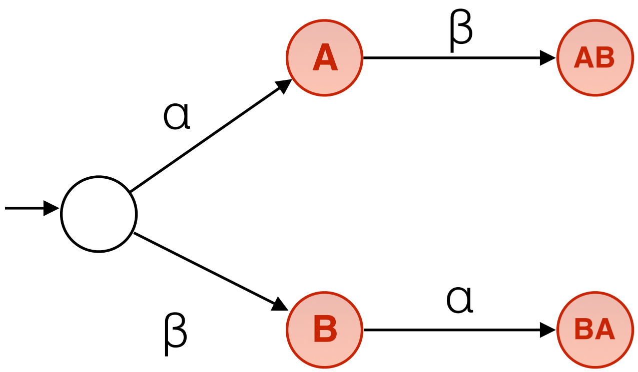
Reduction techniques:

- Bisimulation minimisation:
  - existing reduction technique for Markov chains
- Don't care propagation:
  - only consider failures making a difference
- Symmetry reduction:
  - exploit symmetric structures
- Modularisation:
  - compositional analysis for reliability

Don't care propagation

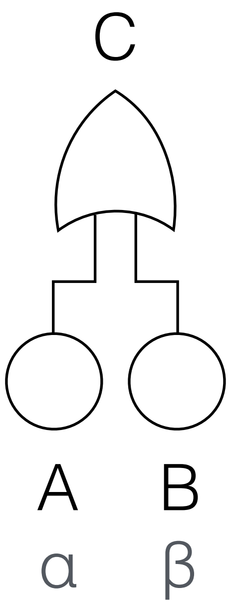
only consider failures making a difference

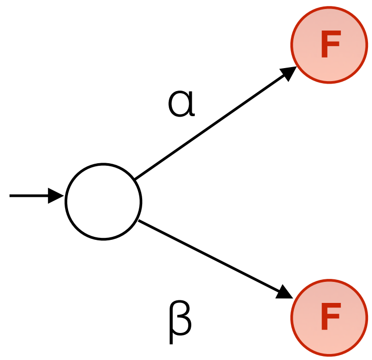




Don’t care propagation

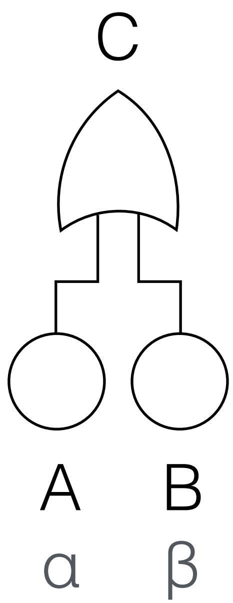
only consider failures making a difference

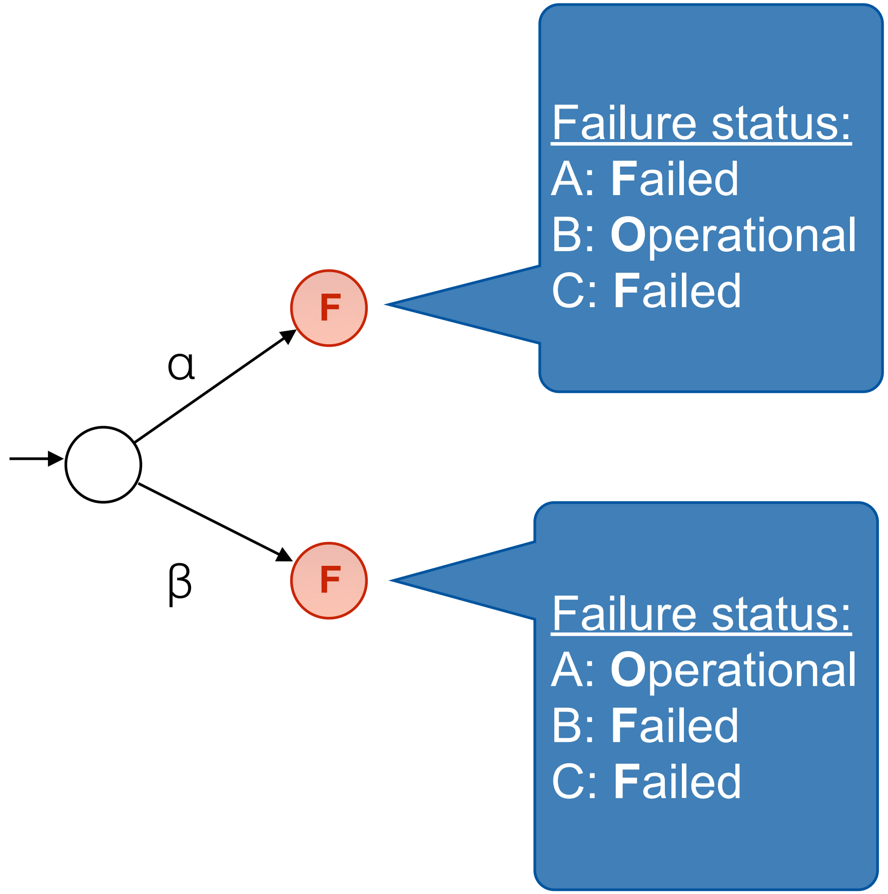




Don’t care propagation

only consider failures making a difference

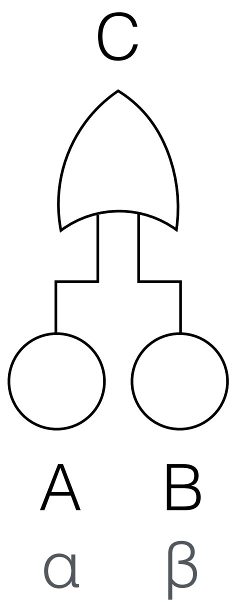


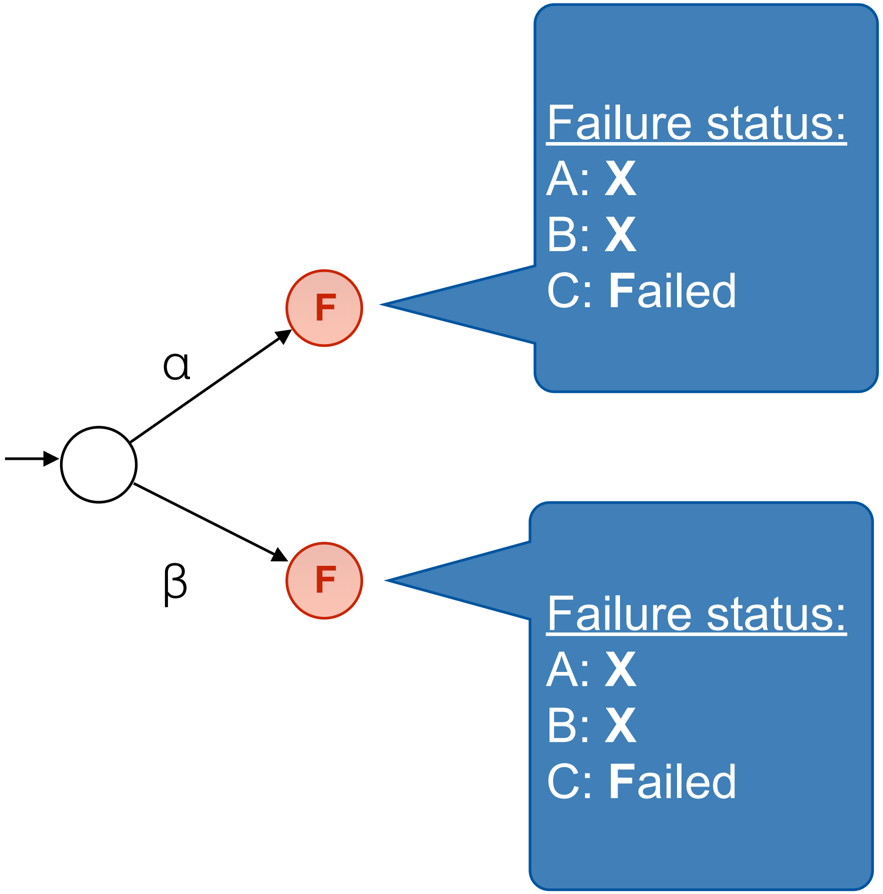


Research Training Group – Uncertainty and Randomness in Algorithms, Verification, and Logic
UnRAVeL
RWTH AACHEN UNIVERSITY

Don’t care propagation

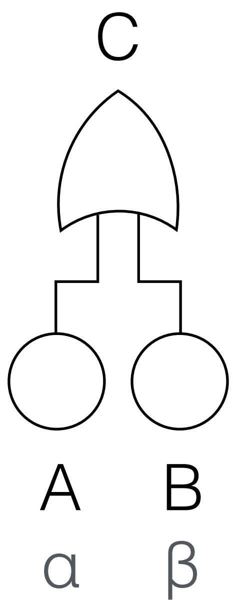
only consider failures making a difference

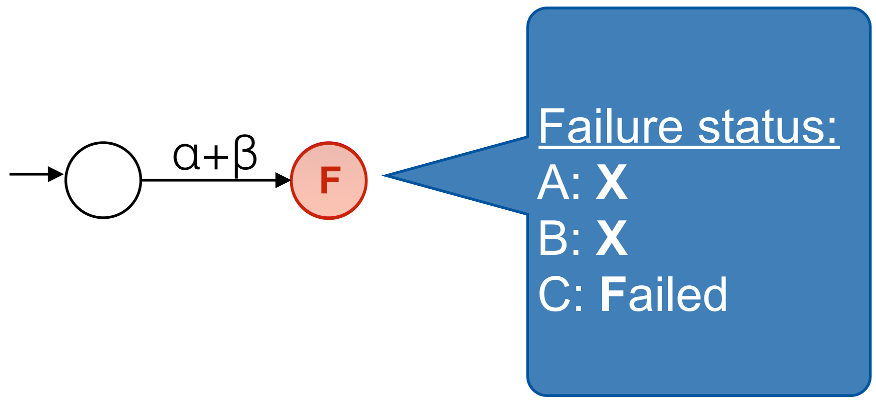




Don’t care propagation

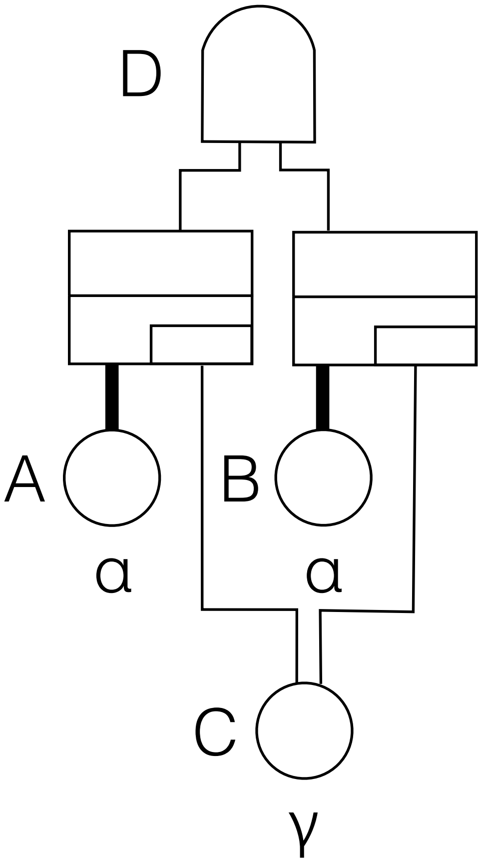
only consider failures making a difference

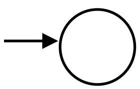




Symmetry reduction

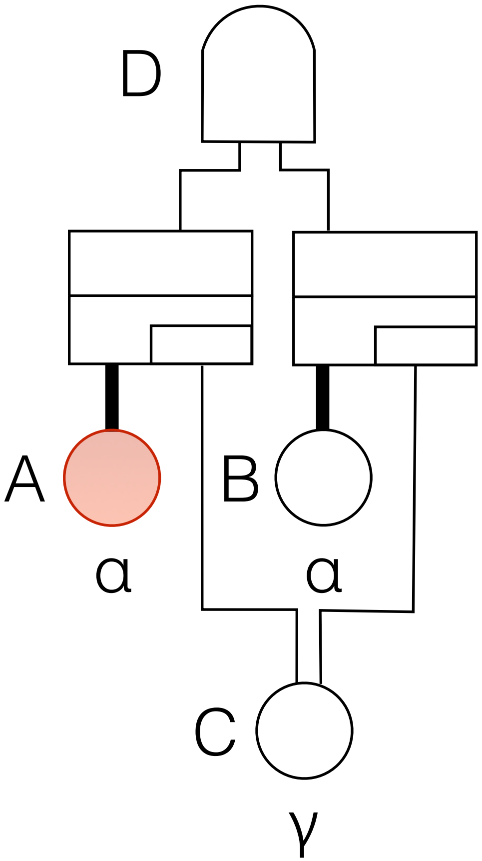
exploit symmetric structures

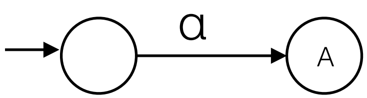




Symmetry reduction

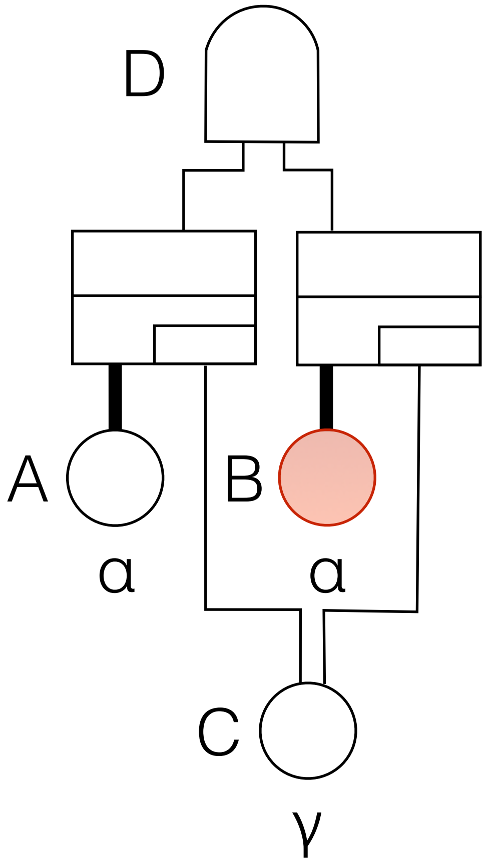
exploit symmetric structures

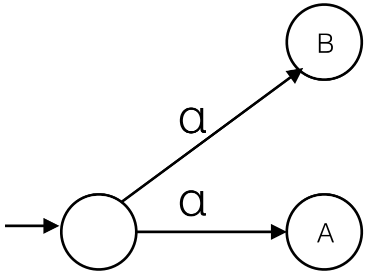




Symmetry reduction

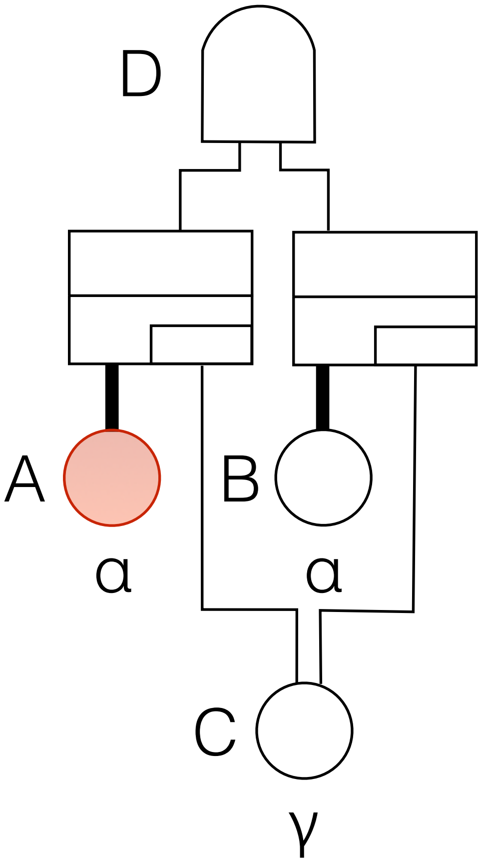
exploit symmetric structures

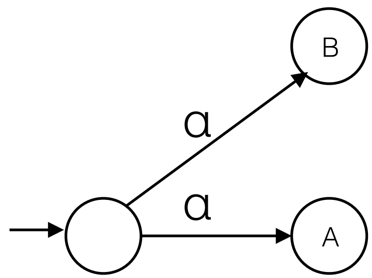




Symmetry reduction

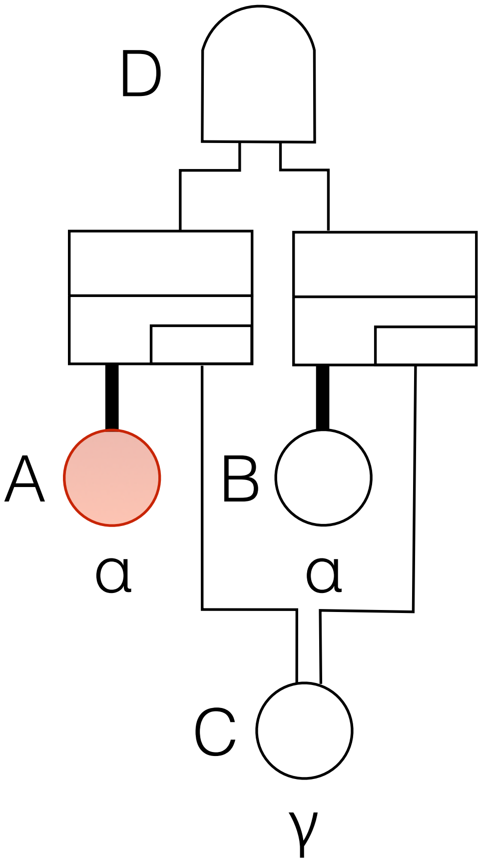
exploit symmetric structures

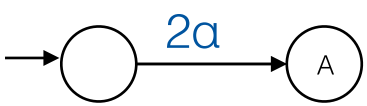




Symmetry reduction

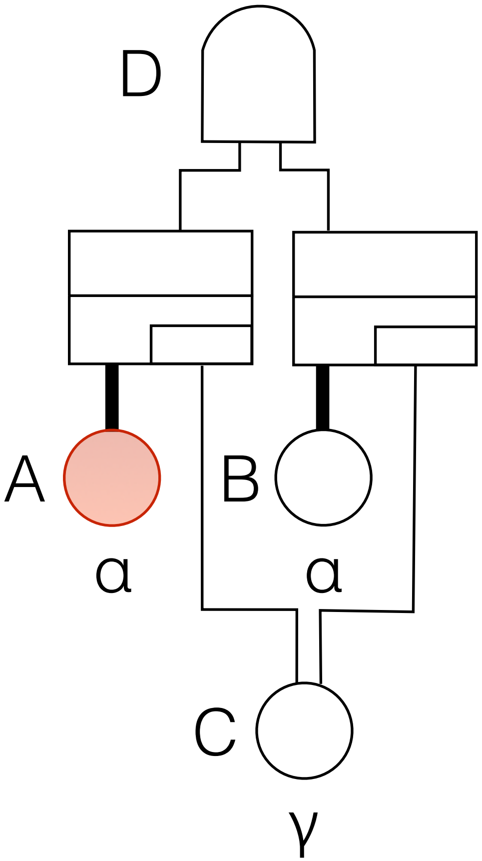
exploit symmetric structures

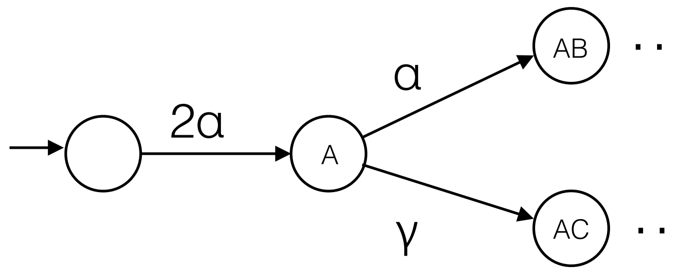




Symmetry reduction

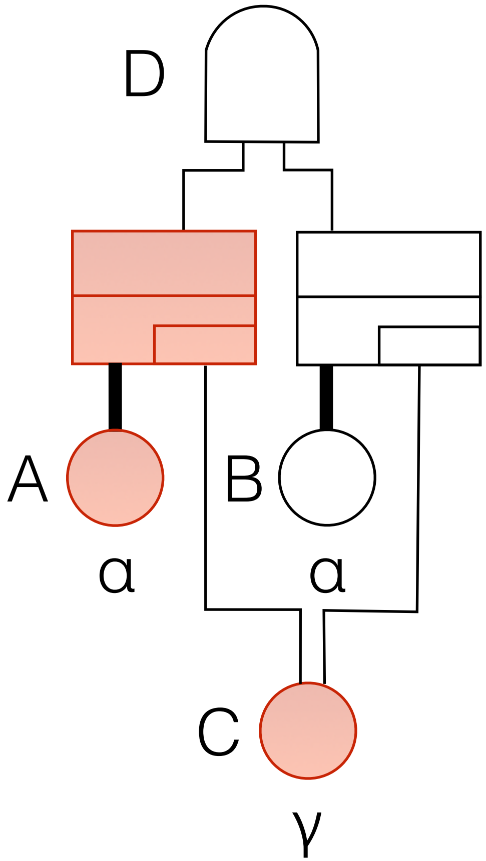
exploit symmetric structures

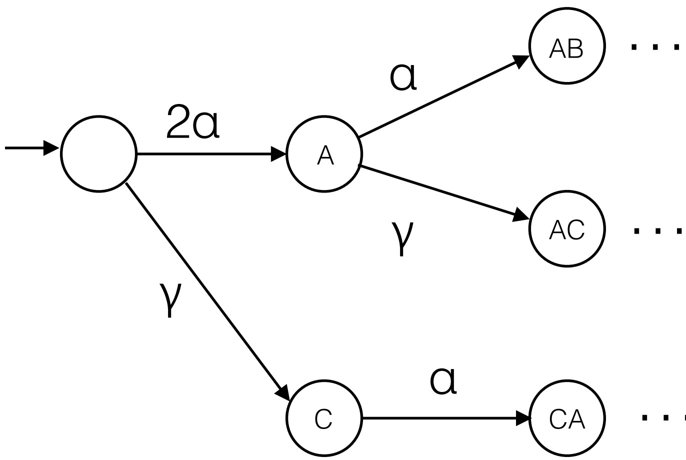




Symmetry reduction

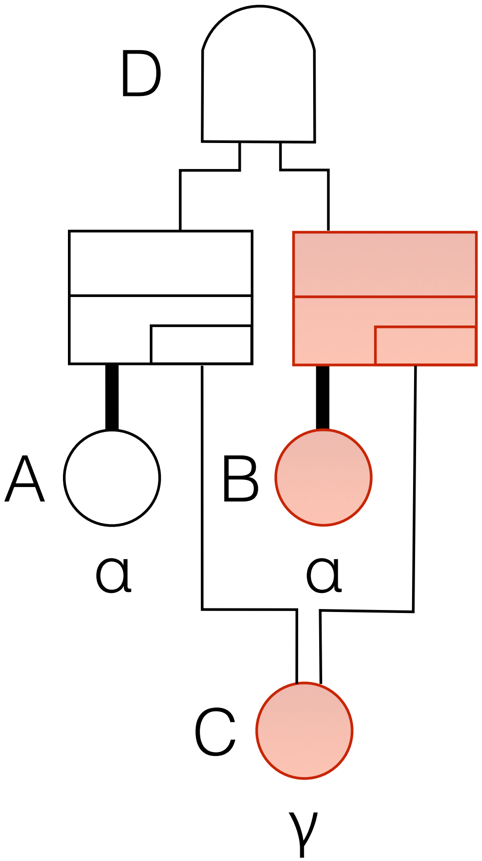
exploit symmetric structures

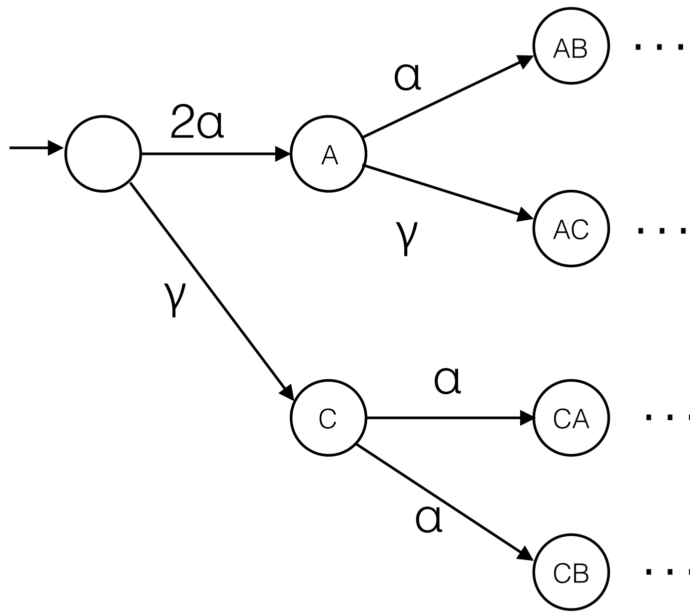




Symmetry reduction

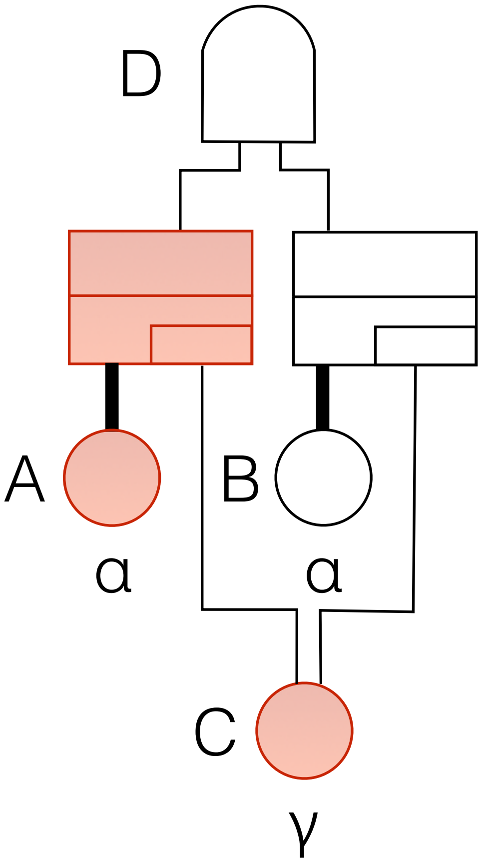
exploit symmetric structures

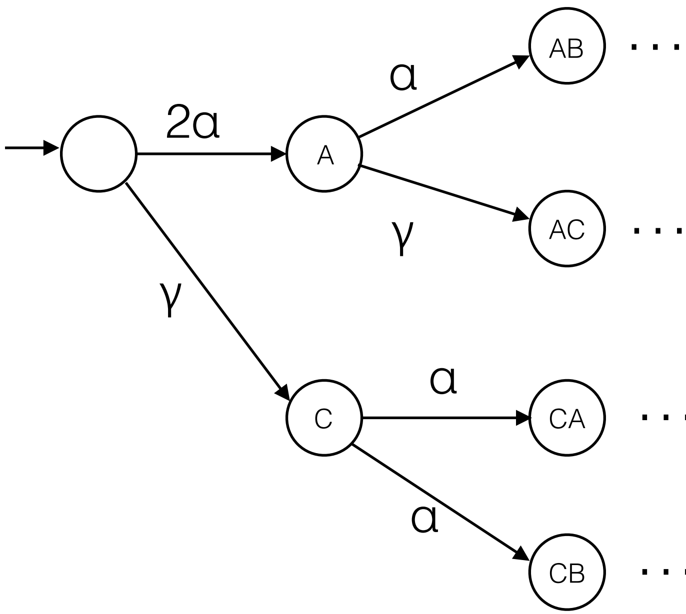




Symmetry reduction

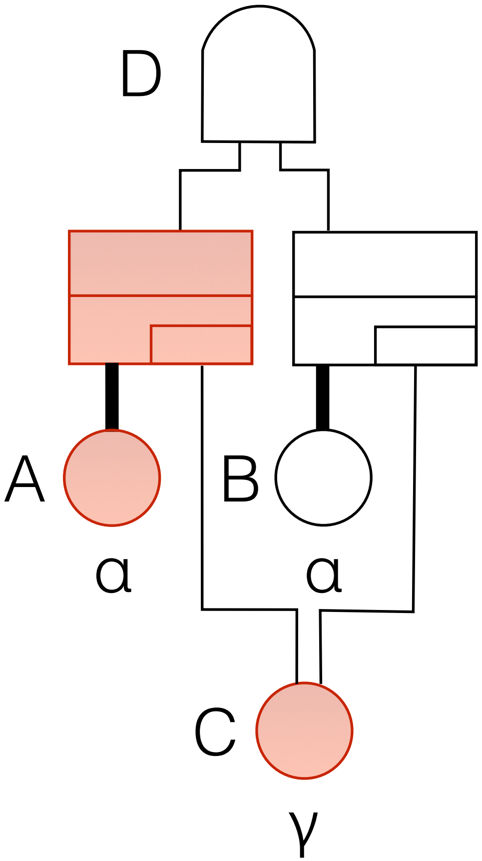
exploit symmetric structures

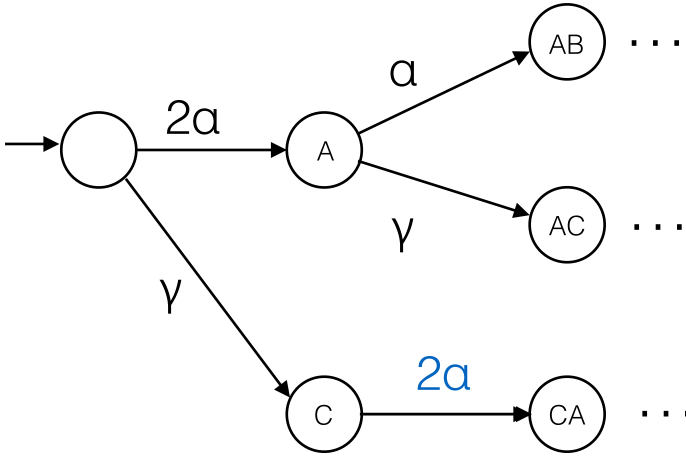




Symmetry reduction

exploit symmetric structures

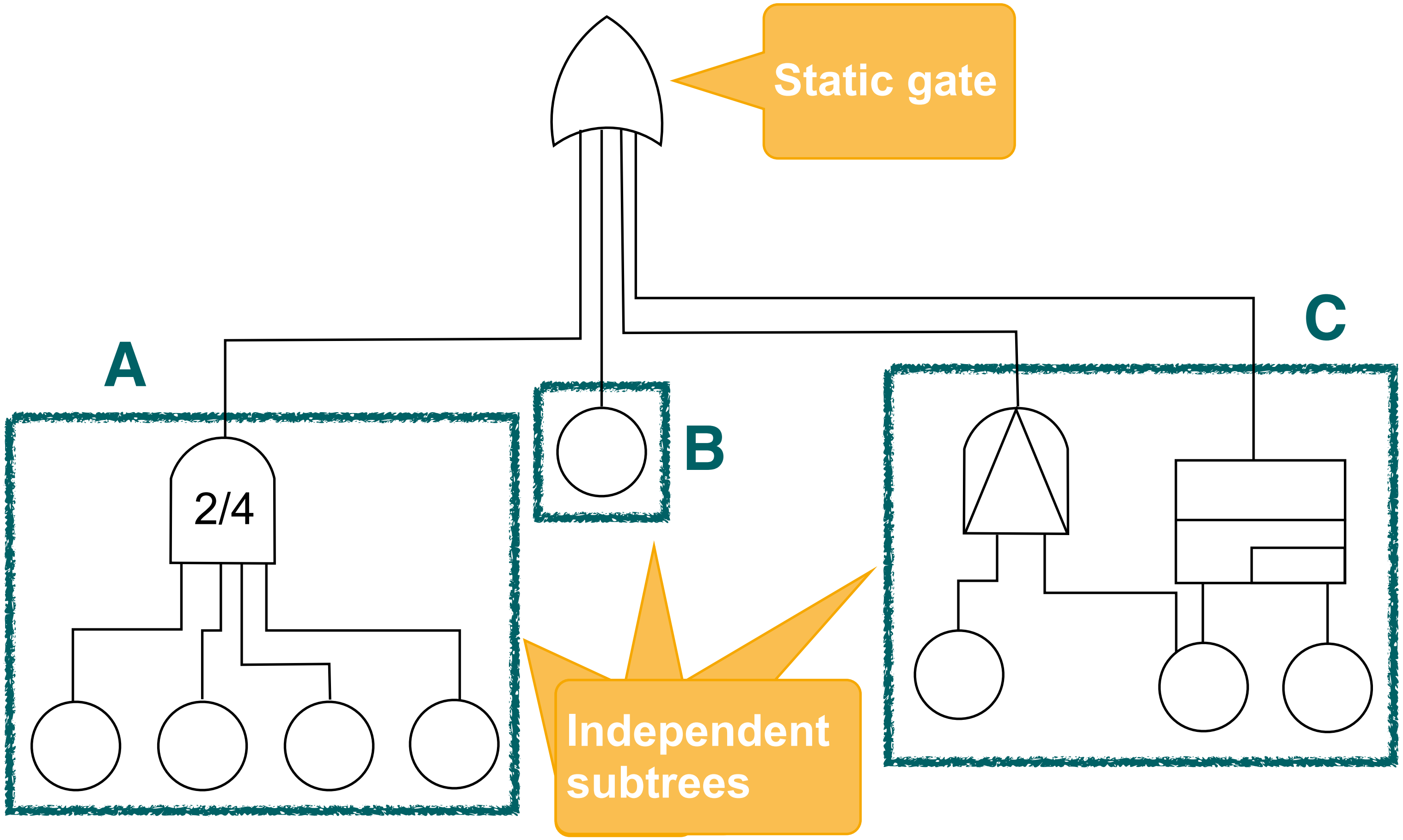




Modularisation

[Gulati, Dugan, RAMS 1997]

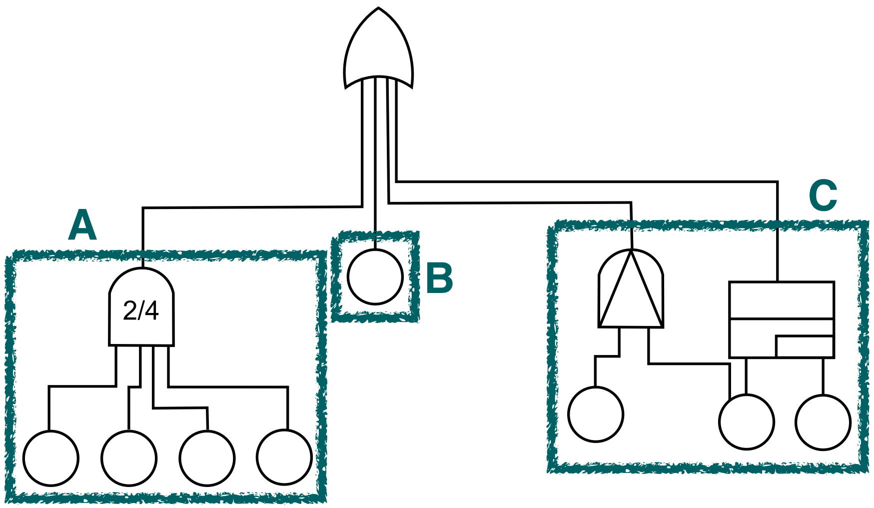
compositional analysis for reliability



Modularisation

[Gulati, Dugan, RAMS 1997]

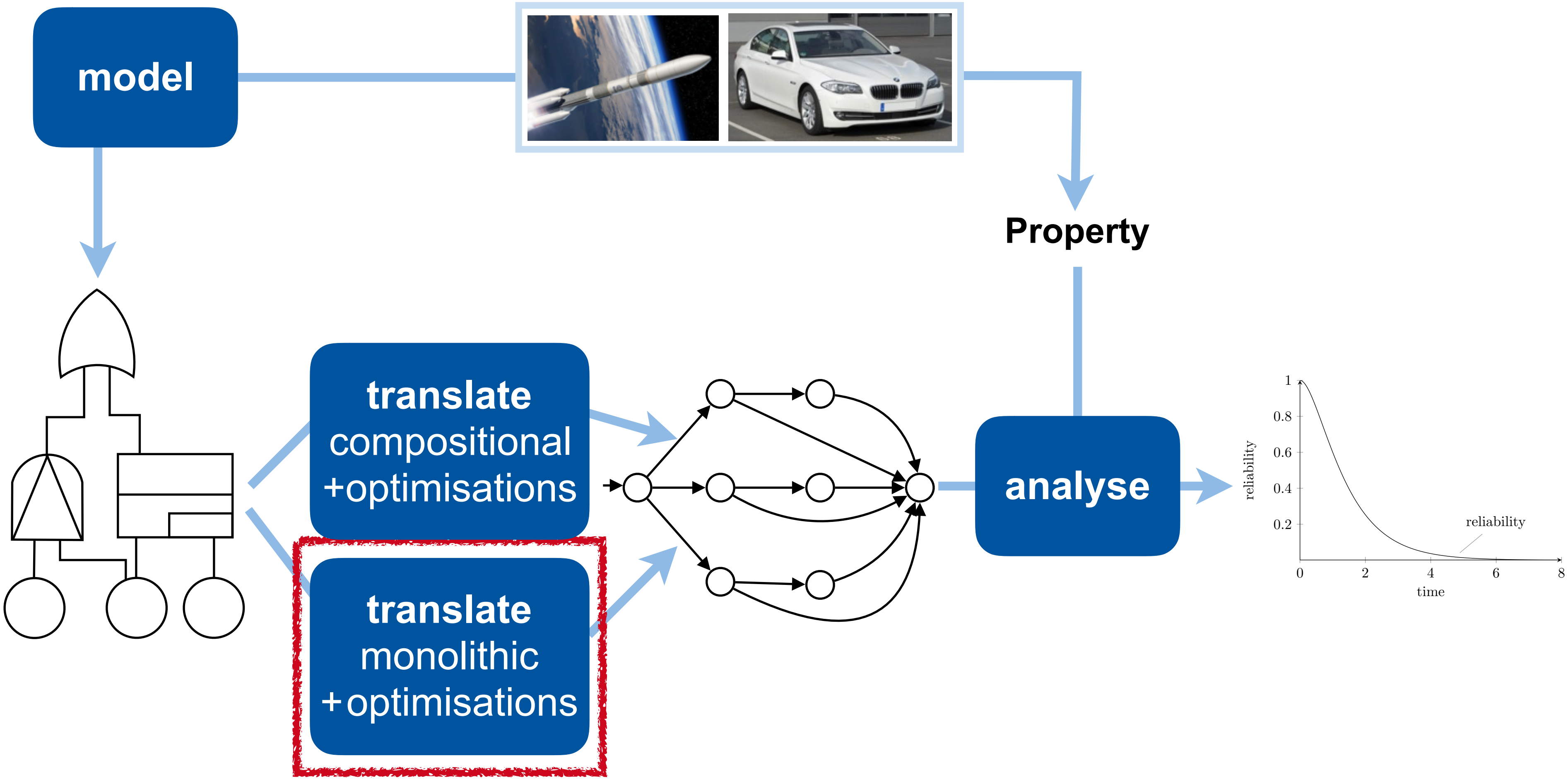
compositional analysis for reliability



1. Compute failure probability of A: p_A
2. Compute failure probability of B: p_B
3. Compute failure probability of C: p_C
4. Compute complete probability:

$$1 - (1-p_A)(1-p_B)(1-p_C)$$

Overview


model
Property
translate
compositional
+optimisations
translate
monolithic
+optimisations
analyse
1
0.8
0.6
0.4
0.2
reliability
reliability
0
2
4
6
8
time

Experimental evaluation

Storm

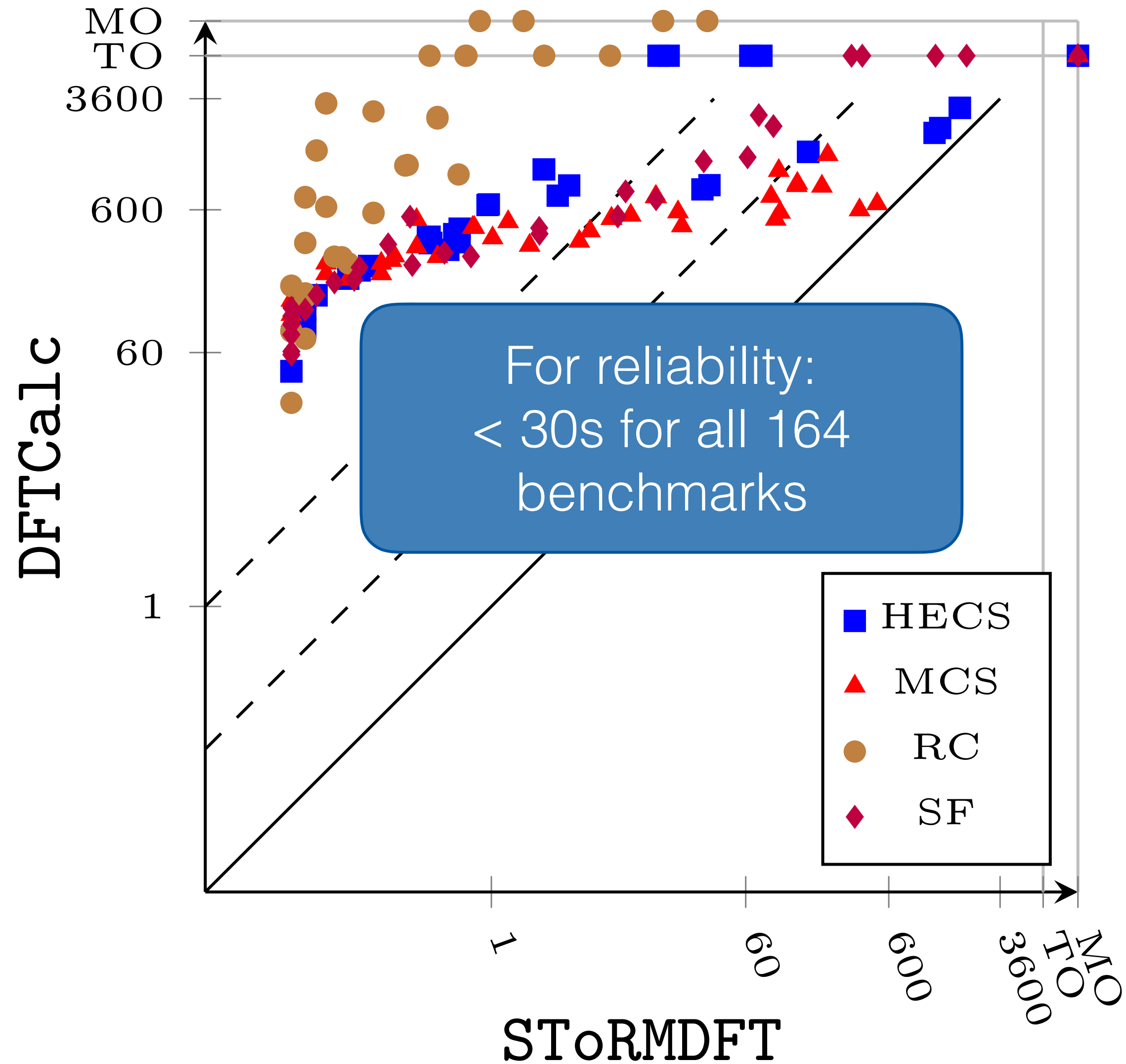
http://www.stormchecker.org

Experimental evaluation

- Comparison to compositional approach (DFTCalc)
- 4 sets of benchmarks (164 DFTs in total):
  - HECS (Hypothetical Example Computer System)
  - MCS (Multiprocessor Computing System)
  - RC (Railway Crossing)
  - SF (Sensor Filter)
- largest DFT: over 120 BEs
- mostly: 50-60 BEs
- Machine: 2,0 GHz, 8 GB RAM, 1 hour

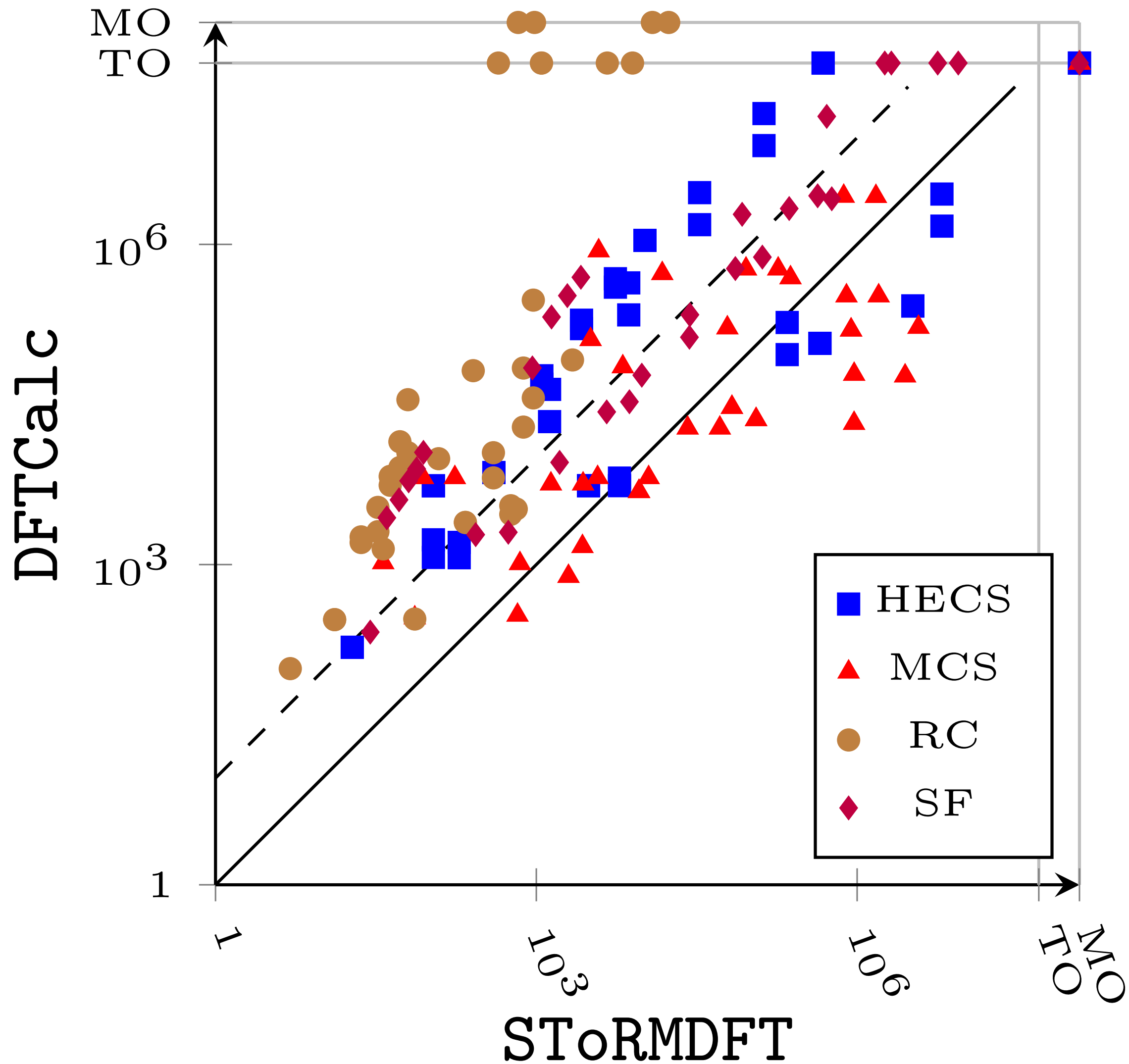
Analysis run times for MTTF

[Volk et al., SAFECOMP 2016]



Max. number of states during exploration

[Volk et al., SAFECOMP 2016]



Analysis run times: Tool comparison

[Volk et al., SAFECOMP 2016]

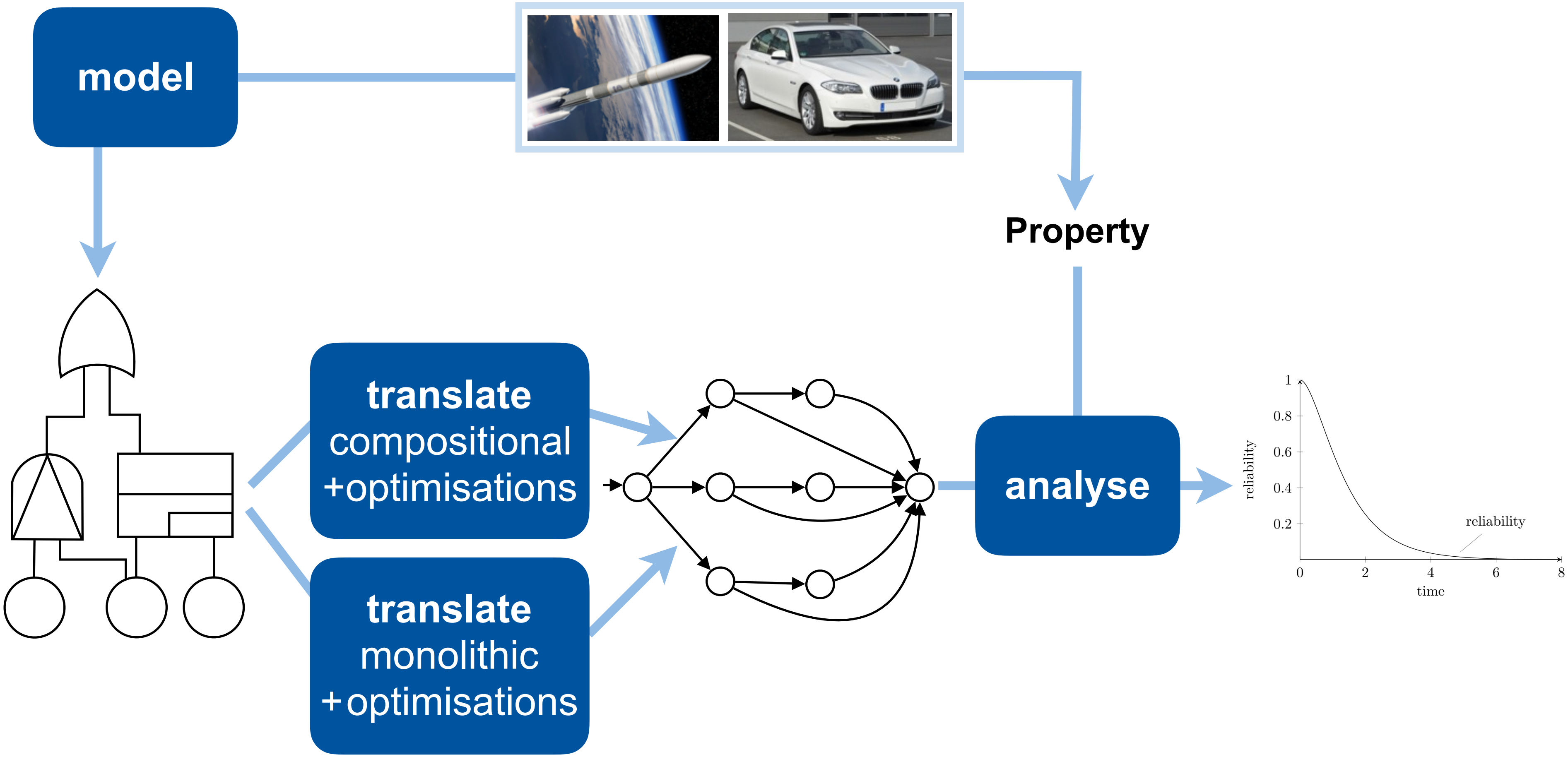
		Reliability				MTTF			
		DFTCalc		SToRMDFT		DFTCalc		SToRMDFT	
		#	Time	#	Time	#	Time	#	Time
HECS	**42**	38	27517,06s	42	3,07s	36	25580,93s	40	6973,02s
MCS	**42**	40	21342,25s	42	20,80s	38	18671,74s	38	2079,08s
RC	**38**	29	27495,17s	38	2,09s	29	27386,01s	38	65,09s
SF	**30**	26	16137,67s	30	1,82s	25	13825,36s	29	4390,85s
CAS	**8**	8	1301,41s	8	0,36s	8	1299,89s	8	0,37s
SAP	**4**	4	357,64s	4	0,30s	4	316,02s	4	0,16s

Analysis run times: Optimisations

[Volk et al., SAFECOMP 2016]

		Reliability					MTTF
		no opti.	Sym. Red.	Don't Care	Modular.	all	all (SR+DC)
$HECS_2$	time	30,3s	15,6s	1,1s	0,05s	0,04s	0,61s
	max. states	864.001	432.073	11.881	4	4	5.995
MCS_2	time	337,8s	46,0s	1,1s	0,05s	0,05s	0,21s
	max. states	10.469.377	1.374.946	17.689	67	37	2.701
RC_{10}	time	53,6s	0,1s	53,5s	0,20s	0,05s	0,07s
	max. states	1.048.577	122	1.048.577	3	3	122
$SF_{6,2}$	time	22,1s	7,4s	0,3s	0,04s	0,04s	0,08s
	max. states	1.132.097	355.111	2.602	4	4	919

Overview


model
Property
translate
compositional
+optimisations
translate
monolithic
+optimisations
analyse
reliability
time
1
0.8
0.6
0.4
0.2
0
2
4
6
8

A Modern Perspective on Fault Tree Analysis

Joost-Pieter Katoen and Matthias Volk

Joint work with: Majdi Ghadhab (BMW), Dennis Guck (TWT), Sebastian Junges (RWTH), Matthias Kuntz (BMW), Enno Ruijters (U. Twente) and Mariëlle Stoelinga (U. Twente)

Tutorial MMB 2018, Erlangen, BY

Roadmap of This Tutorial

Part 1. What are Dynamic Fault Trees?

- DFT Elements, Benchmarks, Intricacies,
  DFTs as Stochastic Petri Nets

Part 2. From DFTs to Markov Models, Compositionally

- Compositional State-Space Minimisation,
  Non-Determinacy

Part 3. From DFTs to Markov Models, Monolithically

- Symmetry Reduction, Don't Care Propagation

Part 4. DFT Analysis by Model Checking

- Reliability Measures, Core Algorithms, `Storm` Tool

Part 5. Advanced Optimisations

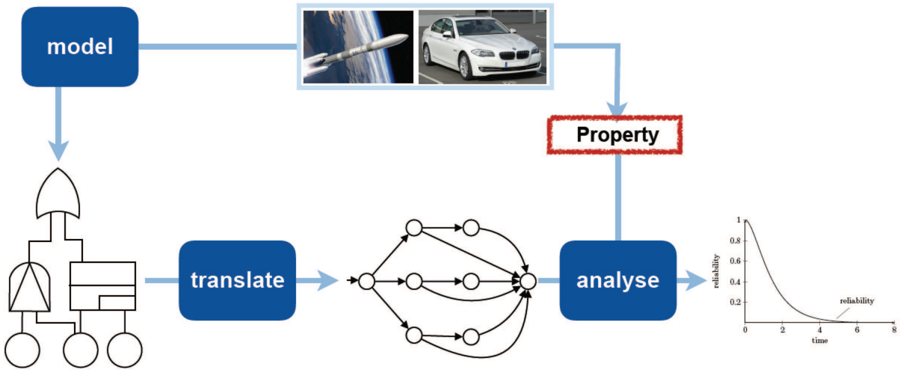
- Graph Rewriting, Partial State-Space Generation

Part 6. Industrial Applications and Outlook

Focus is on conveying intuition and experimental results

Overview

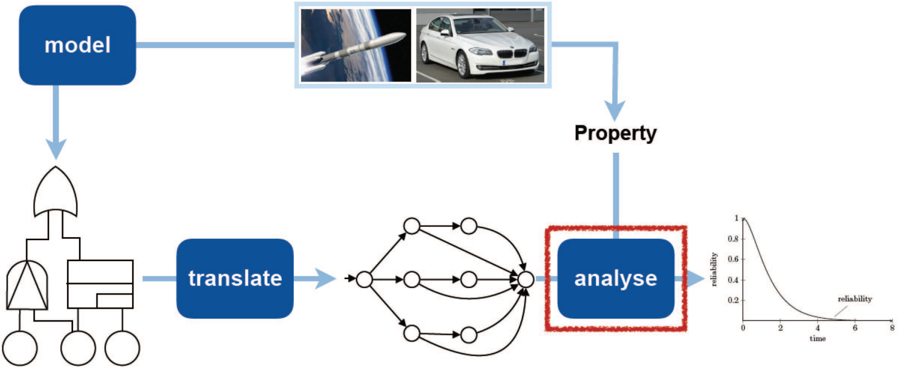
Graphical Overview



Reliability Measures [Barlow and Proschan, 1985]

- The reliability of DFT F is the probability that the system it represents operates for a certain amount of time without failing.
- The availability at time t is the probability that the system is functioning at a given time.
  - Availability of interval $[t, t']$ is the fraction of $[t, t']$ in which the system is operational
  - For repairable DFTs, also the long-run availability is considered
- The Mean Time To Failure (MTTF) is the expected time from the moment the system becomes operational, to the moment the system subsequently fails.
- For repairable systems, the Mean Time Between Failure (MTBF) denotes the mean time between two successive failures

Graphical overview

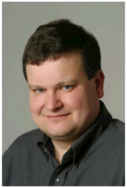


Probabilistic Model Checking

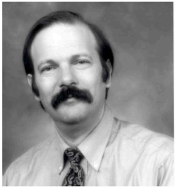
"A promising new direction in formal methods research these days is probabilistic model checking, with associated tools for quantitative evaluation of system performance along with correctness."

Theory in Practice for System Design and Verification

Rajeev Alur
Univ. of Pennsylvania



Thomas A. Henzinger
IST Austria



Moshe Y. Vardi
Rice University

ACM SIGLOG News 2015

Model Checking and ISO 26262

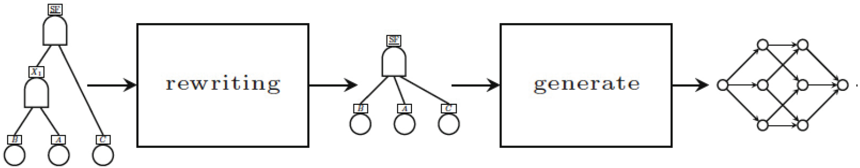
"Metrics are verifiable and precise enough
to differentiate between different architectures"

$\Rightarrow$ PMC provides hard guarantees; no statistical ones

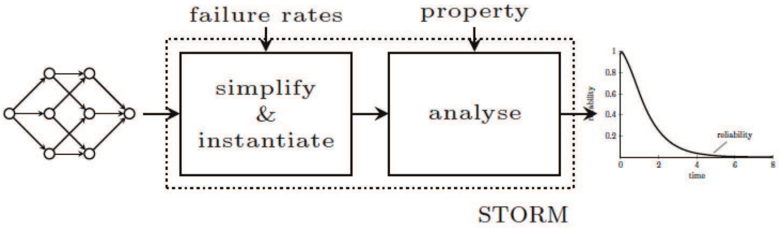
"[for systems where the] concept is based on redundant safety mechanisms,
multiple-point failures of a higher order than two are considered in the analysis"

$\Rightarrow$ PMC naturally supports analysis of multiple-point of failures

Model Checking DFTs



DFT simplification and state-space generation



DFT analysis using model checking

Properties in PMC

	Discrete	Continuous
Logic	probabilistic CTL	probabilistic timed CTL
Monitors	deterministic automata (safety and LTL)	deterministic timed automata (MITL fragments)

Others: e.g., conditional probs, multi-objective, rewards, quantiles, etc.

Core problem: computing (timed) reachability probabilities

Reachability Probabilities

Problem

Consider a finite MC with $s \in S$ and $G \subseteq S$.

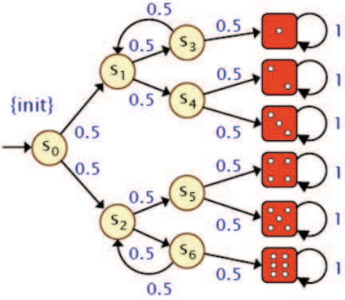
Aim: determine $\Pr(s \models \Diamond G) = \Pr_s\{ \pi \in Paths(s) \mid \pi \models \Diamond G \}$

Characterisation of reachability probabilities

- Let variable $x_s = \Pr(s \models \Diamond G)$ for any state s
  - if G is not reachable from s, then $x_s = 0$
  - if $s \in G$ then $x_s = 1$
- For any state $s \in Pre^*(G) \setminus G$:

$$x_s = \underbrace{\sum_{t \in S \setminus G} \mathbf{P}(s,t) \cdot x_t}_{\text{reach } G \text{ via } t \in S \setminus G} + \underbrace{\sum_{u \in G} \mathbf{P}(s,u)}_{\text{reach } G \text{ in one step}}$$

Reachability Probabilities: Knuth-Yao's Die



- Consider the event $\Diamond 4$
- We obtain:

$$x_1 = x_2 = x_3 = x_5 = x_6 = 0 \text{ and } x_4 = 1$$

$$x_{s_1} = x_{s_3} = x_{s_4} = 0$$

$$x_{s_0} = \frac{1}{2}x_{s_1} + \frac{1}{2}x_{s_2}$$

$$x_{s_2} = \frac{1}{2}x_{s_5} + \frac{1}{2}x_{s_6}$$

$$x_{s_5} = \frac{1}{2}x_5 + \frac{1}{2}x_4$$

$$x_{s_6} = \frac{1}{2}x_{s_2} + \frac{1}{2}x_6$$

- Gaussian elimination yields:

$$x_{s_5} = \frac{1}{2},\ x_{s_2} = \frac{1}{3},\ x_{s_6} = \frac{1}{6}, \text{ and } \boxed{x_{s_0} = \frac{1}{6}}$$

Reachability Probabilities are Pivotal

- Repeated reachability $\Pr(s \vDash \Box\Diamond G)$:

  Determine probability to reach a terminal SCCs containing a G-state

- Probabilistic CTL model checking

  Recursive descent on parse tree using reach-probabilities at nodes

- LTL formulas $\Pr(s \vDash \varphi)$:
  1. Transform φ into a deterministic (Rabin) automaton
  2. Take the product of the Markov chain and the automaton
  3. Determine the probability to reach an accepting terminal SCC from s

This covers (much) more than the reliability measures on DFTs.

Probabilistic CTL

[Hansson & Jonsson, 1989]

- PCTL interpretation is Boolean, i.e., a formula is satisfied or not.
- For path-formula φ and threshold $\succ p$ with $\succ \in \{ >, \geqslant \}$ and $p \in \mathbb{Q}$:

  PCTL-formula $[\varphi]_{\succ p}$ denotes

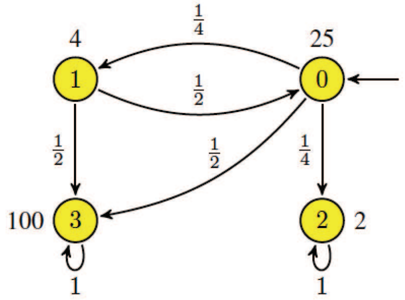
  all paths satisfying φ occur with probability $\succ p$
- $[\cdot]_{\succ p}$ is probabilistic counterpart of CTL path-quantifiers $\exists$ and $\forall$.
- Examples: $[\Diamond a]_{>1/2}$, $[\Diamond [\Box a]_{=1}]_{>1/2}$ and $[\Box(\neg a \wedge [\Diamond a]_{>0})]_{>0}$.

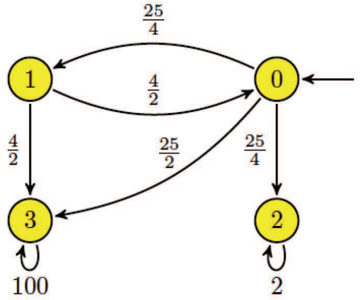
PCTL model checking is in P.

Random Timing

Continuous-Time Markov Chains

A CTMC is a DTMC with an *exit rate* function $r : S \rightarrow \mathbb{R}_{>0}$ where $r(s)$ is the rate of an exponential distribution.





Zenoness

Zeno theorem

In every CTMC, almost surely no Zeno runs occur.

In contrast to timed automata verification, Zeno runs thus pose no problem.

Timed Reachability Probabilities [Baier *et al.*, 2003]

Problem

Consider a finite CTMC with $s \in S$, $t \in \mathbb{R}_{\geqslant 0}$ and $G \subseteq S$.

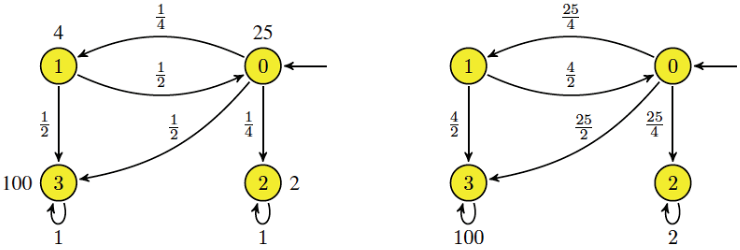
Aim: determine $\Pr(s \models \Diamond^{\leqslant t} G)$.

Characterisation of timed reachability probabilities

- Let function $x_s(t) = \Pr(s \models \Diamond^{\leqslant t} G)$ for any state s
  - if G is not reachable from s, then $x_s(t) = 0$ for all t
  - if $s \in G$ then $x_s(t) = 1$ for all t
- For any state $s \in Pre^*(G) \setminus G$:

$$x_s(t) = \int_0^t \sum_{s' \in S} \underbrace{\mathbf{R}(s, s') \cdot e^{-r(s) \cdot x}}_{\text{probability to move to state } s' \text{ at time } x} \cdot \underbrace{x_{s'}(t-x)}_{\text{prob. to fulfill } \Diamond^{\leqslant t-x} G \text{ from } s'} \, dx$$

Timed Reachability Probabilities



Integral equations for $\Diamond^{\leqslant 10}\, 2$:

- $x_3(d) = 0$ and $x_2(d) = 1$ for all d
- $x_0(d) = \int_0^d {}^{25}\!/_4 \cdot e^{-25 \cdot x} \cdot x_1(d-x) \; + \; {}^{25}\!/_4 \cdot e^{-25 \cdot x} \cdot x_2(d-x) \; dx$
- $x_1(d) = \int_0^d {}^{4}\!/_2 \cdot e^{-4 \cdot x} \cdot x_0(d-x) \; + \; {}^{4}\!/_2 \cdot e^{-4 \cdot x} \cdot x_3(d-x) \; dx$

Timed Reachability Probabilities

Reachability probabilities

Solve a system of linear equations for which many efficient techniques exist.

Timed reachability probabilities

Solve a system of Volterra integral equations.
Non-trivial, inefficient, and has several pitfalls such as numerical stability.

Solution

Reduce $\Pr(s \vDash \Diamond^{\leqslant t} G)$ to computing transient probabilities.

Timed Reachability Probabilities = Transient Probabilities

Aim

Compute $\Pr(s \models \Diamond^{\leqslant t} G)$ in CTMC $\mathcal{C}$. Observe that once a path π reaches G within t time, then the remaining behaviour along π is not important.
$\Rightarrow$ make all states in G absorbing.

$$\underbrace{\Pr(s \models \Diamond^{\leqslant t} G)}_{\text{timed reachability in } \mathcal{C}} = \underbrace{\Pr(s \models \Diamond^{=t} G)}_{\text{timed reachability in } \mathcal{C}[G]} = \underbrace{\vec{p}(t) \text{ with } \vec{p}(0) = \mathbf{1}_s}_{\text{transient prob. in } \mathcal{C}[G]}.$$

Transient probabilities can be efficiently computed as solutions of linear differential equations.

Computing Transient Probabilities

By solving a linear differential equation system

The transient probability vector $\underline{p}(t) = (p_{s_1}(t), \ldots, p_{s_k}(t))$ satisfies:

$$\underline{p}'(t) \;=\; \underline{p}(t) \cdot (\mathbf{R} - \mathbf{r}) \qquad \text{given} \qquad \underline{p}(0)$$

where $\mathbf{r}$ is the diagonal matrix of vector $\underline{r}$.

Solution using standard knowledge yields: $\underline{p}(t) \;=\; \underline{p}(0) \cdot e^{(\mathbf{R}-\mathbf{r}) \cdot t}$.

Computing the matrix exponential is a challenging numerical problem[1].

[1] 19 dubious ways to compute a matrix exponential [Moler & Van Loan, 1978/2003].

Uniformisation

CTMC $\mathcal{C}$ is uniform if $r(s) = r$ for all $s \in S$ for some $r \in \mathbb{R}_{>0}$.

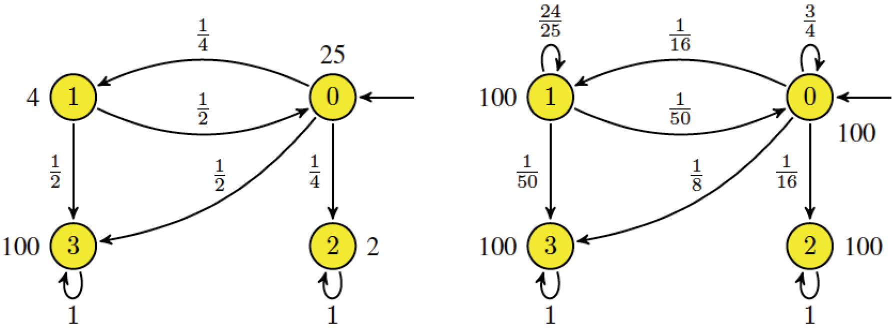
Uniformisation [Gross and Miller, 1984]

Let $r \in \mathbb{R}_{>0}$ such that $r \geqslant max_{s \in S}\ r(s)$. Then $\overline{r}(\mathcal{C})$ is the CTMC $\mathcal{C}$ with two changes: $\overline{r}(s) = r$ for all $s \in S$, and:

$$\overline{\mathbf{P}}(s,s') = \frac{r(s)}{r} \cdot \mathbf{P}(s,s') \text{ if } s' \neq s \quad \text{and} \quad \overline{\mathbf{P}}(s,s) = \frac{r(s)}{r} \cdot \mathbf{P}(s,s) + 1 - \frac{r(s)}{r}.$$

$\overline{\mathbf{P}}$ is a stochastic matrix and $\overline{r}(\mathcal{C})$ is uniform.

Uniformisation by Example



Uniformisation amounts to normalise the residence time in every CTMC state.

Benefits of Uniformisation

Transient probabilities of a CTMC and its uniformized CTMC coincide.

Thus: $\underbrace{\underline{p}(t) = \underline{p}(0) \cdot e^{(\mathbf{R} - \mathbf{r}) \cdot t}}_{\text{transient probablity in } \mathcal{C}} = \underbrace{\underline{p}(0) \cdot e^{(\overline{\mathbf{R}} - \overline{\mathbf{r}}) \cdot t}}_{\text{transient probablity in } \overline{r}(\mathcal{C})} = \underline{p}(0) \cdot e^{-r \cdot t} \cdot e^{r \cdot t \cdot \overline{\mathbf{P}}}$

Still a matrix exponential remains. Did we gain anything?
Yes. Since $\overline{\mathbf{P}}$ is stochastic, Taylor-Maclaurin yields $\sum_i \ldots \overline{\mathbf{P}}^i$.

Other Properties on CTMCs

- Expected time objectives

  Can be characterised as solution of set of linear equations

- Long-run average objectives
  1. Determine the limiting distribution in any terminal SCC
  2. Take weighted sum with reachability probabilities terminal SCCs

- Probabilistic timed CTL model checking

  recursive descent over parse tree

- Deterministic timed automata objectives
  1. Take product of the MC and the Zone automaton of the DTA[2]
  2. Determine the probability to reach an accepting zone

[2] This yields a piecewise deterministic Markov process.

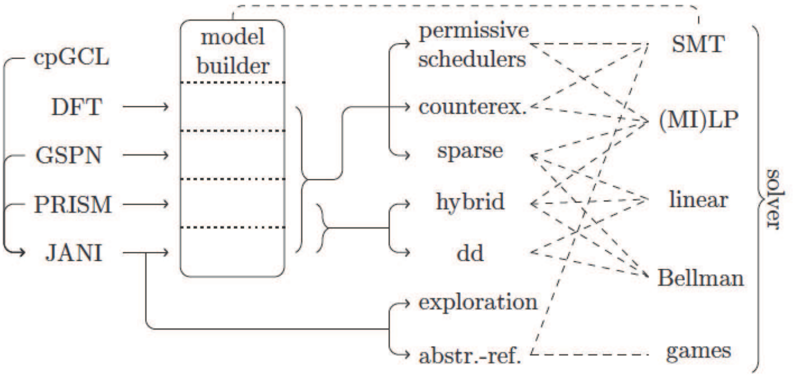
Probabilistic Model Checkers

- PRISM[3] [Kwiatkowska, Parker *et al.*]
- MRMC [Katoen *et al.*]
- iscasMC [Zhang *et al.*]
- iBioSim [Myers *et al.*]
- GreatSPN [Franceschinis *et al.*]
- SMART [Ciardo *et al.*]
- MarCie [Heiner *et al.*]
- PAT [Song Dong *et al.*]
- STоRM [Dehnert, Katoen *et al.*]
- ......

Statistical model checkers: Ymer, Vesta, UppAal-SMC, PlasmaLab, ...

[3] Recipient HVC Award 2016.

The Probabilistic Model Checker STORM [Dehnert *et al.*, CAV 2017]

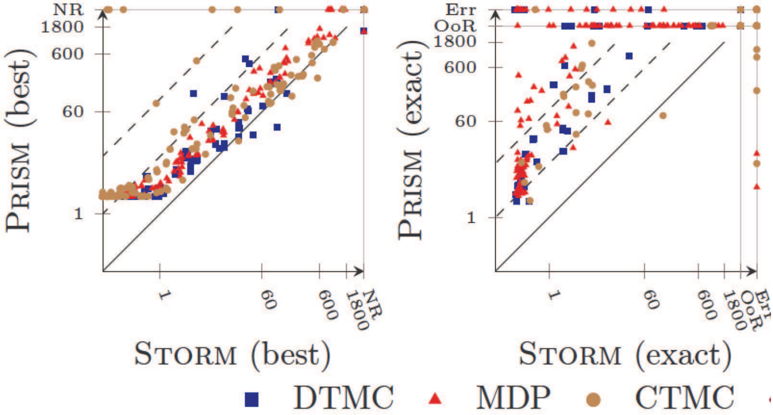


Native support for Dugan's dynamic fault trees

About 100,000 lines of C++ code

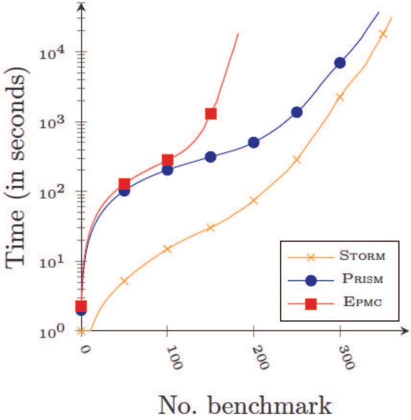
stormchecker.org

SToRM's Performance



Comparing the best engines of PRISM and StoRM

SToRM's Performance



Comparing the best engines for all

Fault Tree Analysis by Model Checking

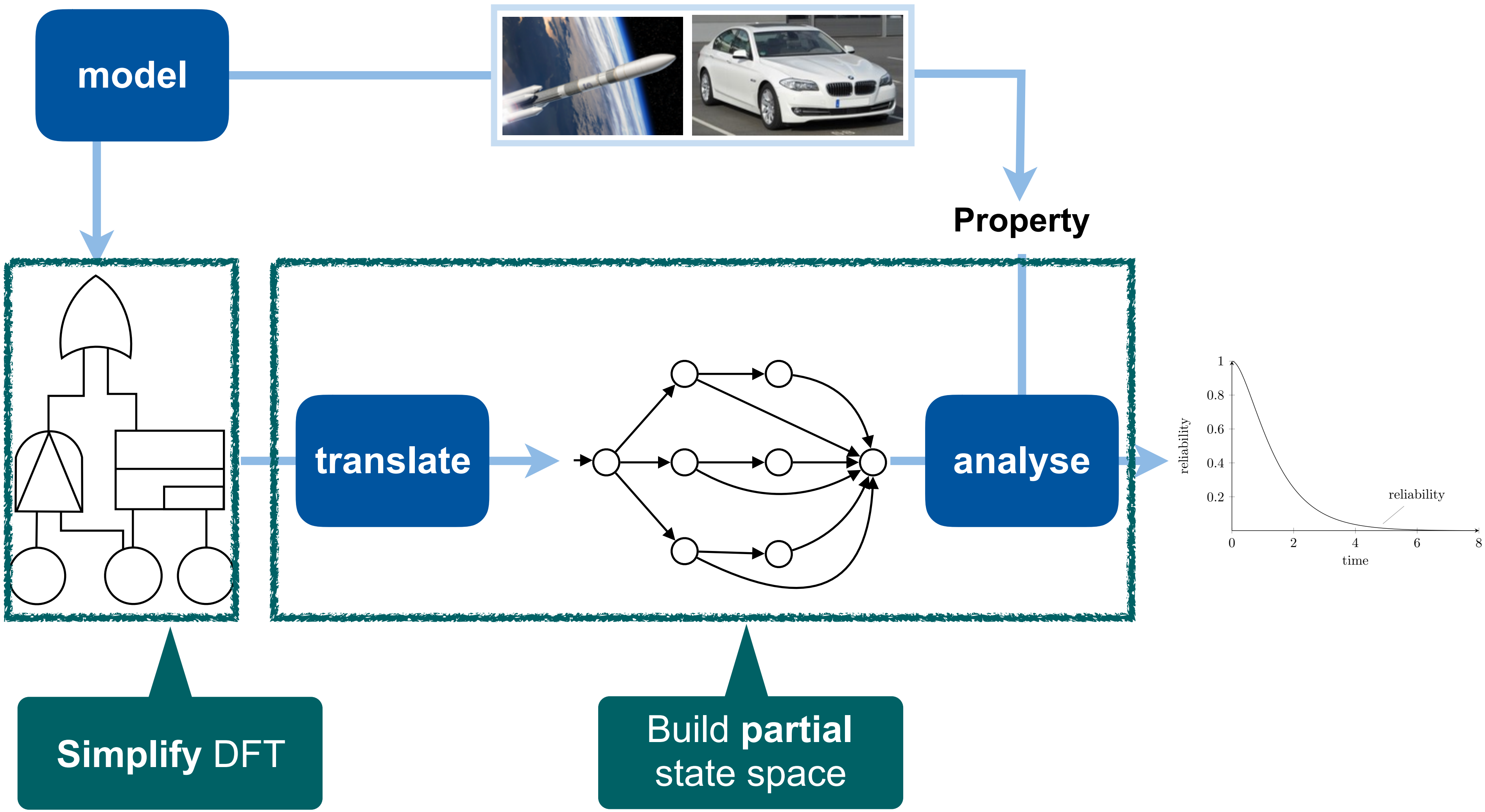
Probabilistic model checking ≈ automated verification of models with randomness

Its Pros:

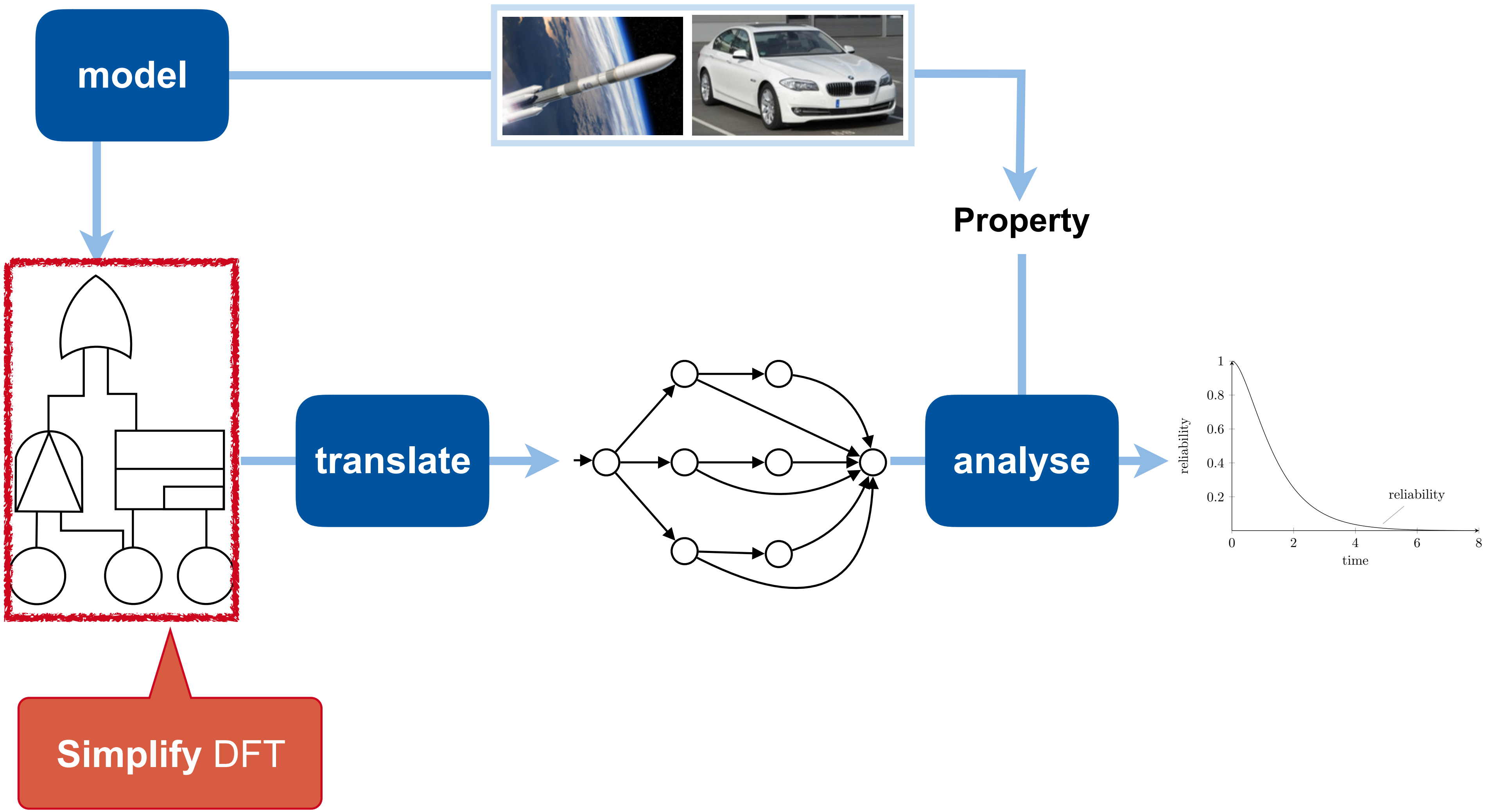
- Efficient and effective techniques for slim state-space generation
- Fully automated approach typically much faster than FT analysis
- Beyond MTTF, availability and reliability: many safety measures
- Supports checking functional correctness of FTs
- Supports non-determinacy as first-class citizen
- Tailored abstraction . . . $\Longrightarrow$ . . . scalability

Model-checking times are negligible compared to state-space generation times

Overview

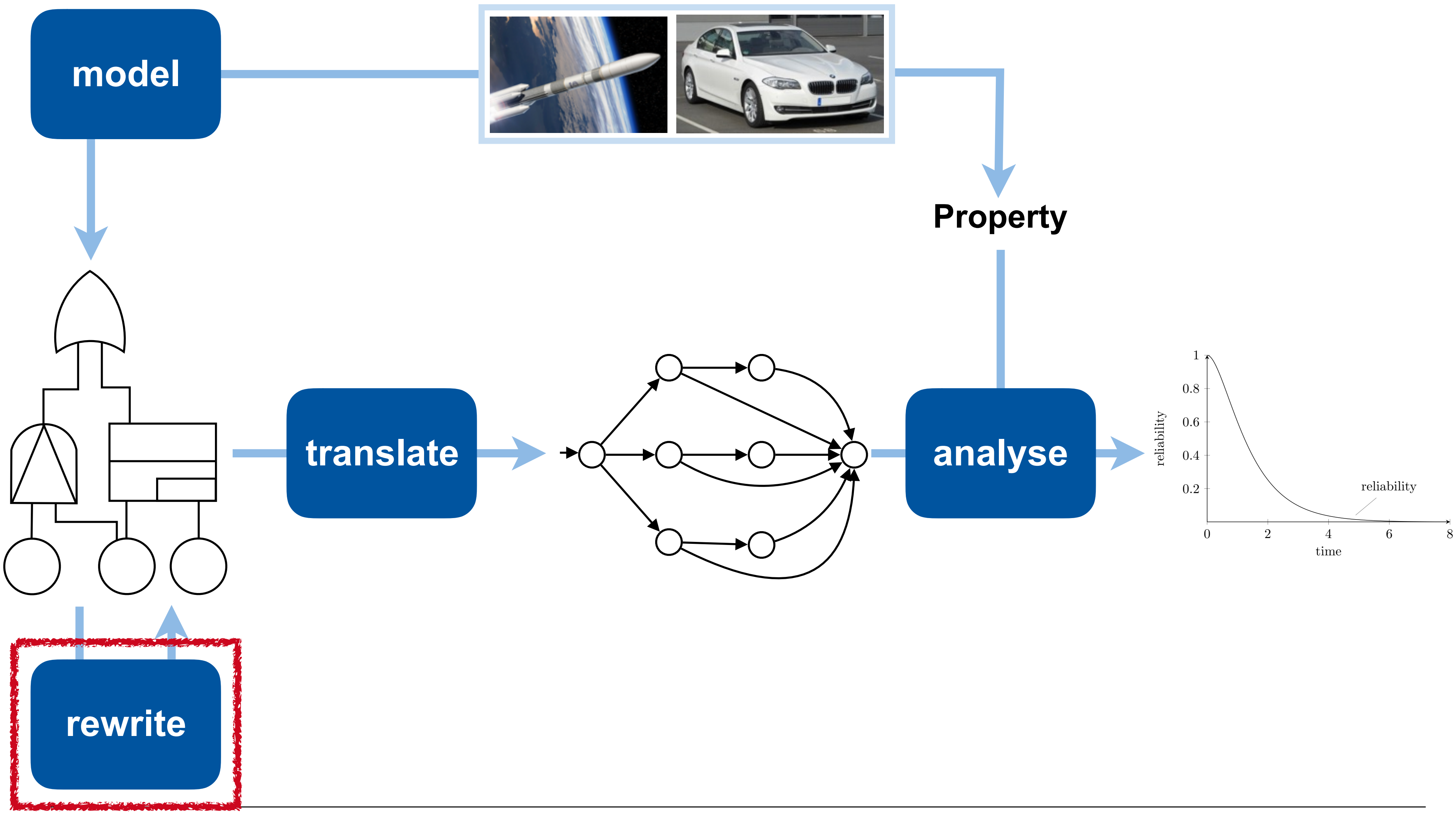

model
Property
translate
analyse
1
0.8
0.6
0.4
0.2
reliability
reliability
0
2
4
6
8
time
Simplify DFT
Build partial state space

Overview



model
Property
translate
analyse
reliability
1
0.8
0.6
0.4
0.2
0
2
4
6
8
time
reliability
Simplify DFT

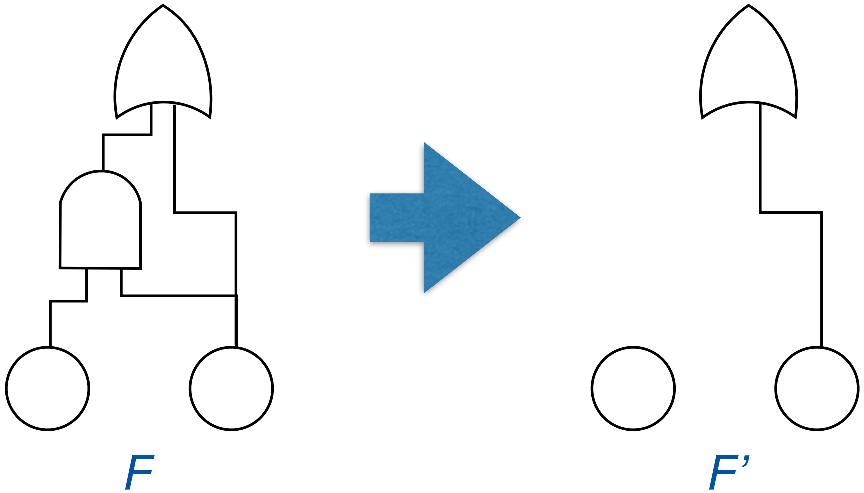
Overview


model
Property
translate
analyse
rewrite
reliability
time
1
0.8
0.6
0.4
0.2
0
2
4
6
8

Rewriting DFTs

[Junges et al., Formal Aspects of Computing 2017]

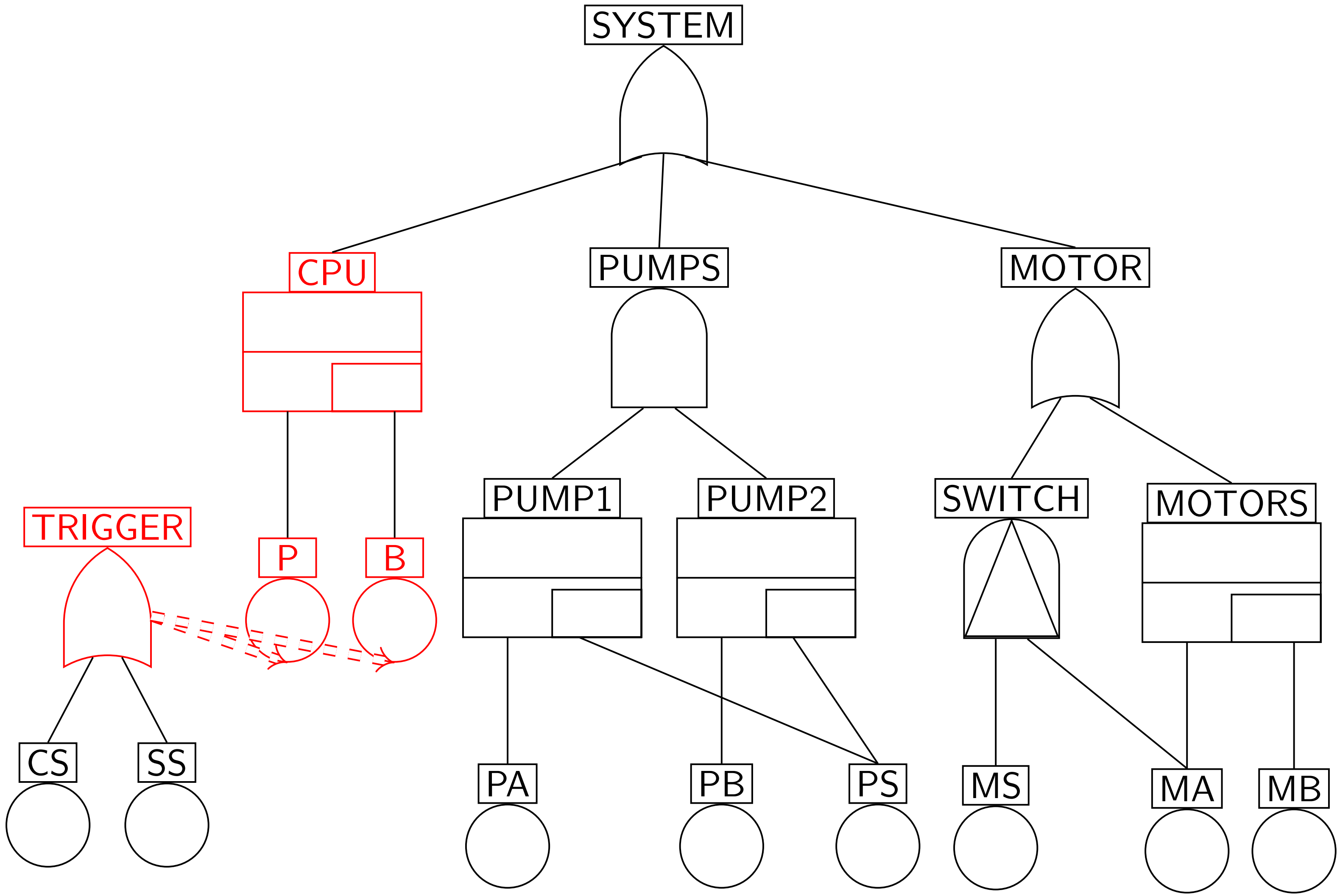
- Simplify DFTs before analysis
- Reduce DFT *F* via graph rewriting to smaller DFT *F'*



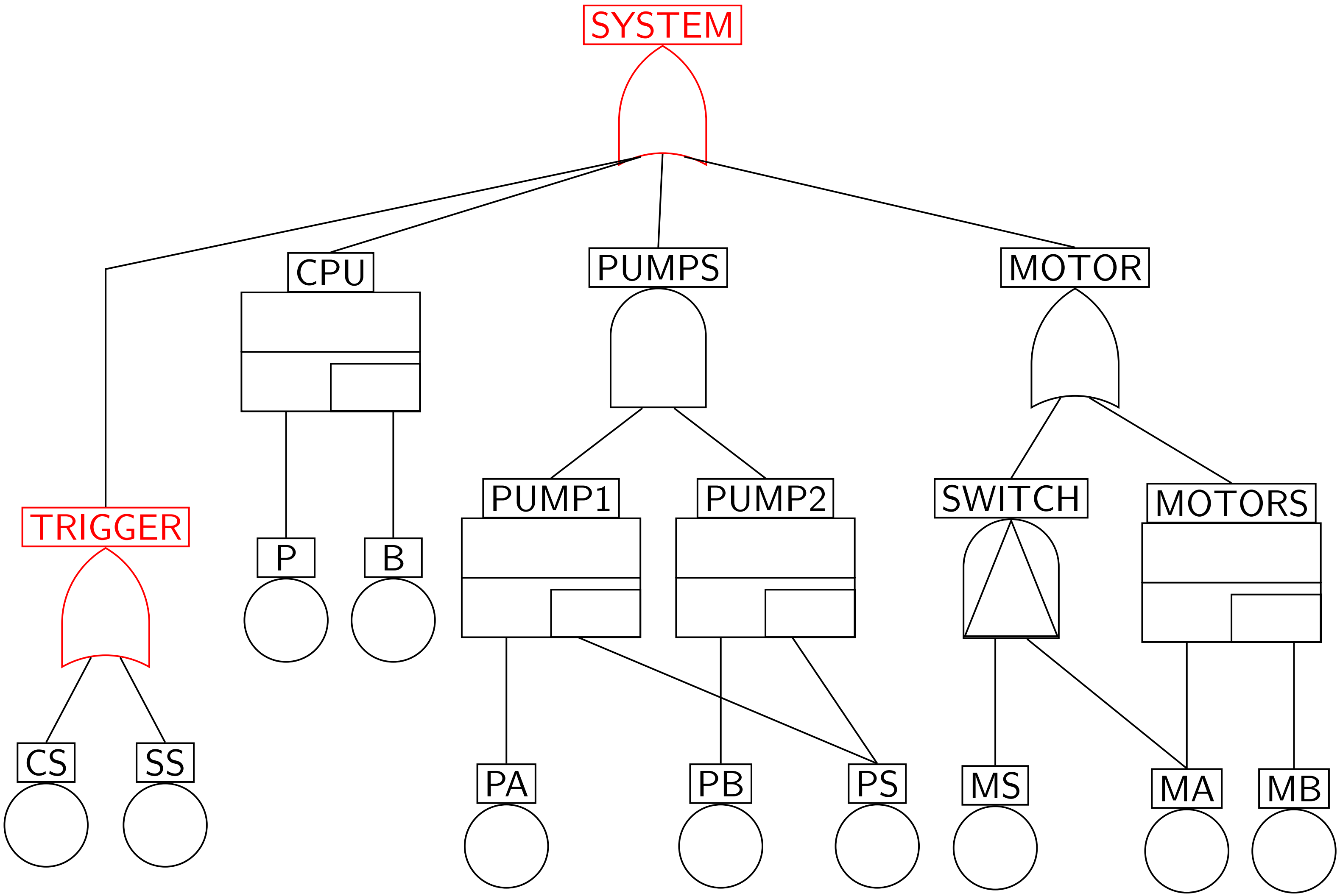
- Rewriting preserves measures of interest (reliability, MTTF, …)

  ➡ Suffices to analyse *F'*

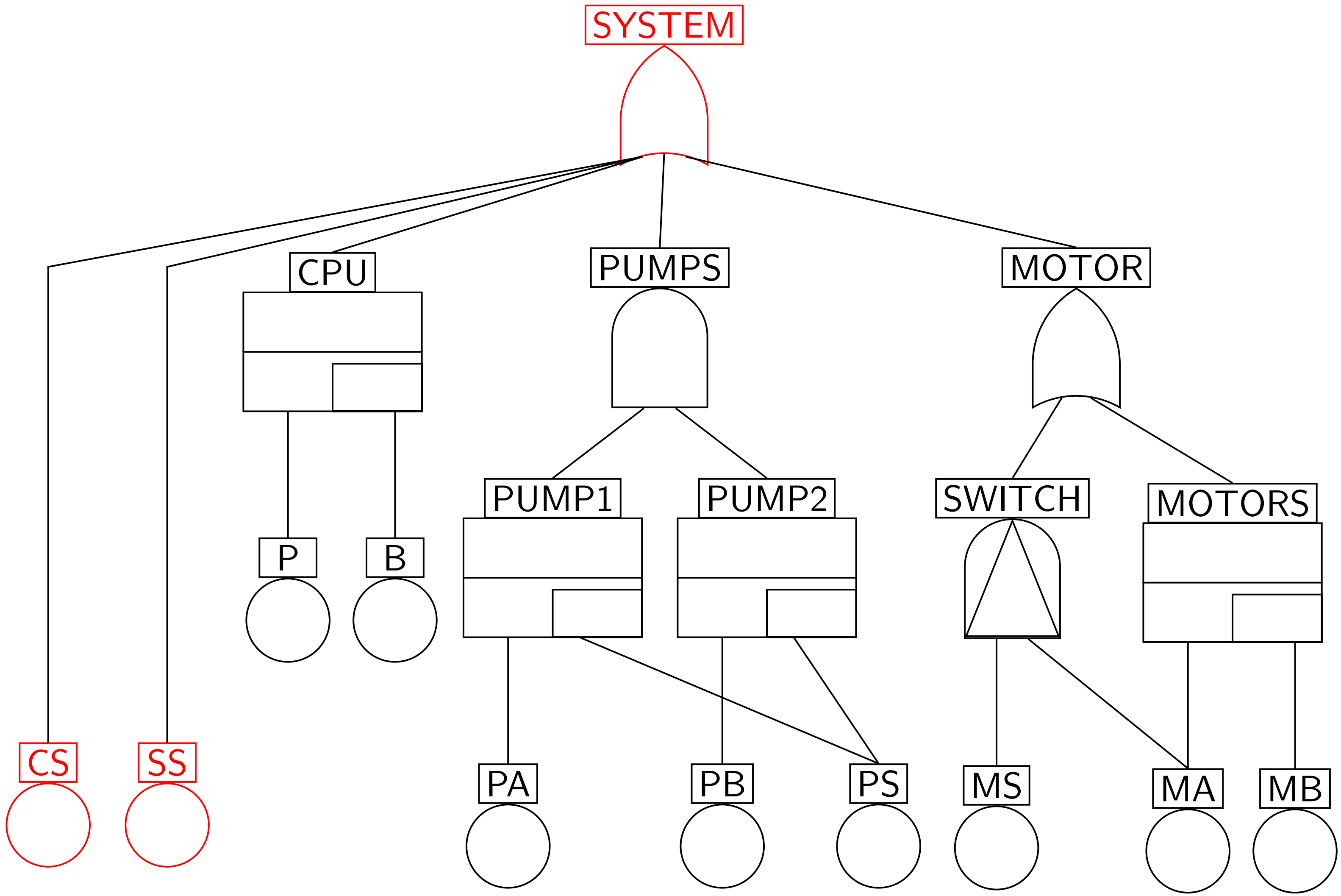
Example simplification



Example simplification

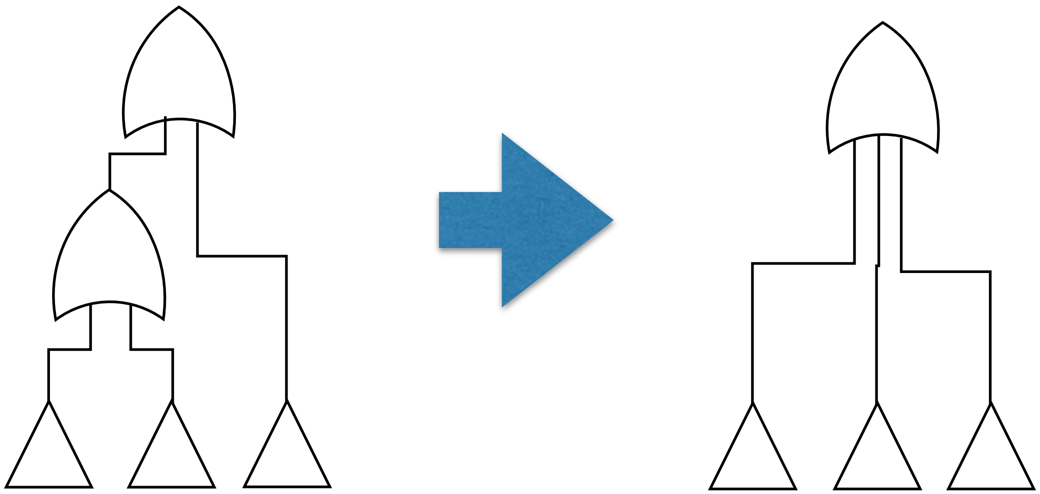


Example simplification



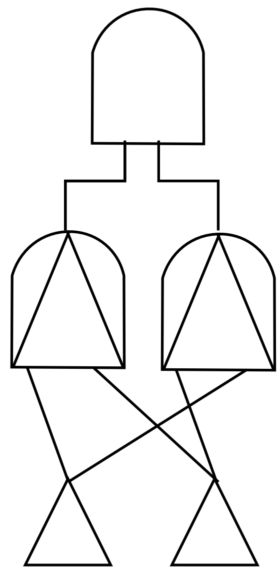
Reduction rule I

Flattening of static gates



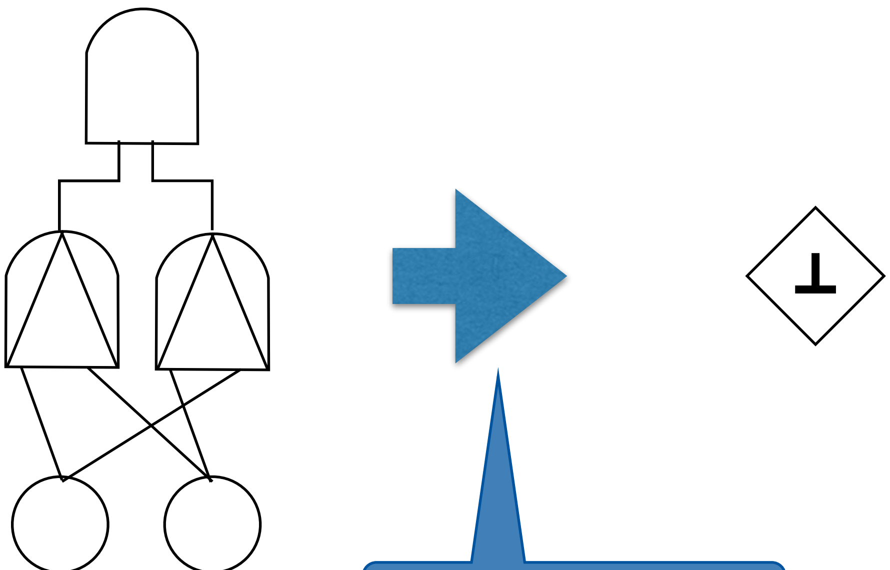
Reduction rule II

Simplification of conflicting PAND gates



Reduction rule II

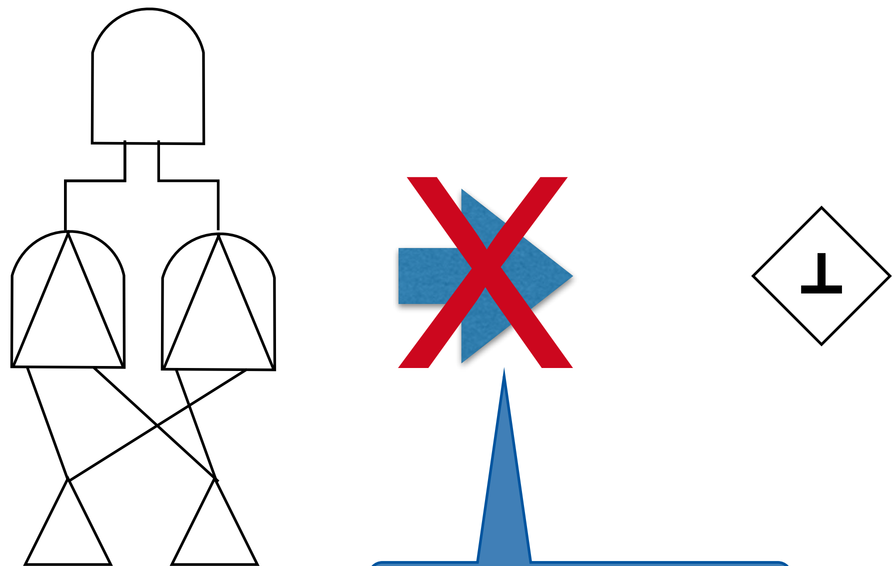
Simplification of conflicting PAND gates



Failures of basic elements are **independent**

Reduction rule II

Simplification of conflicting PAND gates



Subtrees could fail **simultaneously**

Rewriting DFTs

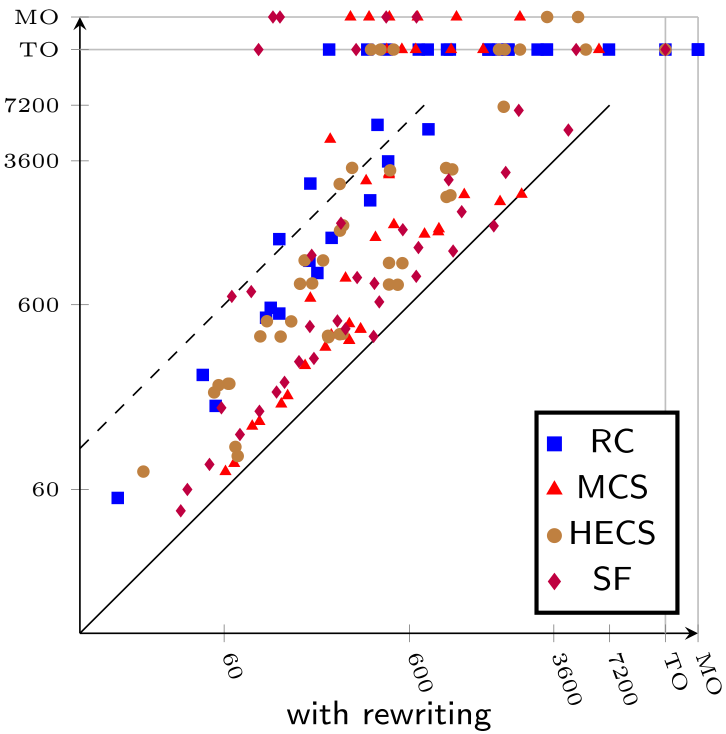
- Context-sensitive rewrite rules
- 29 rule families:
    - flattening of AND, OR, PAND
    - conflicting PAND gates
    - simplifying FDEP gates
    - …
- Fully automated graph rewriting

Experimental evaluation

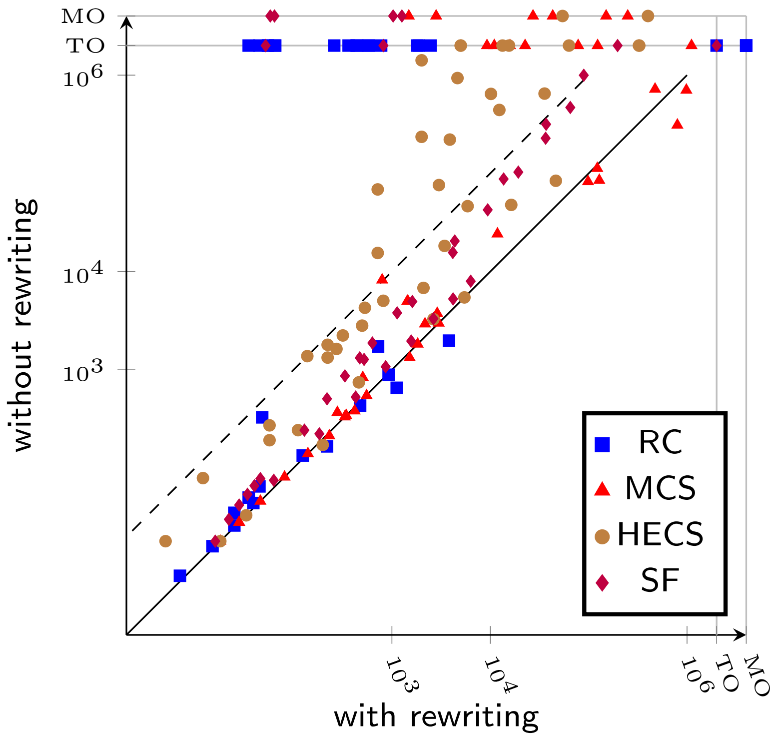
[Junges et al., Formal Aspects of Computing 2017]

- Analysis with DFTCalc
- could solve 27% more examples with rewriting

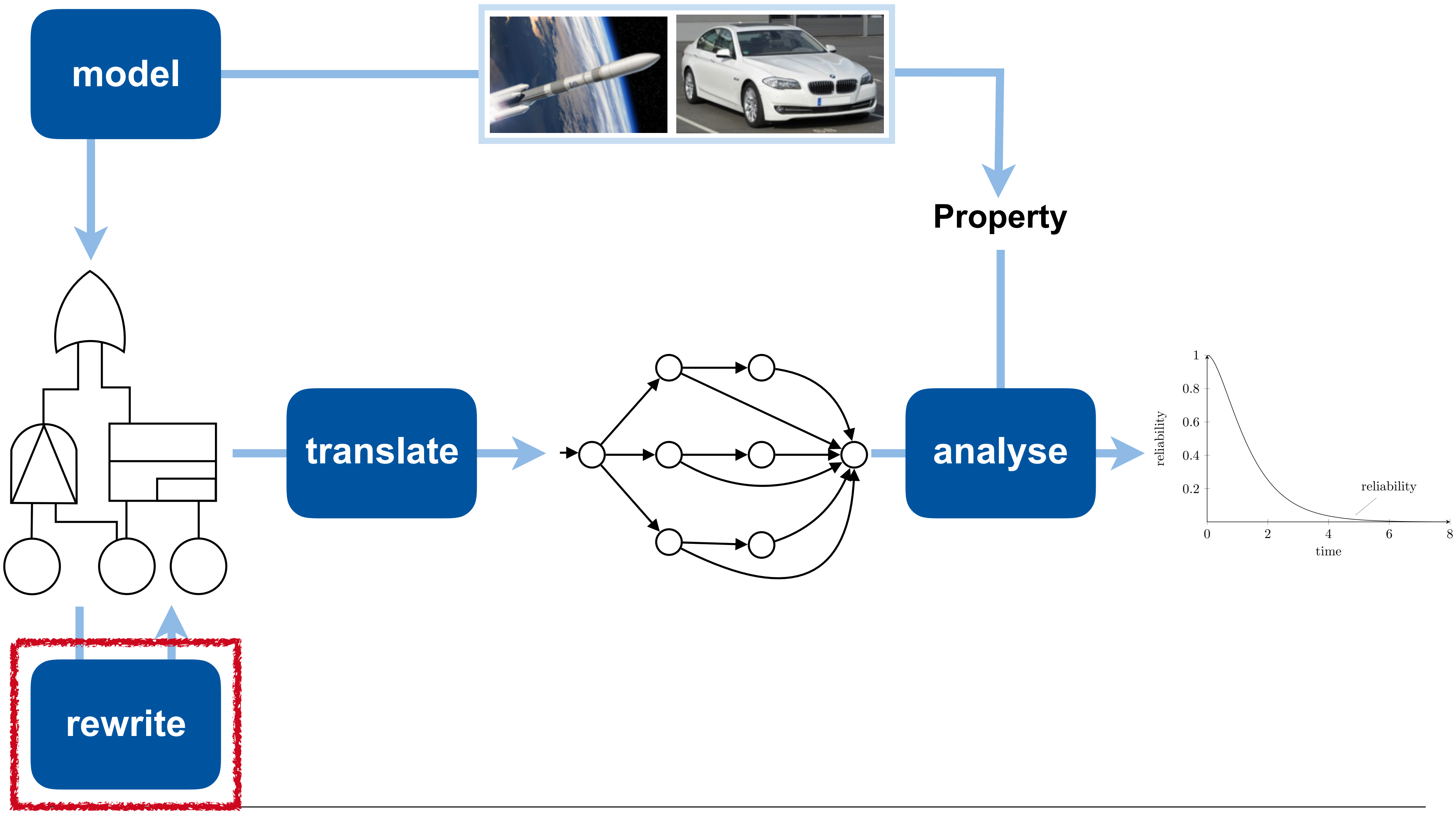
Time



State space

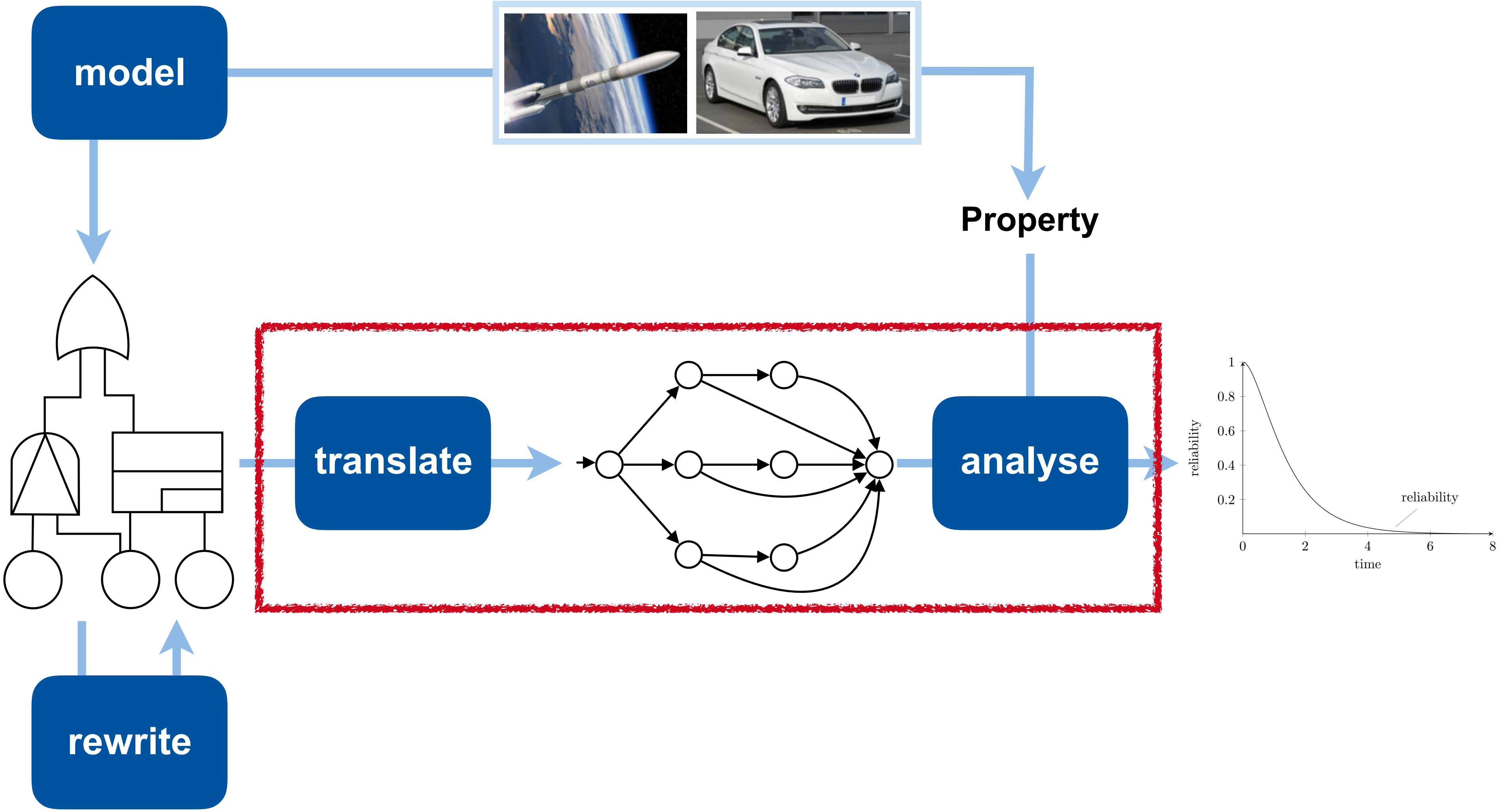


Overview

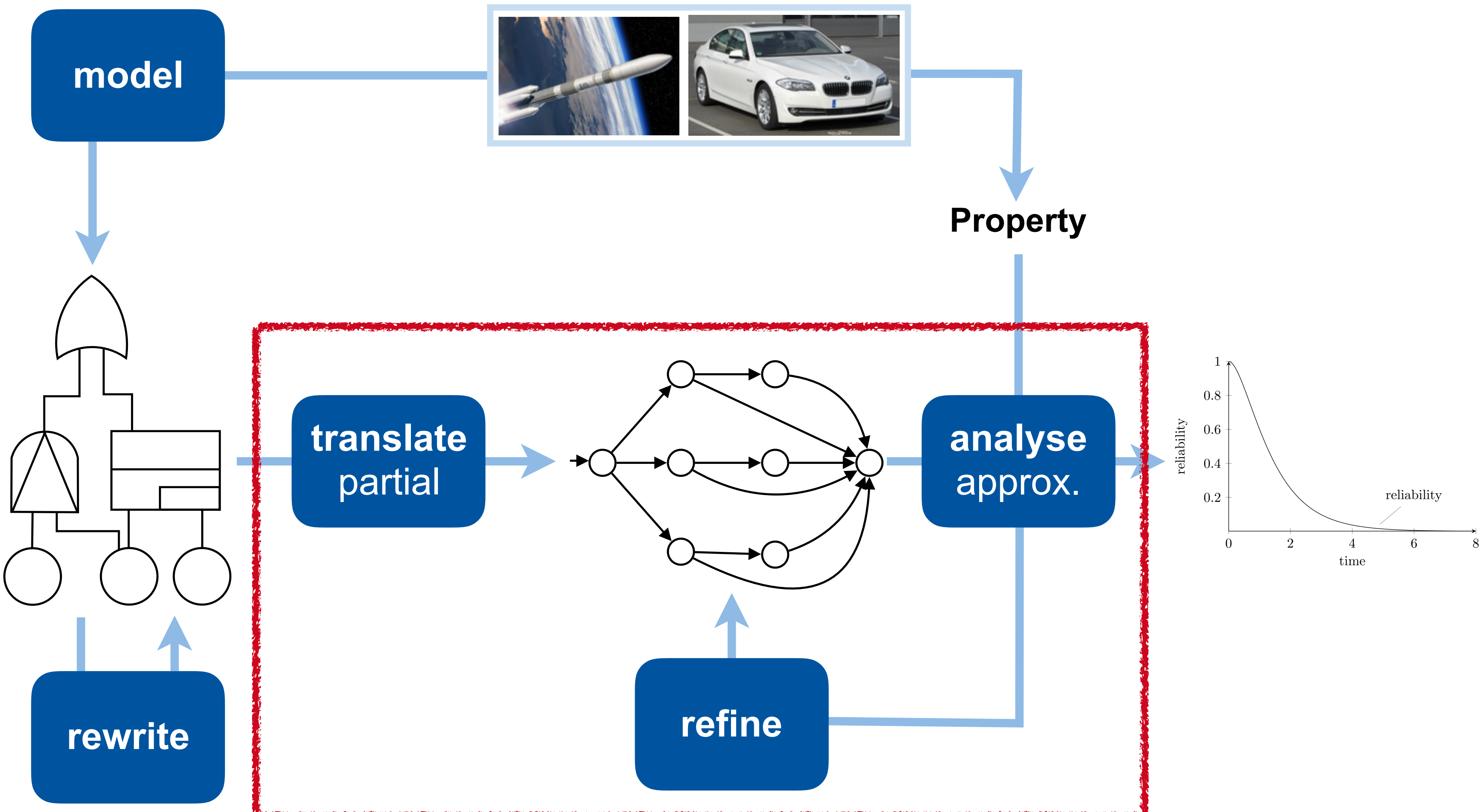


model
Property
translate
analyse
rewrite
1
0.8
0.6
0.4
0.2
reliability
reliability
0
2
4
6
8
time

Overview


model
Property
translate
analyse
rewrite
1
0.8
0.6
0.4
0.2
reliability
0
2
4
6
8
time
reliability

Overview



model
Property
translate partial
analyse approx.
refine
rewrite
reliability
1
0.8
0.6
0.4
0.2
0
2
4
6
8
time

Approximation

[Volk et al., IEEE TII 2018]

Problem:
state space might still be **large**

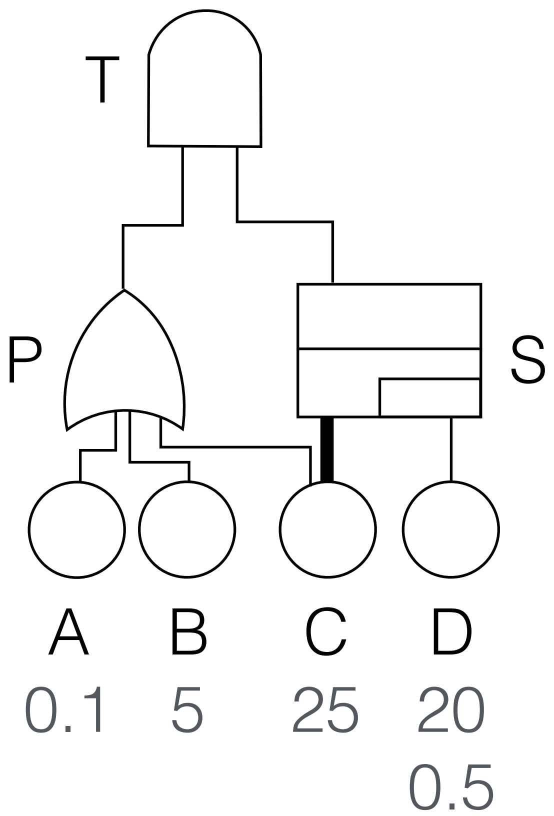
Observations:

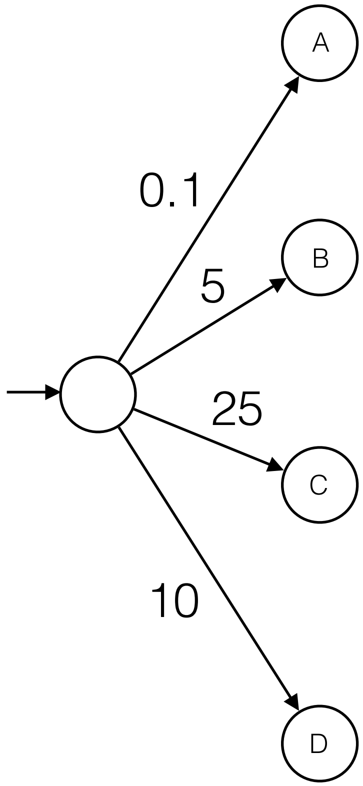
1. **Exact** solution often not necessary:
   e.g. ISO 26262 for automobiles requires analysis of 2 independent failures
   ➡ consider only **paths of length 2**
2. **Differences in rates** in orders of magnitude

Solution:

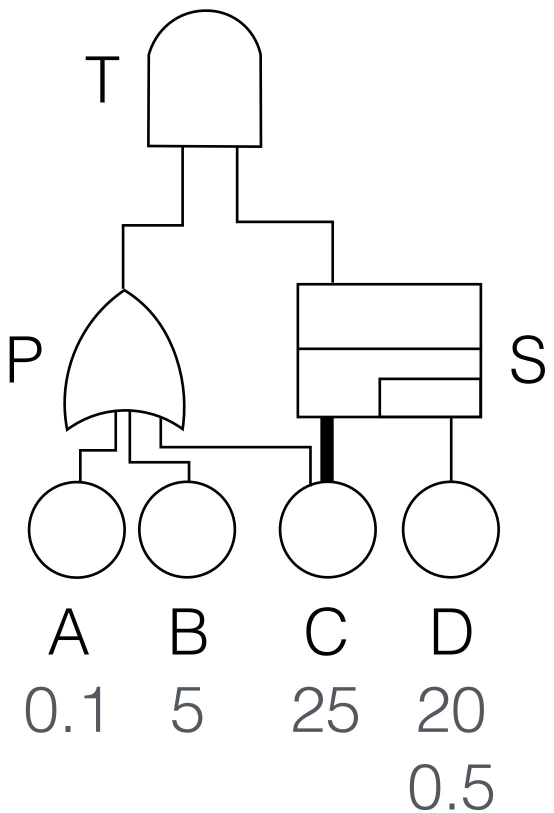
➡ **Approximate result**:
- build only **parts of state space**
- give **over-** and **under-approximation** of exact result

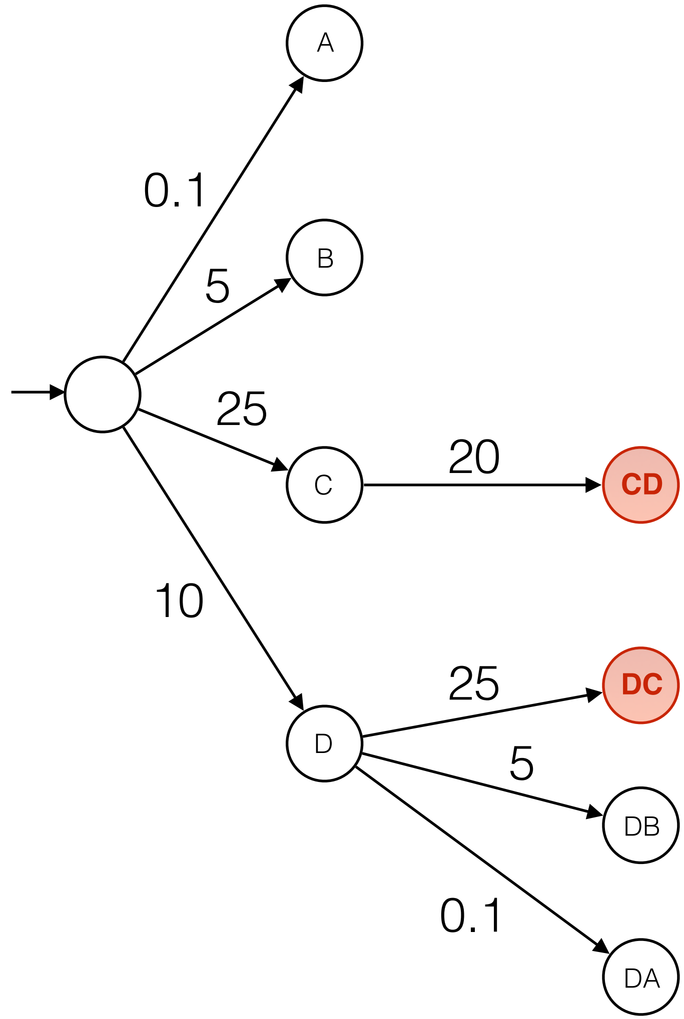
Approximation idea



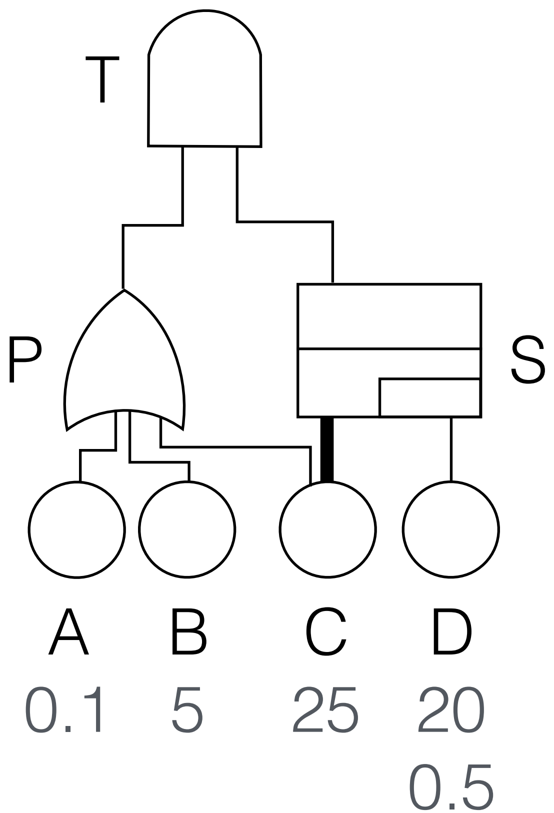


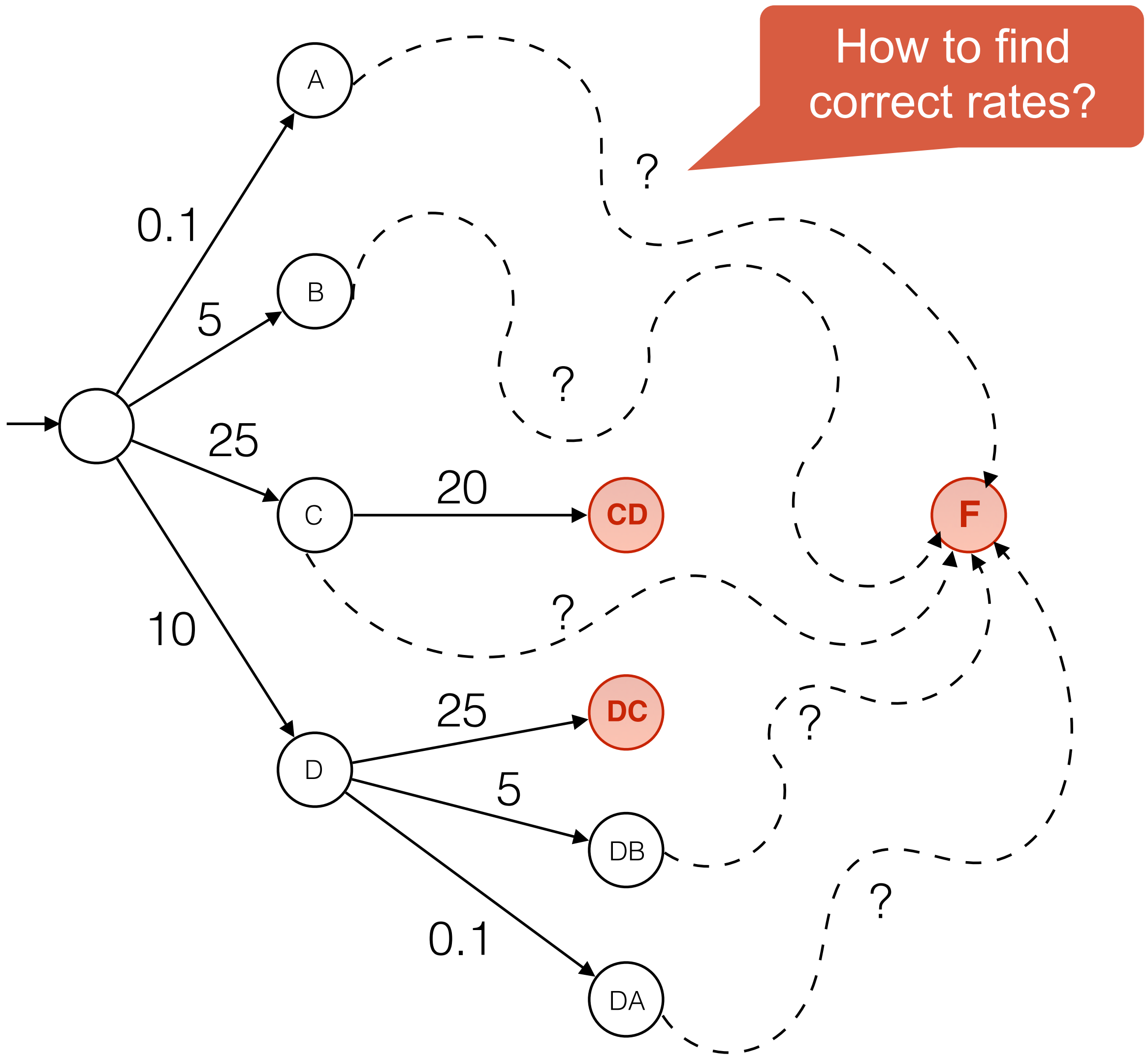
Approximation idea





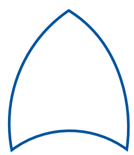
Approximation idea

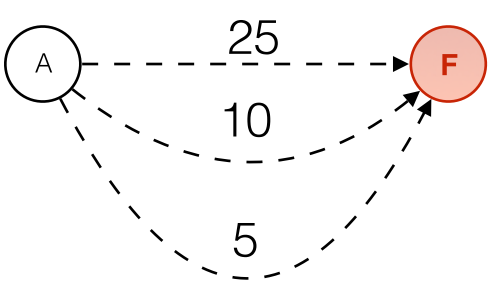




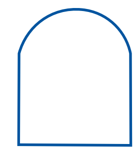
Under and over approximation for MTTF

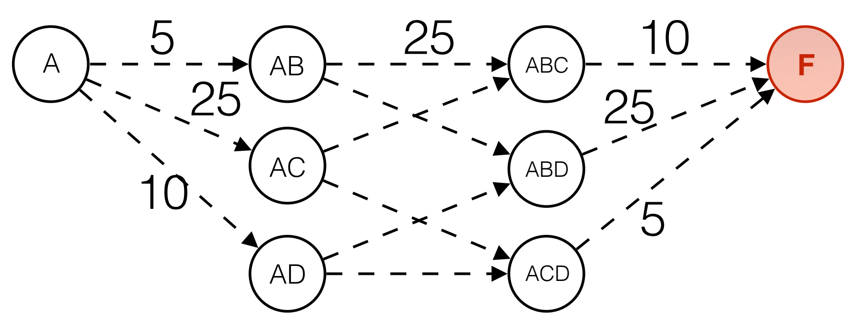
- **Under approximation:**
  - next BE leads to complete failure



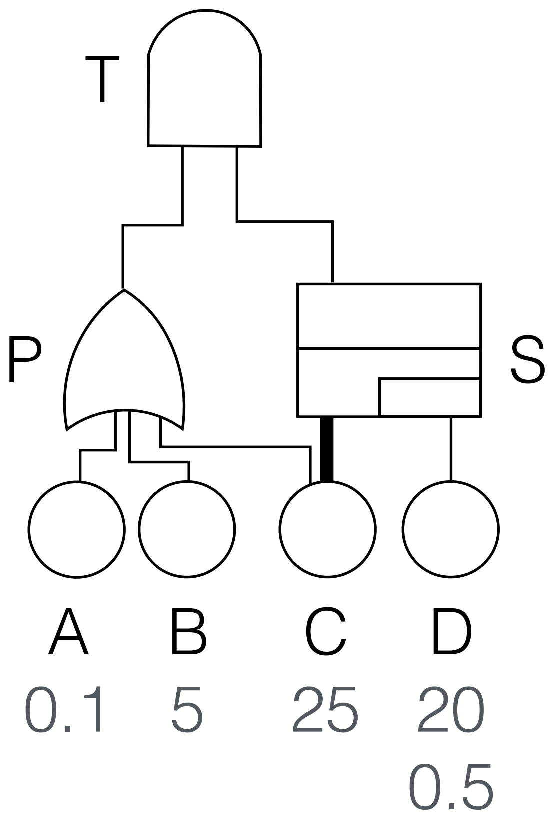


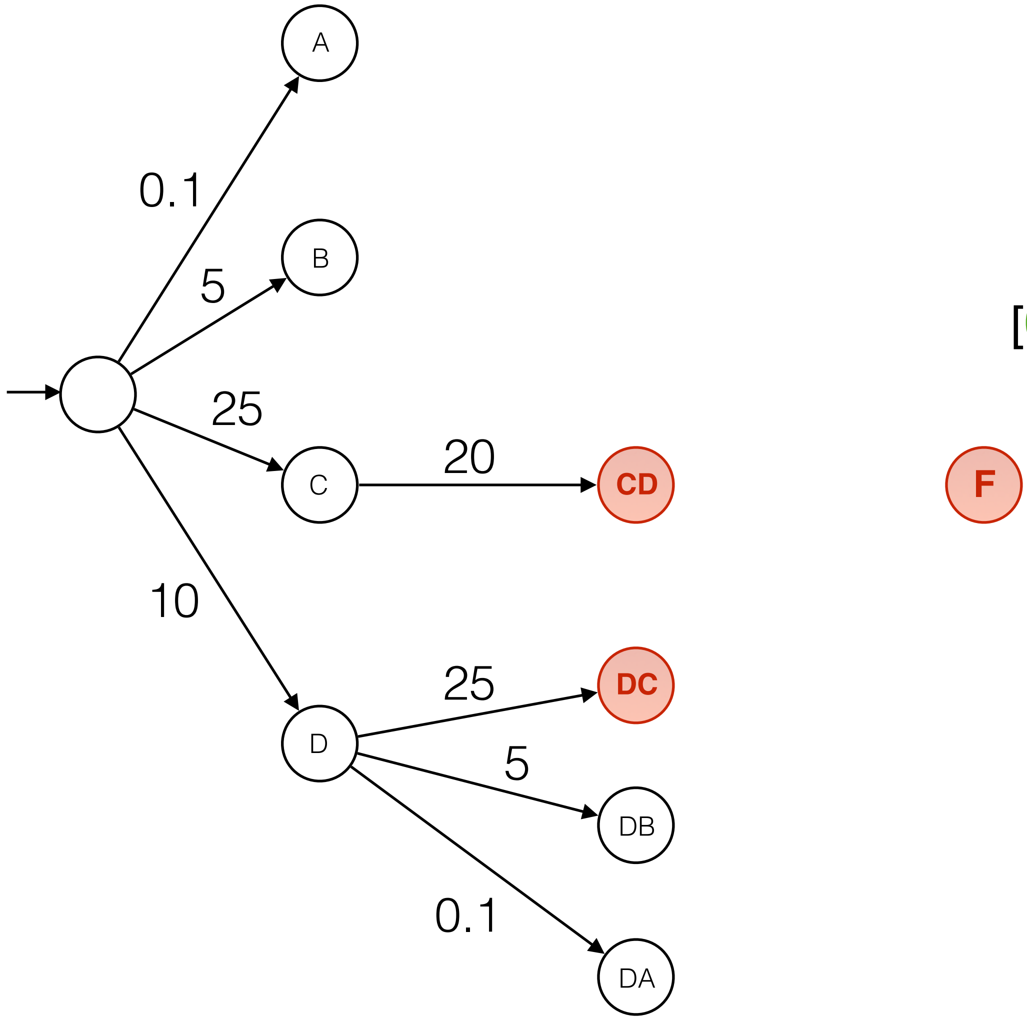
- **Over approximation:**
  - complete failure only if all remaining BEs failed





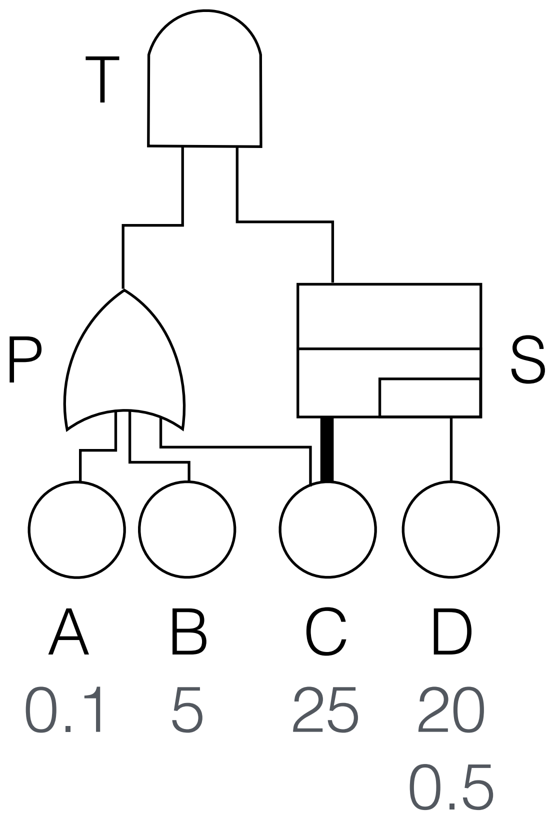
Approximation algorithm

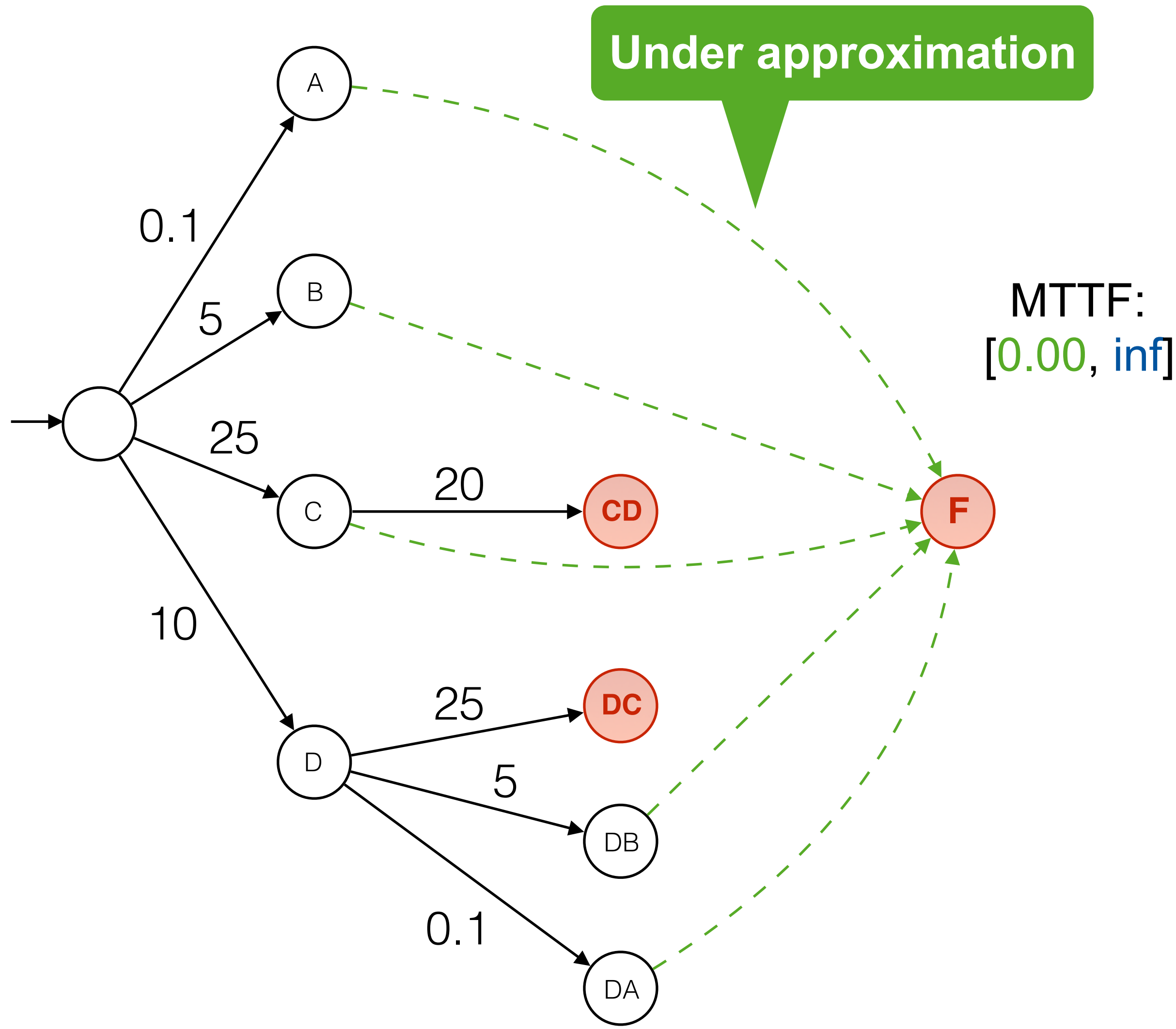




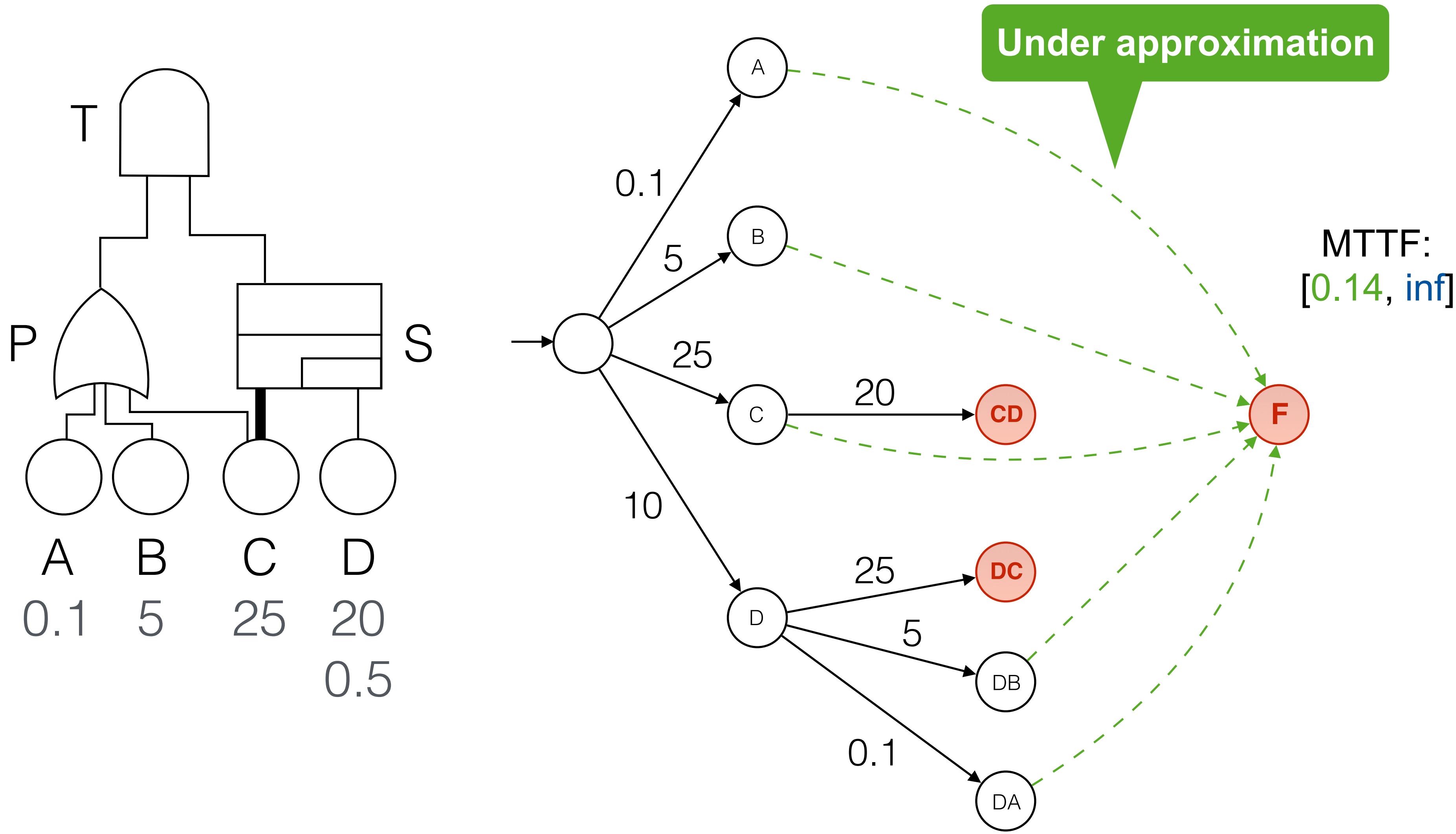
MTTF:
[0.00, inf]

Approximation algorithm

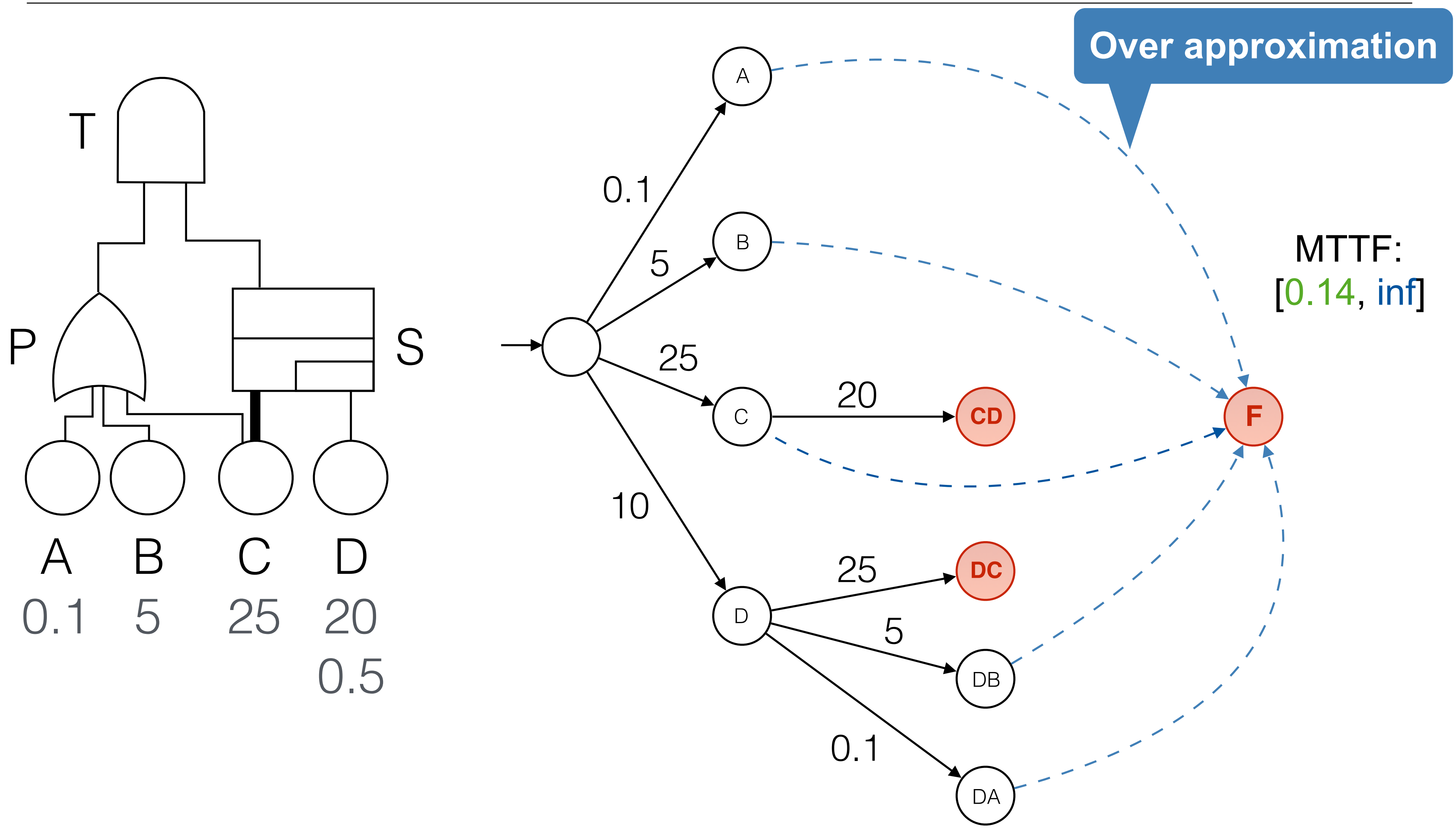




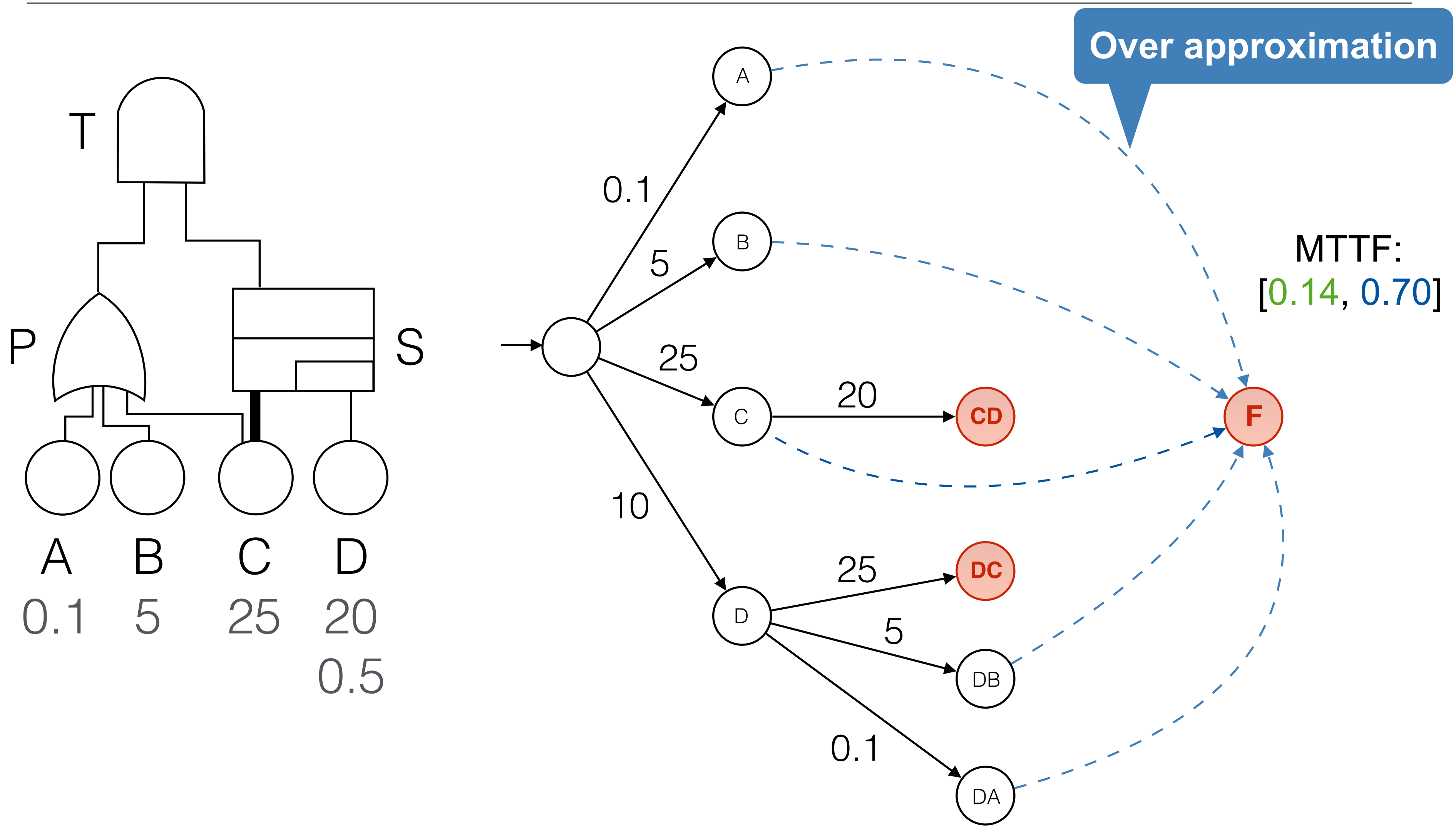
Approximation algorithm



Approximation algorithm

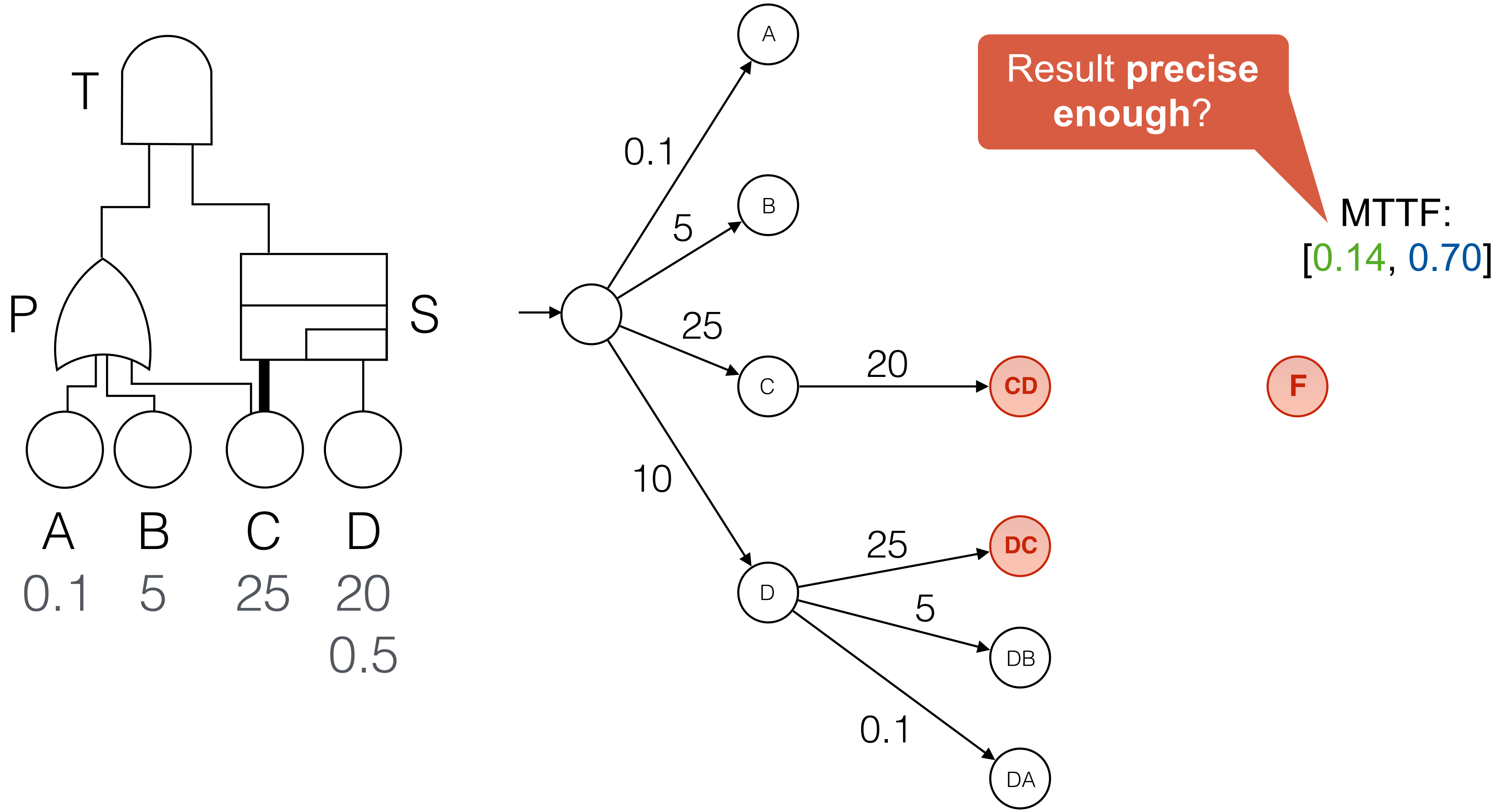


Approximation algorithm

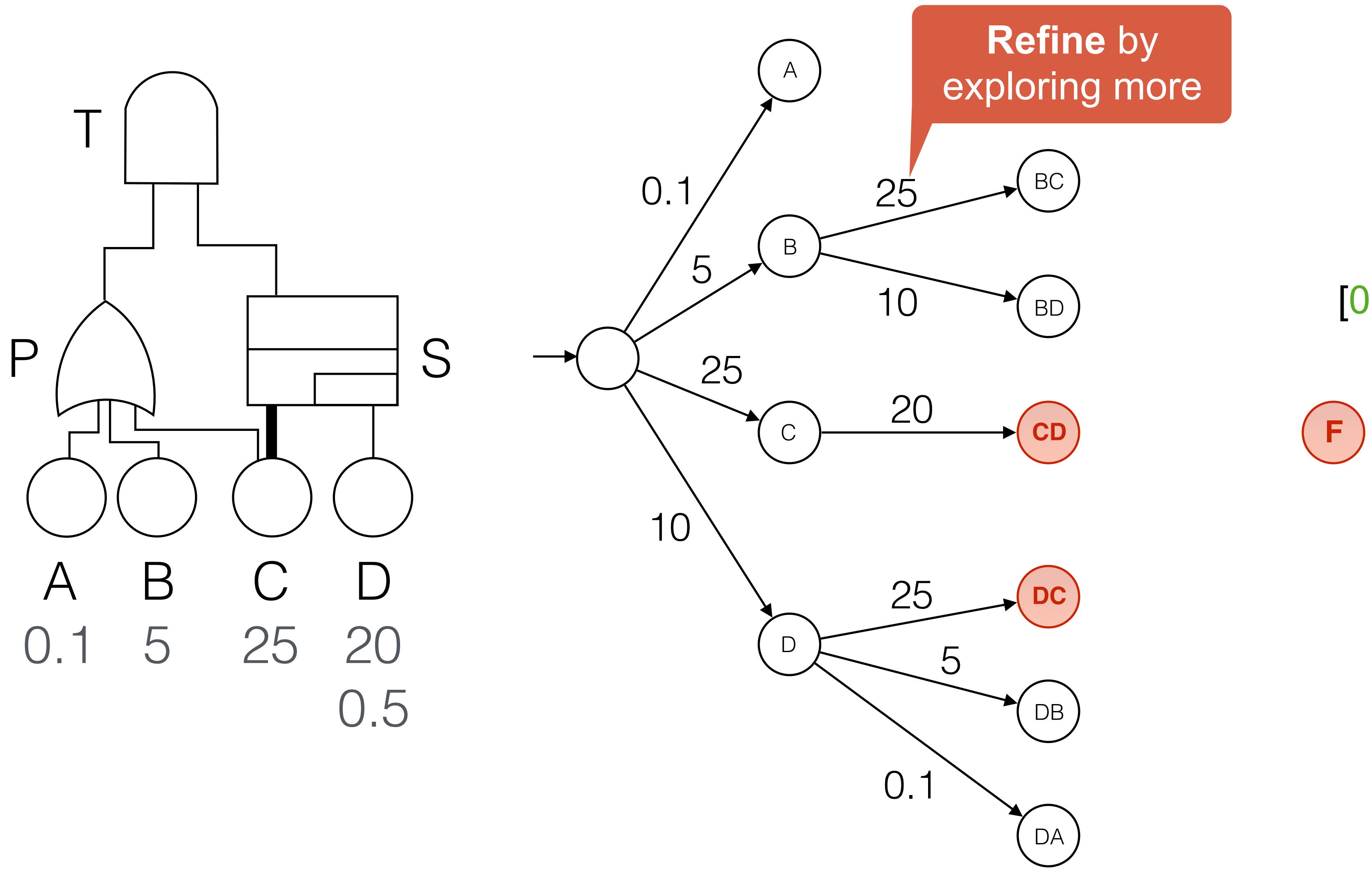


MTTF:
[0.14, 0.70]

Approximation algorithm

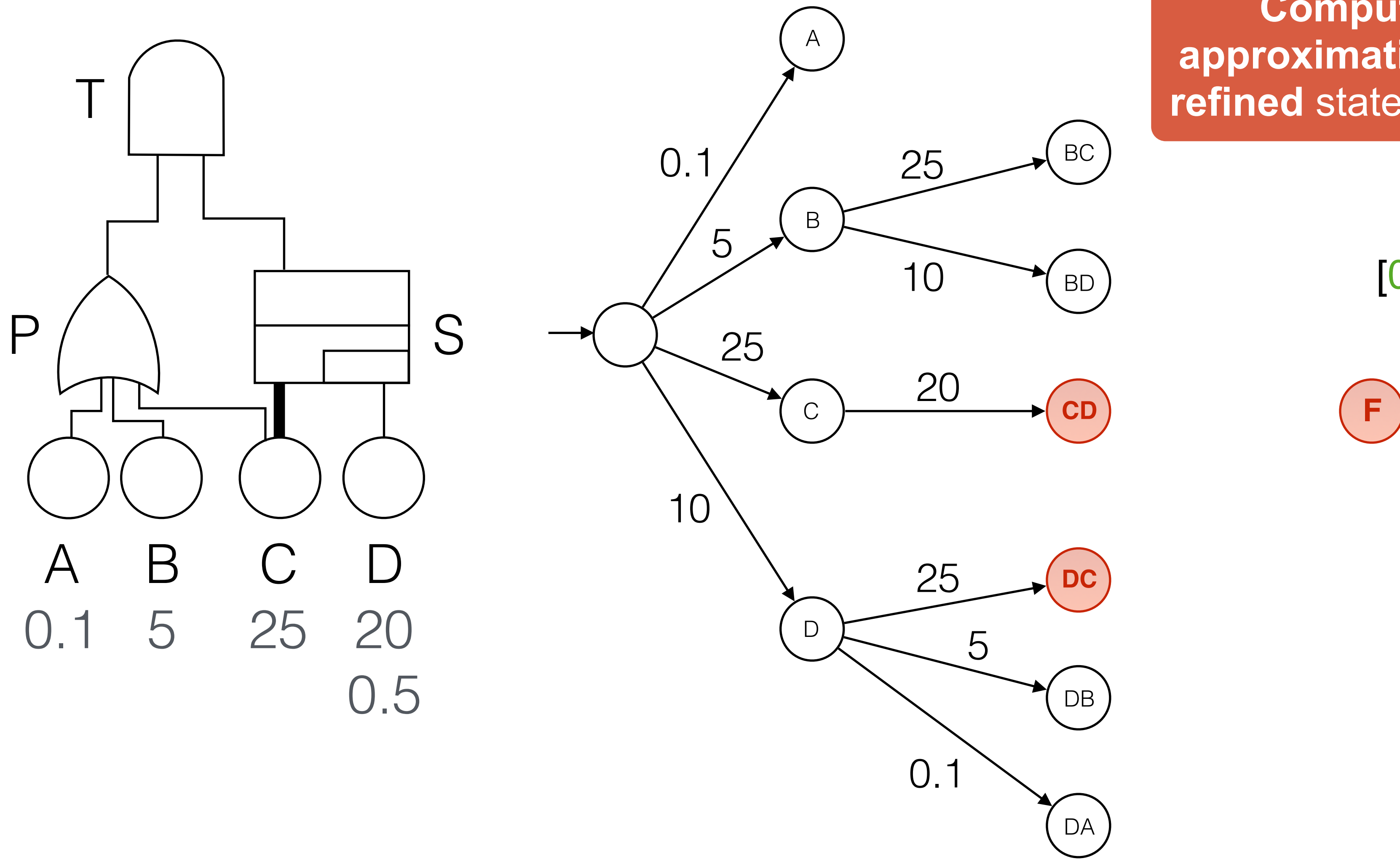


Approximation algorithm



MTTF: [0.14, 0.70]

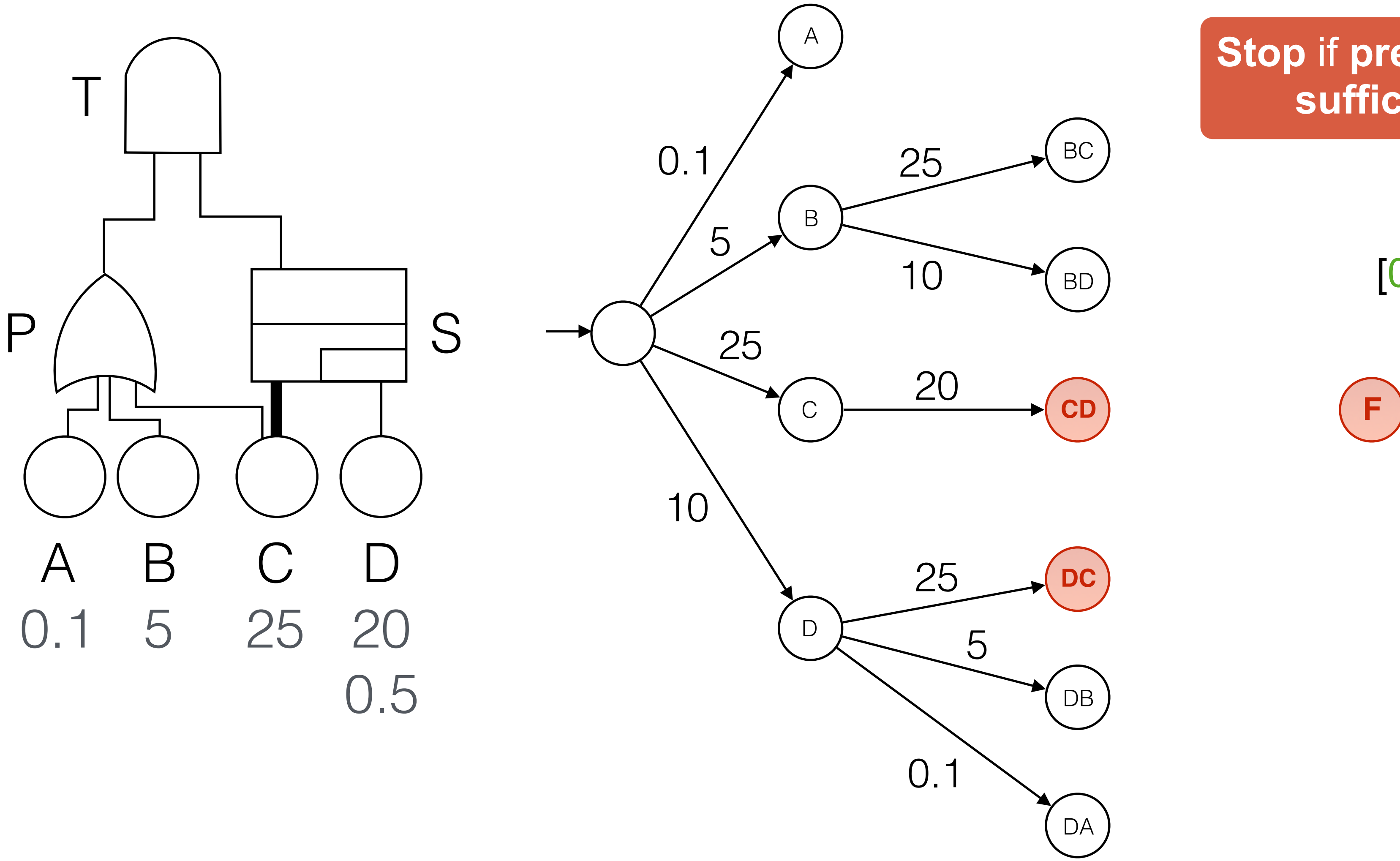
Approximation algorithm



Compute **approximation** on **refined** state space

MTTF: [0.20, 0.32]

Approximation algorithm



Stop if precision suffices

MTTF:
[0.27, 0.28]

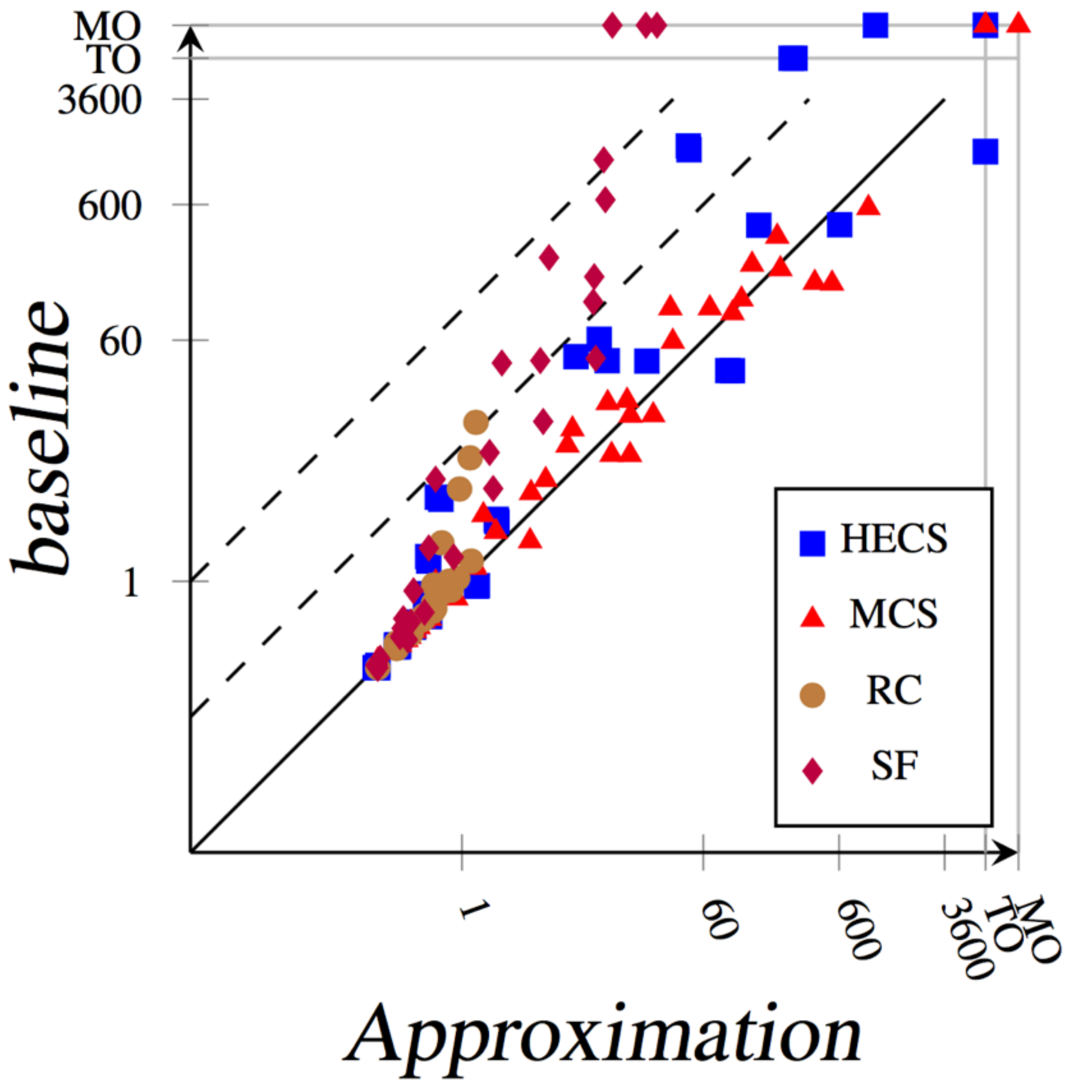
Approximation algorithm

- Approximation applicable for reliability and MTTF
- Optimizations still applicable
- Implemented in Storm
- Fully automated for given precision

  but extendable to iterative computation with user feedback

Analysis run times: Approximation

[Volk et al., IEEE TII 2018]

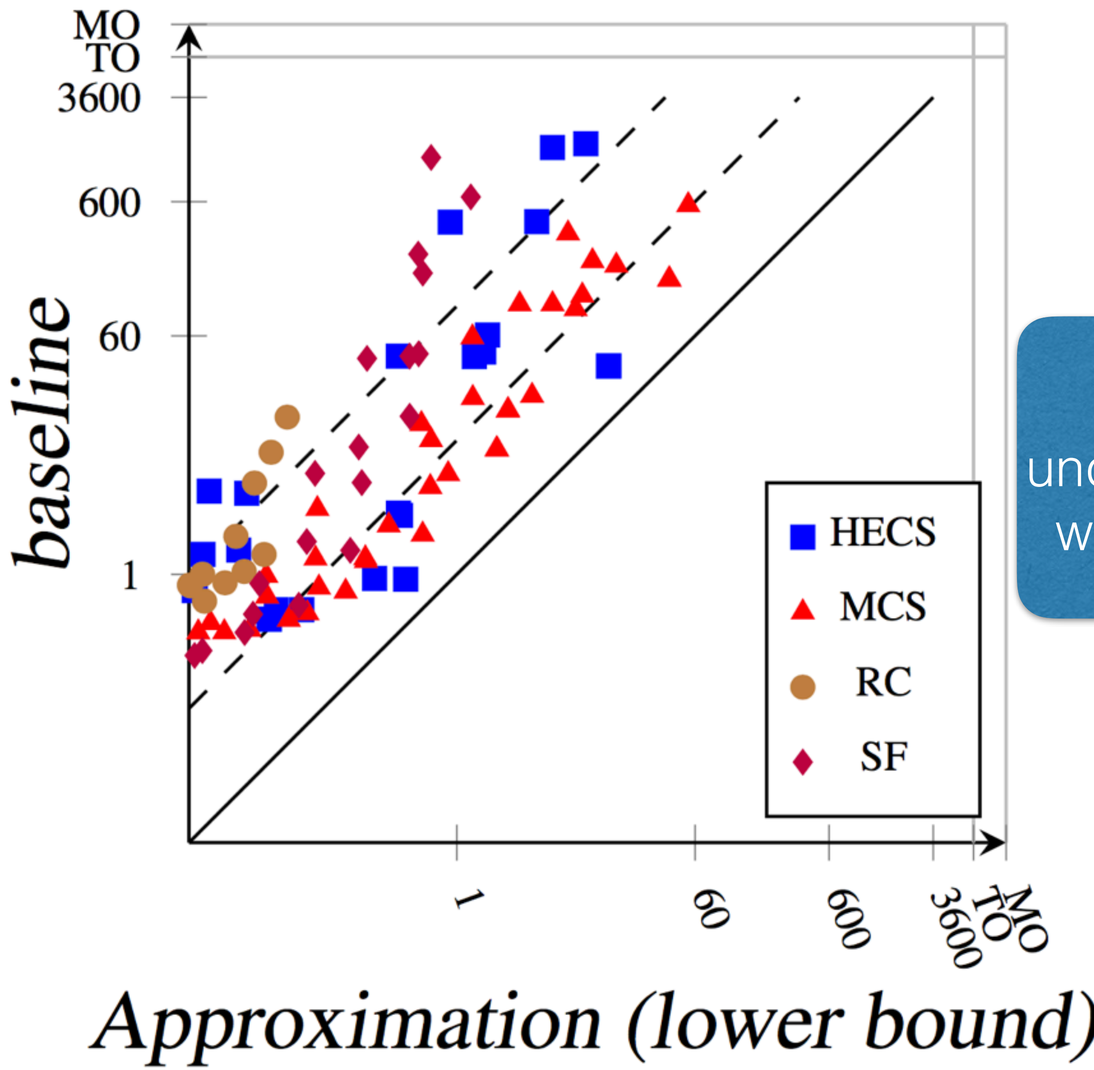


Precision 10%

u-l < 0.1 (u+l)/2

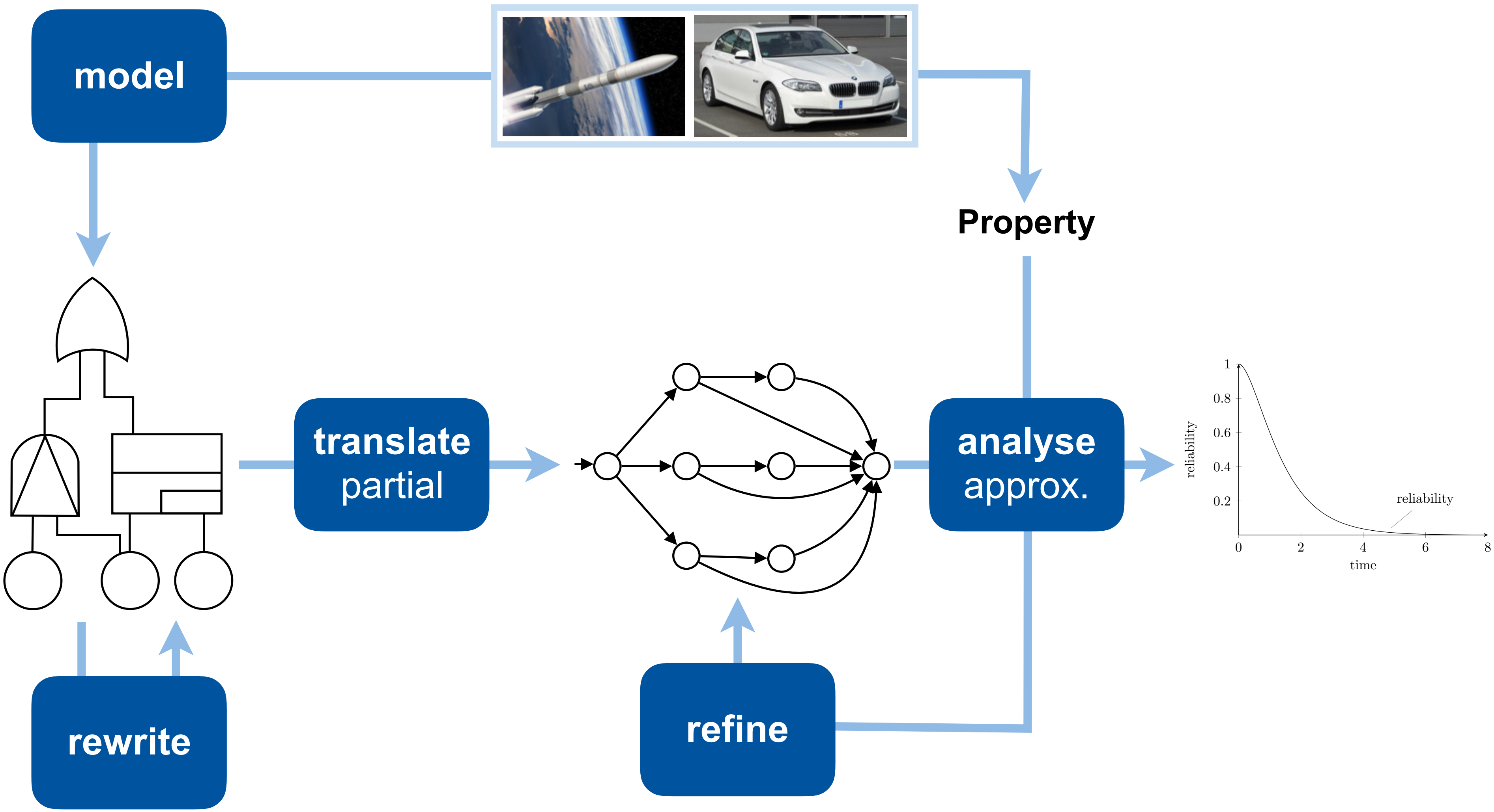
Analysis run times: Only under approximation

[Volk et al., IEEE TII 2018]



Time until under approximation was **95% of MTTF**

Overview


model
Property
translate
partial
analyse
approx.
rewrite
refine
reliability
time
reliability
1
0.8
0.6
0.4
0.2
0
2
4
6
8

A Modern Perspective on Fault Tree Analysis

Joost-Pieter Katoen and Matthias Volk

Joint work with: Majdi Ghadhab (BMW), Dennis Guck (TWT), Sebastian Junges (RWTH), Matthias Kuntz (BMW), Enno Ruijters (U. Twente) and Mariëlle Stoelinga (U. Twente)

Tutorial MMB 2018, Erlangen, BY

Roadmap of This Tutorial

Part 1. What are Dynamic Fault Trees?

- DFT Elements, Benchmarks, Intricacies, DFTs as Stochastic Petri Nets

Part 2. From DFTs to Markov Models, Compositionally

- Compositional State-Space Minimisation, Non-Determinacy

Part 3. From DFTs to Markov Models, Monolithically

- Symmetry Reduction, Don't Care Propagation

Part 4. DFT Analysis by Model Checking

- Reliability Measures, Core Algorithms, `Storm` Tool

Part 5. Advanced Optimisations

- Graph Rewriting, Partial State-Space Generation

Part 6. Industrial Applications and Outlook

Focus is on conveying intuition and experimental results

Overview

BMW case study [Ghadhab *et al.*, 2017]

- Topic: Design-phase safety analysis of autonomous vehicle guidance
  - ASIL[1] D, i.e., 10^{-8} residual hardware failures per hour
  - Fail-operational: continue to operate a while after component failure
- Inputs:
  - Functional blocks: environment perception, trajectory planning, etc.
  - Safety concepts: TMR, nominal+safety path, main+fall-back path
  - Hardware architectures: for different safety concepts
- Outputs: which function-2-hardware mapping yields optimal safety?
- Approach:
  - Generate Dugan's dynamic fault trees
  - Analyse them using probabilistic model checking
- Analysis outcomes

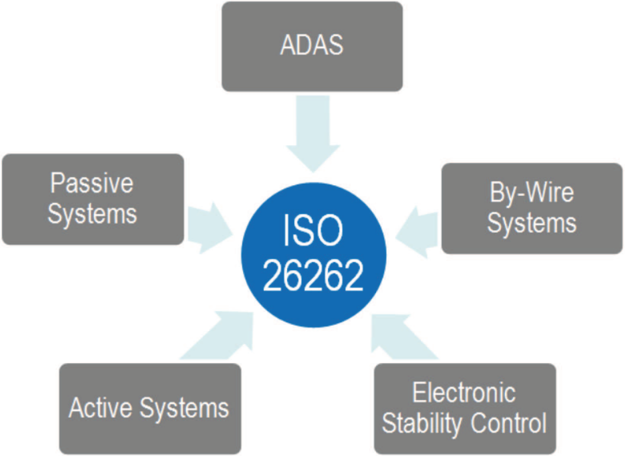
[1] Automotive Safety Integrity Level

Autonomous Vehicle Guidance

Major safety goal: avoid wrong vehicle guidance.

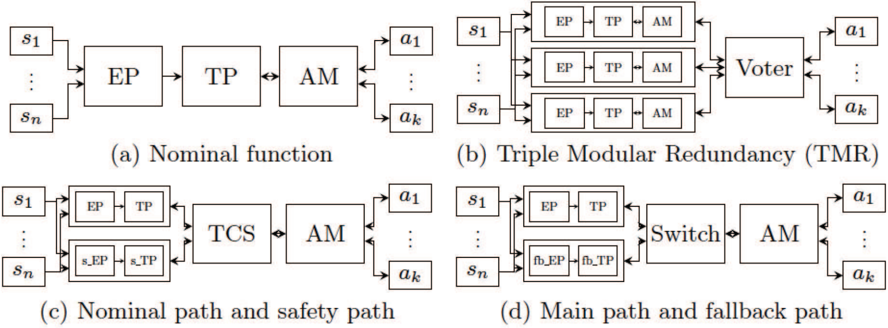
Automotive Safety Integrity Level D, i.e., 10^{-8} residual hardware failures per hour

Fail Operational



Fail-operational: continue to operate a while after component failure

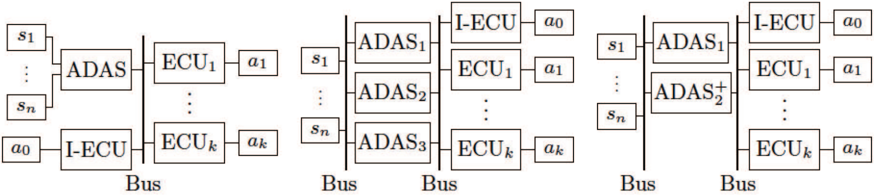
Functional Blocks+Safety Concepts



Fail-operational design patterns for autonomous driving.

EP = Environment Perception, TP = Trajectory Planning
AM = Actuator Mgt, TCS = Trajectory Checking and Selection

Sample Car Architectures



(a) E/E architecture A (b) E/E architecture B (c) E/E architecture C

(a) nominal, (b) "TMR", and (c) ADAS+ architecture.

Assumption: during a transient fault, no other faults occur (conform ISO 26262)

ADAS = Advanced Driver Assistance System, I-ECU = Integration ECU

Sample Safety Metrics

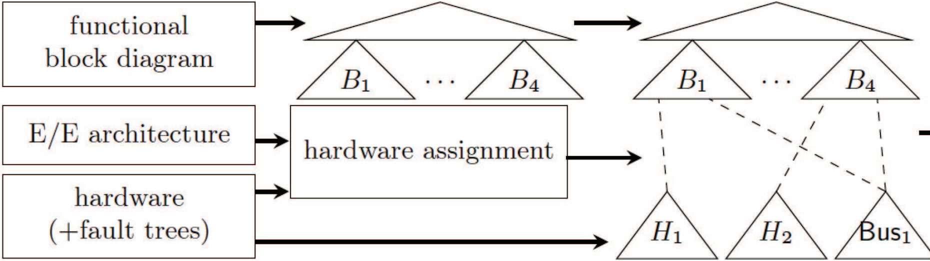
System integrity ≈ probability of safe operation during operational lifetime

1. How probable is it that the system is fully functional at time t?
2. What is the fraction of system failures w/o being degraded first?
3. The expected time to failure upon becoming degraded?
4. Criticality: how likely is it to fail within a drive cycle once degraded?
5. System integrity when limiting operational time after degradation?

Phrasing in Temporal Logic

	Measure	Model Checking Queries
System	integrity	$1 - P(\Diamond^{\leq t} \text{ failed})$
	FIT	$\frac{1}{\text{lifetime}} \cdot \left(1 - P(\Diamond^{\leq \text{lifetime}} \text{ failed})\right)$
	MTTF	$\mathsf{ET}(\Diamond \text{ failed})$
Degradation	FFA	$1 - P\left(\Diamond^{\leq t} (\text{failed} \vee \text{degraded})\right)$
	FWD	$P\left((\neg\text{degraded}) \mathsf{U}^{\leq t} (\neg\text{degraded} \wedge \text{failed})\right)$
	MTDF	$\Sigma_{s \in \text{degraded}}\left(P(\neg\text{degraded} \mathsf{U} s) \cdot \mathsf{ET}^{s}(\Diamond \text{ failed})\right)$
	MDR	$\text{argmin}_{s \in \text{degraded}}\left(1 - P^{s}(\Diamond^{\leq t} \text{ failed})\right)$
	SILFO	$1 - \left(FWD + \Sigma_{s \in \text{degraded}}\left(P(\neg\text{degraded} \mathsf{U}^{\leq t} s) \cdot P^{s}(\Diamond^{\leq \text{drivecycle}} \text{ failed})\right)\right)$

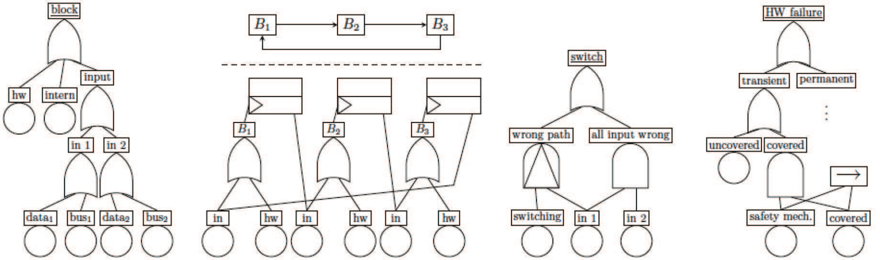
Fault Tree Generation



Three-level fault trees: (1) system, (2) block, and (3) HW level

- Why DFTs?
  - Warm+cold redundancies, spare components, state-dependent faults
- Communication via fallible buses, depending on HW assignment

Sample DFTs for BMW Case Study



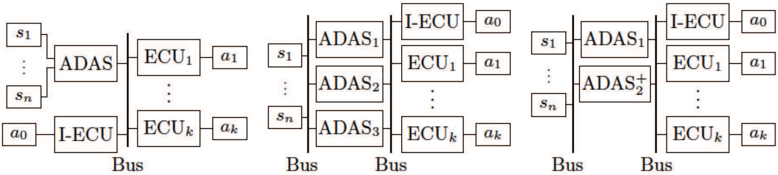
(a) Block FT (b) Feedback loop and FT (c) Switch FT (d) Hardware FT

SPARE gates are used for modelling cold stand-by of fall-back paths

Case Study Characteristics

	Scenario					DFT			CTMC		
	SC	Arch.	Adap.	Sens.	Act.	#BE	#Dyn.	#Elem.	#States	#Trans.	Degrad.
I	SC1	B	—	2/4	4/4	76	25	233	5,377	42,753	—
II	SC2	B	—	2/4	4/4	70	23	211	5,953	50,049	19.35%
III	SC2	C	ADAS+	2/4	4/4	57	19	168	1,153	7,681	16.65%
IV	SC3	C	—	2/4	4/4	57	21	170	385	1,985	12.47%
V	SC2	A	—	2/4	4/4	58	19	185	193	897	0.00%
VI	SC2	B	w/o I-ECU	2/4	4/4	65	21	199	1,201	8,241	19.98%
VII	SC2	B	5 ADAS	2/8	7/7	96	30	266	$2\ 10^5$	$2\ 10^6$	19.35%
VIII	SC2	B	8 ADAS	6/8	7/7	114	36	305	$4\ 10^6$	$66\ 10^6$	10.90%

SC1 = TMR, SC2 = nominal and safety path, SC3 = main and fall-back path



(a) E/E architecture A (b) E/E architecture B (c) E/E architecture C

(A) nominal, (B) "TMR", and (C) ADAS+ architecture.

Case Study Characteristics

	Scenario					DFT			CTMC		
	SC	Arch.	Adap.	Sens.	Act.	#BE	#Dyn.	#Elem.	#States	#Trans.	Degrad.
I	SC1	B	—	2/4	4/4	76	25	233	5,377	42,753	—
II	SC2	B	—	2/4	4/4	70	23	211	5,953	50,049	19.35%
III	SC2	C	ADAS+	2/4	4/4	57	19	168	1,153	7,681	16.65%
IV	SC3	C	—	2/4	4/4	57	21	170	385	1,985	12.47%
V	SC2	A	—	2/4	4/4	58	19	185	193	897	0.00%
VI	SC2	B	w/o I-ECU	2/4	4/4	65	21	199	1,201	8,241	19.98%
VII	SC2	B	5 ADAS	2/8	7/7	96	30	266	$2\ 10^5$	$2\ 10^6$	19.35%
VIII	SC2	B	8 ADAS	6/8	7/7	114	36	305	$4\ 10^6$	$66\ 10^6$	10.90%

#BE = the number of basic events (aka: leaves) in the DFT

#Dyn. = the number of dynamic gates in the DFT

#Elem. = the total number of elements in the DFT

Case Study Characteristics

	Scenario					DFT			CTMC		
	SC	Arch.	Adap.	Sens.	Act.	#BE	#Dyn.	#Elem.	#States	#Trans.	Degrad.
I	SC1	B	—	2/4	4/4	76	25	233	5,377	42,753	—
II	SC2	B	—	2/4	4/4	70	23	211	5,953	50,049	19.35%
III	SC2	C	ADAS+	2/4	4/4	57	19	168	1,153	7,681	16.65%
IV	SC3	C	—	2/4	4/4	57	21	170	385	1,985	12.47%
V	SC2	A	—	2/4	4/4	58	19	185	193	897	0.00%
VI	SC2	B	w/o I-ECU	2/4	4/4	65	21	199	1,201	8,241	19.98%
VII	SC2	B	5 ADAS	2/8	7/7	96	30	266	$2\ 10^5$	$2\ 10^6$	19.35%
VIII	SC2	B	8 ADAS	6/8	7/7	114	36	305	$4\ 10^6$	$66\ 10^6$	10.90%

Degrad. = fraction of degraded states in the DFT's Markov chain

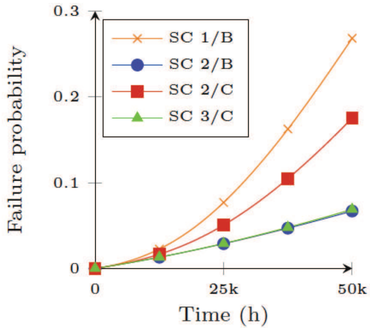
Timings

	I	II	III	IV	V	VI	VII	VIII
Model generation	1.02 s	1.02 s	0.38 s	0.33 s	0.34 s	0.40 s	25.13 s	632.89 s
System + FFA + FWD	0.02 s	0.02 s	0.00 s	0.00 s	0.00 s	0.00 s	1.46 s	46.67 s
MTDF	—	2.67 s	0.18 s	0.03 s	0.02 s	0.20 s	2892.42 s	>3600 s
MDR	—	0.60 s	0.11 s	0.02 s	0.02 s	0.11 s	26.07 s	781.93 s
SILFO	—	1.83 s	0.17 s	0.04 s	0.02 s	0.18 s	1694.91 s	>3600 s

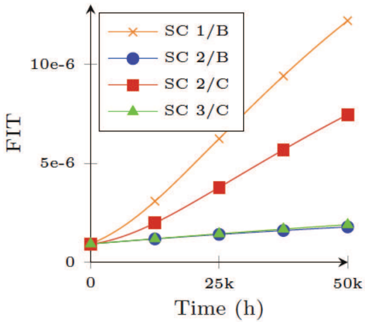
Computing MTDF and SILFO are computationally intensive

Partial state-space exploration for VIII of ≈ 10% yields bounds of 3% error in 22s.

Analysis Results

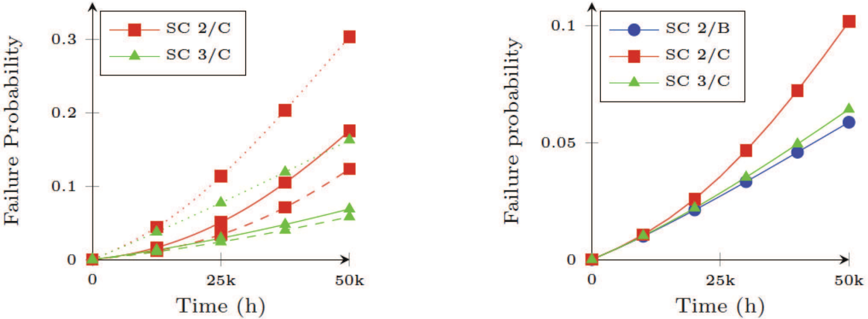


(a) Probability of failure



(b) FIT

Analysis Results



(c) Sensitivity analysis

(d) SILFO

Sensitivity is investigated by varying the failure rates

SILFO = System Integrity under Limited Fail-Operation

Model Checking Boosts FT Analysis

- More safety measures
- Larger classes of DFTs can be analysed
- Typically substantially faster than classical FT analysis
- Abstraction aggravates this further:
  - tailored simplification for FTs + abstraction–refinement on FTs

  yields several orders of magnitude improvements.
- Full automation

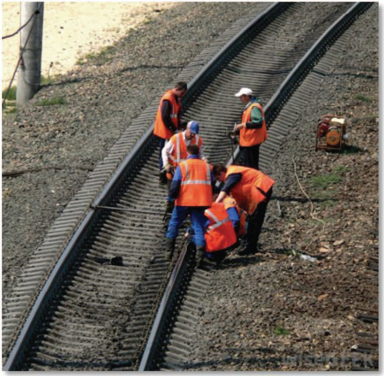
Try it out yourself: `stormchecker.org`

No myths.

Railway case studies

- Series of case studies with stakeholders from railway engineering
  - asset manager ProRail
  - rolling stock maintenance company NS/NedTrain, and
  - consultancy company Movares
- Focus: study of effect of maintencance strategies
- Property: trade-off between reliability and maintenance costs
- Based on extension of DFTs with simple maintenance
- Analysis using probabilistic and statistical model checking

Maintenance



Maintenance

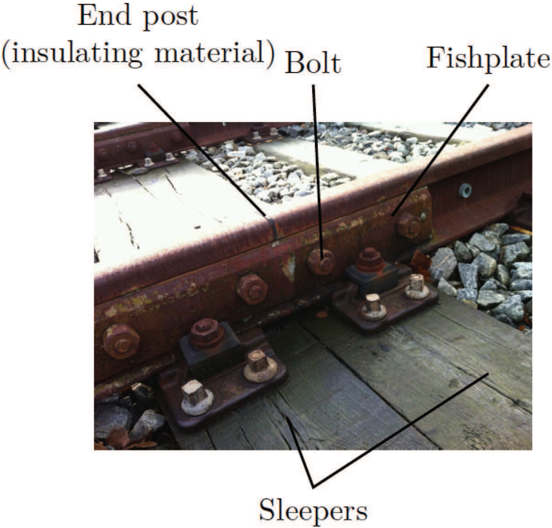
Types:

- Corrective maintenance
- Preventive maintenance

Strategies:

- Age-based
- Use-based
- Condition-based

Analysing Electrically Insulated Joint [Ruijters *et al.*, 2016]

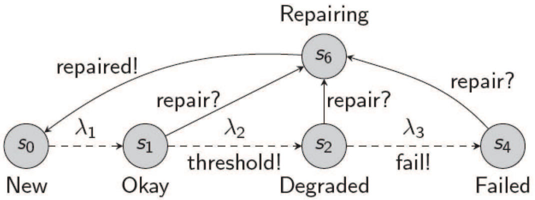


Maintenance in DFTs

- Many failures are not random events
  - Wear over time
  - Production faults
  - Caused by other failures
- Maintenance is essential for reliability
  - Reduce or prevent wear
  - Replace or repair worn components
  - Correct failures when they occur
- Maintenance is not a first-class citizen in DFTs

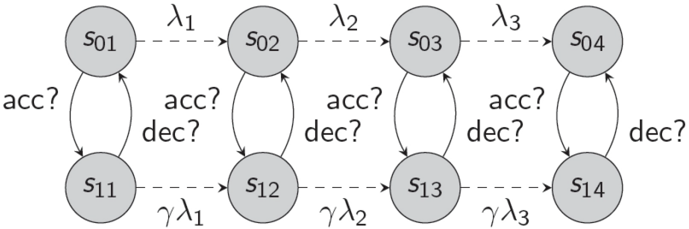
DFTs with Maintenance

- Idea: equip BEs with several degradation stages
- Maintenance := timed automaton with degradation stages
- Signals for composition:
  - Maintenance threshold, Repair, and Failure
- Other modules will send/receive these signals

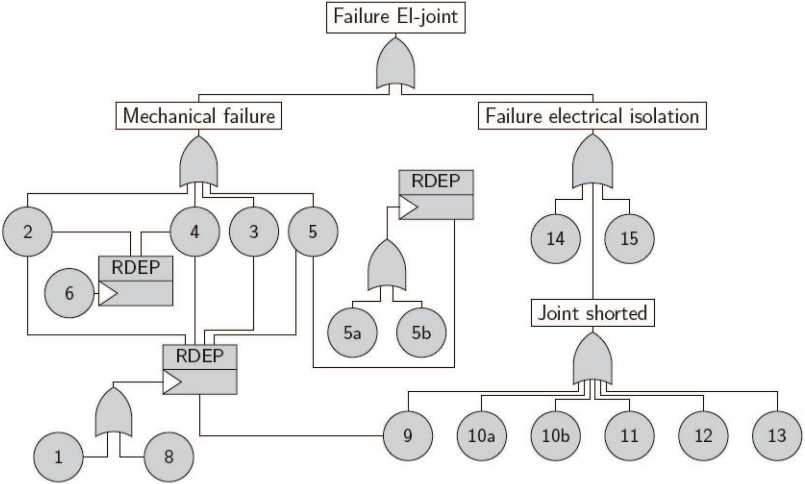


Rate-affecting failures

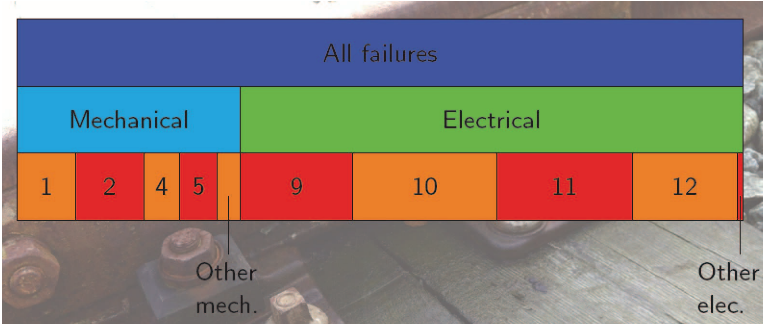
- Some failures accelerate wear of other components
- New variant on the FDEP gate: rate dependency (RDEP)
  - Failure of trigger BE accelerates degradation by factor γ
  - Repair of trigger BE does not repair triggered BE



DFT for the EI Joint



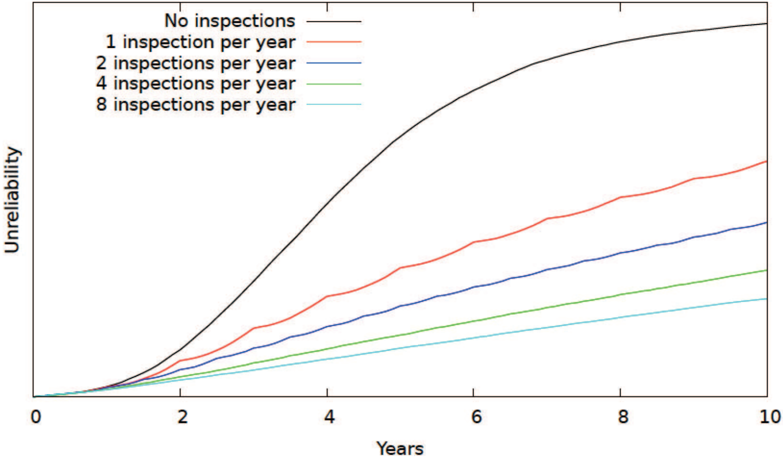
Failure Causes



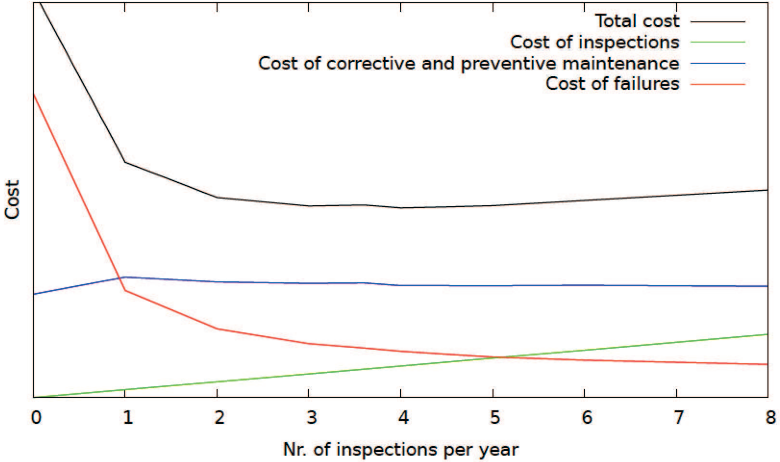
Parameters for the BEs for the EI-Joint

BE nr.	Failure mode	ETTF	ETTF NRG	Phases	Prob. cnd.
1	Poor geometry	5	5	4	10%
2	Broken fishplate	8	12	4	33%
3	Broken bolts	15	20	4	33%
4	Rail head broken out	10	15	4	33%
5	Glue connection broken	10	15	4	33%
5a	Manufacturing defect	-	-	-	0.25%
5b	Installation error	-	-	-	0.25%
6	Battered head	20	22	4	5%
7	Arc damage	1	1	3	0.2%
8	End post broken out	7	8	3	33%
9	Joint bypassed: overhang	5	8	4	100%
10a	Joint shorted: shavings (normal)	1	1	4	12%
10b	Joint shorted: shavings (coated)	10	10	4	3%
11	Joint shorted: splinters	200	200	1	100%
12	Joint shorted: foreign object	250	250	1	100%
13	Joint shorted: shavings (grinding)	5000	5000	1	100%
14	Sleeper shifted	5000	5000	1	100%
15	Internal insulation failure	5000	5000	1	100%
16	End post jutting out	20	20	1	100%

Unreliability



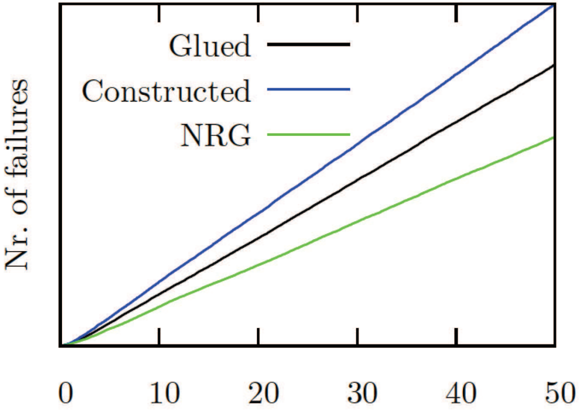
Analysis results: inspection rate



The new NRG joint

- Elongated fishplate to spread the stress on the plate
- Six bolts instead of four, to reduce flexing when a train drives over the joint
- Repositioned bolts to distribute stress over the bolts more evenly

Failure rates (over the years)



Conclusions

- The current maintenance policy is close to cost-optimal
- Increasing joint reliability by e.g., more inspections does not pay off
- Additional maintenance costs outweigh the reduced cost of failures
- Combination of exponential and deterministic timings
  - Analysed here by statistical model checking (Uppaal SMC)
  - Semantics and numerical algorithms would be of interest

The Need for Parameter Synthesis

Fact:
Probabilistic model checking is applicable to various areas, e.g.:

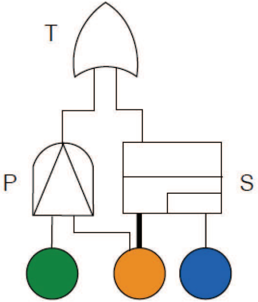
- reliability engineering
- randomised algorithms
- systems biology

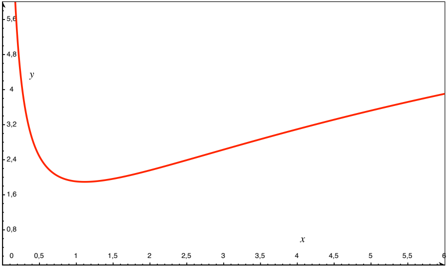
Markov models of hundreds of millions of states can be handled.

Limitation:
Probabilities need to be known a priori. Precisely.
How sensitive are results when transition probabilities fluctuate?

Goal:
Treat parametric models, synthesise "safe" parameter values

Parametric Fault Trees

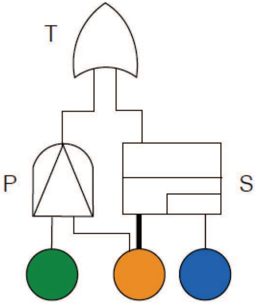


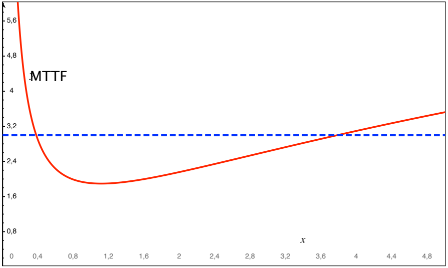


Sample parametric DFT and its MTTF

$$\mathrm{MTTF} = \frac{200x^2 + 20x + 201}{x \cdot (20x + 201)} \text{ for } (\alpha, \beta, \gamma, d) = (10, x, 0.1, 0.5)$$

Parametric Fault Trees

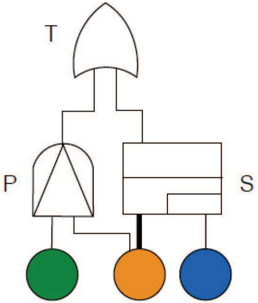


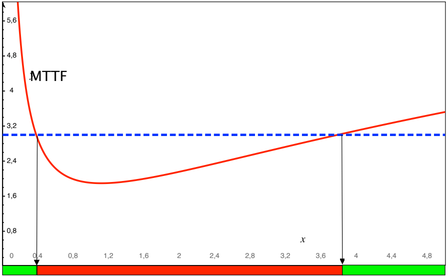


Sample parametric DFT and its MTTF

For which $1/10 \leqslant x \leqslant 10$ does MTTF $\geqslant$ 3 hold?

Parametric Fault Trees





Sample parametric DFT and its MTTF

For which $1/10 \leqslant x \leqslant 10$ does MTTF $\geqslant$ 3 hold?

Further readings

- M. Volk, S. Junges, J-P. Katoen. *Fast dynamic fault tree analysis by model-checking techniques.* IEEE Transactions on Industrial Informatics, 2018.
- M. Volk, S. Junges, J-P. Katoen and M. Stoelinga. *One net fits all: A unifying semantics of DFTs using GSPNs.* 2018 (submitted).
- S. Junges *et al.* *Fault trees on a diet: automated reduction by graph rewriting.* Formal Aspects of Computing, 2017.
- M. Ghadhab *et al.* *Model-based safety analysis for vehicle guidance systems.* SafeComp 2017.
- E. Ruijters and M. Stoelinga. *Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools.* Computer Science Review, 2015.
- S. Junges, D. Guck, J-P. Katoen and M. Stoelinga. *Uncovering DFTs.* DSN 2016.
- J-P. Katoen and M. Stoelinga. *Boosting fault tree analysis by formal methods.* Festschrift Ed Brinksma, 2017.

Tool support: `www.stormchecker.org`