PFL: A Probabilistic Logic for Fault Trees

Stefano M. Nicoletti[1](✉) , Milan Lopuhaä-Zwakenberg[1] ,
E. Moritz Hahn[1] , and Mariëlle Stoelinga[1,2]

[1] Formal Methods and Tools, University of Twente, Enschede, The Netherlands
{s.m.nicoletti,m.a.lopuhaa,e.m.hahn,m.i.a.stoelinga}@utwente.nl
[2] Department of Software Science, Radboud University, Nijmegen, The Netherlands

Abstract. Safety-critical infrastructures must operate in a safe and reliable way. Fault tree analysis is a widespread method used for risk assessment of these systems: fault trees (FTs) are required by, e.g., the Federal Aviation Administration and the Nuclear Regulatory Commission. In spite of their popularity, little work has been done on formulating structural queries about FTs and analyzing these, e.g., when evaluating potential scenarios, and to give practitioners instruments to formulate queries on FTs in an understandable yet powerful way. In this paper, we aim to fill this gap by extending *BFL* [37], a logic that reasons about Boolean FTs. To do so, we introduce a Probabilistic Fault tree Logic (PFL). PFL is a simple, yet expressive logic that supports easier formulation of complex scenarios and specification of FT properties that comprise probabilities. Alongside PFL, we present LangPFL, a domain specific language to further ease property specification. We showcase PFL and LangPFL by applying them to a COVID-19 related FT and to a FT for an oil/gas pipeline. Finally, we present theory and model checking algorithms based on binary decision diagrams (BDDs).

1 Introduction

Our self-driving cars, power plants, oil/gas refineries and transportation systems must operate in a safe and reliable way. Risk assessment is a key activity to identify, analyze and prioritize the risk in a system, and come up with (cost-)effective countermeasures. Fault tree analysis (FTA) [43,45] is a widespread formalism to support risk assessment. FTA is applied to many safety-critical systems and the use of fault trees is required, e.g., by the Federal Aviation Administration (FAA), the Nuclear Regulatory Commission (NRC), in the ISO 26262 standard [28] for autonomous driving and for software development in aerospace systems. A fault tree (FT) models how component failures arise and propagate through the system, eventually leading to system level failures. Leaves in a FT represent

basic events (BEs), i.e. elements of the tree that do not need further refinement. Once these fail, the failure is propagated through the *intermediate events* (IEs) via *gates*, to eventually reach the *top level event* (TLE), which symbolizes system failure. In the (sub)tree represented in Fig. 1, the TLE—*Medium Corrosion*—is refined by an AND-gate (*MeC*). For *MeC* to fail, water must be present, i.e., the *With Water* (*WW*) BE must fail, and there must be at least one acid medium in the pipes, i.e., *Acid Medium* (*AcM*) has to happen. This last OR-gate is further refined with three BEs: for it to fail, at least one of its three children needs to fail. This means that either *Hydrogen sulfide* (H_2S) or *Oxygen* (O_2) or *Carbon dioxide* (CO_2) must be present. Fault tree analysis supports qualitative and quantitative analysis. Qualitative analysis aims at pointing out root causes and critical paths in the system. One can identify the *minimal cut sets* (MCSs) of a FT, i.e. minimal sets of BEs that, when failed, cause the system to fail. One can also identify *minimal path sets* (MPSs), i.e. minimal sets of BEs that - when operational - guarantee that the system will remain operational. Quantitative analysis allows to compute relevant dependability metrics, such as the system reliability, availability and mean time to failure. A formal background on FTs is given in Sect. 2.
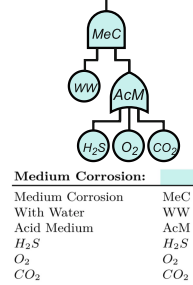

Medium Corrosion:

Medium Corrosion	MeC
With Water	WW
Acid Medium	AcM
H_2S	H_2S
O_2	O_2
CO_2	CO_2

Fig. 1. FT excerpt from Fig. 3.

Probabilistic Fault Tree Logic. In spite of their popularity, little work has been done on formulating structural queries about FTs and analyzing these, e.g., when evaluating potential scenarios, and to give practitioners instruments to formulate queries on FTs in an understandable yet powerful way. Usually, FTs are translated to stochastic models and existing logics specify properties on these, rather than on elements of FTs. Our previous work [37] presents a logic to reason about static FTs when BEs have Boolean values. The present work aims to extend that framework by devising a probabilistic logic for FTs, called PFL, where one could easily reason about FTs also taking probabilities into account. To further meet the need for usability - that we uncovered through interviews with a domain expert [36] - we present a domain specific language for PFL, LangPFL, and showcase property specification with both on two case studies, one with a COVID-19 FT, and one with an oil/gas pipeline FT.

Model Checking. In this paper, we provide model checking algorithms that extend our work in [37]. While we build from algorithms from [37], we require extensions for formulae in which probabilities come into play. We introduce novel algorithms which can decide 1. whether a single probability assignment to all BEs of a FT satisfies a formula; 2. whether a formula is satisfied for all possible probability assignments to BEs and 3. in which regions of the parameter space the considered formula holds. Building on our previous work, all three algorithms are based on construction and manipulation of binary decision diagrams (BDDs). This translation to BDDs constitutes a formal ground to address these procedures in a uniform way, while integrating novel work presented in this paper with previous algorithms.

Related Work. Numerous logics describe properties of state-transition systems, such as labelled transition systems (LTSs) and Markov models, e.g., CTL [14], LTL [40], and their variants for Markov models, PCTL [26] and PLTL [38]. State-transition systems are usually not written by hand, but are the result of the semantics of high-level description mechanisms, such as AADL [9], the hardware description language VHDL [19] or model description languages such as JANI [11] or PRISM [33]. Consequently, these logics are not used to reason about the structure of such models (e.g. the placement of circuit elements in a VHDL model or the structure of modules in a PRISM model), but on the temporal behaviour of the underlying state-transition system. Similarly, related work on model checking on FTs [6,8,46,47] exhibits significant differences: these works perform model checking by referring to states in the underlying stochastic models, and properties are formulated in terms of these stochastic logics, not in terms of events in the given FT. In [48], the author provides a formulation of *Pandora*, a logic for the qualitative analysis of temporal FTs. In spite of the use of logic to capture properties of FTs, [48] focuses on the analysis of time, introducing gates that are different from the ones considered in this work: the Priority-AND-gate (PAND), the Simultaneous-AND-gate (SAND), and the Priority-OR gate (POR). In PFL we do not (yet) consider time and we focus on AND, OR and VOT-gates. Furthermore, [48] focuses more on the algorithmic part of FTA while leaving out any formalization of FTs or the logic defined upon. In [25] the authors investigate how FTA results can be linked to software safety requirements by proposing the same system model for both. They introduce a duration calculus based on discrete time interval logic (ITL) [34] to give FTs formal semantics. Our work, on the other hand, adopts standard semantics for FTs and develops a logic to specify probabilistic properties on FTs. Furthermore, we do not address timed behaviours while [25] disregards probabilistic analysis on FTs. In previous work [37] we presented *BFL*, a logic on FTs that however reasons about FTs only in Boolean terms. We take this framework and develop a logic that extends *BFL* with probabilities. Literature related to FTs, property specification languages, BDDs and parametric model checking is referenced and contextualized in Sect. 2, Sect. 5, Sect. 6.1 and Sect. 6.5.

Contributions. To summarize, in this work:
1. We develop PFL, a probabilistic logic to reason about FTs.
2. We present a domain specific language for PFL, LangPFL, to further ease property specification.
3. We showcase the potential of PFL and LangPFL by applying them to a medium-sized COVID-19 related example and to a large-sized case study of an oil/gas pipeline.
4. We provide model checking algorithms to check properties defined in PFL.
5. We provide the theory and an algorithm to solve problems where the probabilities of BEs are parametric.

Structure of the Paper. Section 2 covers background on FTs, Sect. 3 describes PFL, Sect. 4 shows the application of PFL to case studies, Sect. 5 introduces LangPFL, Sect. 6 presents algorithms and Sect. 7 concludes our work.

2 Fault Trees: Background

Developed in the early '60s [21], FTs are directed acyclic graphs (DAGs) that model how low-level failures can propagate and cause a system-level failure. The overall failure of a system is captured by a *top level event* (TLE), that is refined through the use of *gates*. FTs come with different gate types. For the purposes of our paper and in order to create a modular and functional framework, we will focus on *static* fault trees, featuring OR-gates, AND-gates and VOT*(k/N)*-gates: we foresee support for dynamic gates as a possible future extension (see Sect. 7). For a low-level failure to propagate, at least one child of an OR-gate has to fail, all the children of an AND-gate must fail, and at least k out of N children must fail for a VOT(k/N)-gate to fail. When gates can no longer be refined, we reach the *basic events* (BEs) which are the leaves of the tree. FTs enable both qualitative and quantitative analyses. On the qualitative side, *minimal cut sets* (MCSs) and *minimal path sets* (MPSs) highlight root causes of failures and critical paths in the system. MCSs are minimal sets of events that - when failed - cause the failure of the TLE. MPSs are minimal sets of events that - when remaining operational - guarantee that the TLE will remain operational.

Definition 1 (*Fault Tree*). *A* Fault Tree *is a tuple* $T = (\mathtt{E}, A, t)$ *where* $(\mathtt{E}, A)$ *is a rooted directed acyclic graph (*$\mathtt{E}$ *are the vertices, called* events*) and t is a map* $\mathtt{E} \to \{\mathtt{AND}, \mathtt{OR}, \mathtt{BE}\}$ *such that* $t(e) = \mathtt{BE}$ *iff e is a leaf.*

We denote the top event by e_{top}, and the set of children of an event e by $ch(e) = \{e' \mid (e, e') \in A\}$. Slightly abusing notation, we denote the set of *basic events*, e with $t(e) = \mathtt{BE}$, as $\mathtt{BE}$, whose elements we enumerate $\mathtt{BE} = \{e_1, \ldots, e_n\}$. We also define the set of intermediate events $\mathtt{IE} = \mathtt{E} \backslash \mathtt{BE}$. The behaviour of a FT T can be rigorously expressed through its *structure function* [43] - Φ_T: if we assume the convention that a BE has value 1 if failed and 0 if operational, the structure function indicates the status of the TLE given the status of all the n BEs of T, given by a Boolean vector $\overline{b} = (b_1, \ldots, b_n)$. Such a boolean vector can also be regarded as a subset of BE, allowing us to interpret statements such as $\overline{b'} \subset \overline{b}$.

Definition 2 (*Structure Function*). *The structure function of an FT T is a function* $\Phi_T \colon \mathbb{B}^n \times \mathtt{E} \to \mathbb{B}$ *defined recursively by*

$$\Phi_T(\overline{b}, e) = \begin{cases} b_i & \text{if } e = e_i \in \mathtt{BE} \\ \bigvee_{e' \in ch(e)} \Phi_T(\overline{b}, e') & \text{if } t(e) = \mathtt{OR} \\ \bigwedge_{e' \in ch(e)} \Phi_T(\overline{b}, e') & \text{if } t(e) = \mathtt{AND} \end{cases}$$

Thus, for each set of BEs we can identify its characteristic vector $\overline{b}$. One can extend Definition 2 by allowing gates derived from AND- and OR-gates, e.g., voting gates, where a gate with $t(e) = \mathtt{VOT}(k/N)$ fails if at least k of its children fail, i.e.

$$\sum_{e' \in ch(e)} \Phi_T(\overline{b}, e') \geq k$$

We can also define the classical notions of minimal cut sets and minimal path sets [43]. A cut set is any set of basic events that causes the TLE to occur, i.e., for which the structure function evaluates to 1. A path set is any set of basic events that does not cause the TLE to occur, i.e., for which the structure function evaluates to 0.

Definition 3. *A status vector $\overline{b}$ is a cut set (CS) for $e \in$ E of a given tree T iff $\Phi_T(\overline{b}, e) = 1$. A* minimal cut set (MCS) *is a cut set of which no subset is a cut set: $\overline{b}$ is a MCS for $e \in$ E of T if $\Phi_T(\overline{b}, e) = 1 \wedge \forall \overline{b'} \subset \overline{b}, \Phi_T(\overline{b}, e) = 0$.*

Definition 4. *A status vector $\overline{b}$ is a* path set (PS) *for $e \in$ E of a given tree T iff $\Phi_T(\overline{b}, e) = 0$. A* minimal path set (MPS) *is a path set of which no subset is a path set: $\overline{b}$ is a MPS for $e \in$ E of T if $\Phi_T(\overline{b}, e) = 0 \wedge \forall \overline{b'} \subset \overline{b}, \Phi_T(\overline{b}, e) = 1$.*

3   A Probabilistic Logic to Reason About FTs

3.1   Syntax

Our logic PFL consists of three syntactical layers represented by ϕ, ψ and ξ respectively. To refer to layer-two or layer-three formulae indistinctly we write θ and χ is a generic formula in PFL. Layer-one is Boolean and we indicate atomic formulae with the letter e. Each atomic formula represent an element of a given FT, it being an IE or a BE. Furthermore, in layer-one we have the possibility to arbitrarily set the value of one atom e in complex formulae either to 0 or to 1 by writing $\phi[e \mapsto 0]$ and $\phi[e \mapsto 1]$. Note that $\phi[e \mapsto 0]$ is not equivalent to $\phi \wedge \neg e$: for $\phi = \neg e$, we have $(\neg e)[e \mapsto 0] = true$ while $(\neg e) \wedge \neg e$ does not necessarily equal *true*. Moreover, we have operators to check for MPSs and MCSs for a given layer-one formula. The second layer allows us to reason about probabilities and their bounds. We can check whether the probability of a given layer-one formula (potentially conditioned by another one) respects a certain threshold. We can set the value of one atom e in complex formulae to an arbitrary probability value p. We can also check if two layer-one formulae (e.g., two intermediate events) are stochastically independent. Formulae in ϕ and ψ can be rewritten with the usual negation and conjunction. Finally, the third layer allows us to return the probability value for a given layer-one formula, possibly mapping atoms to an arbitrary probability value p. Note that, for all three layers, we usually assign values to $e \in$ BE. We can however assign values to IEs if 1. e is a module [20], i.e., all paths between descendants of e and the rest of the FT pass through e 2. and none of the descendants of e are present in the formula. If so, we prune that (sub)FT and treat occurring IEs as BEs.

$$\phi ::= e \mid \neg\phi \mid \phi \wedge \phi \mid \phi[e \mapsto 0] \mid \phi[e \mapsto 1] \mid \mathrm{MCS}(\phi)$$
$$\psi ::= \neg\psi \mid \psi \wedge \psi \mid \Pr_{\bowtie p}(\phi \mid \phi) \mid \psi[e \mapsto q] \mid \mathrm{IDP}(\phi, \phi)$$
$$\xi ::= \Pr(\phi \mid \phi) \mid \xi[e \mapsto q]$$

where $\bowtie \in \{<, \leq, =, \geq, >\}$.

Syntactic sugar. We let X_n be the set of layer-n formulae and we define the following derived operators, where formulae θ are in the set of layer-one or layer-two formulae, i.e., such that $\theta \in X_1 \cup X_2$:

$$\theta_1 \vee \theta_2 ::= \neg(\neg\theta_1 \wedge \neg\theta_2) \qquad\qquad \theta_1 \not\Leftrightarrow \theta_2 ::= \neg(\theta_1 \Leftrightarrow \theta_2)$$

$$\theta_1 \Rightarrow \theta_2 ::= \neg(\theta_1 \wedge \neg\theta_2) \qquad\qquad \mathrm{MPS}(\phi) ::= \mathrm{MCS}(\neg\phi)$$

$$\theta_1 \Leftrightarrow \theta_2 ::= (\theta_1 \Rightarrow \theta_2) \wedge (\theta_2 \Rightarrow \theta_1) \qquad \mathrm{SUP}(e) ::= \mathrm{IDP}(e, e_{top})$$

$$\mathop{\mathrm{Vot}}_{\bowtie k}(\phi_1, \ldots, \phi_N) ::= \bigvee_{\substack{U \subseteq \{1,\ldots,N\} \\ |U| \bowtie k}} \left(\bigwedge_{u \in U} \phi_u \right) \wedge \left(\bigwedge_{u \in \{1,\ldots,N\}\setminus U} \neg\phi_u \right) \quad \text{with } k \leq N$$

where MPS checks for minimal path sets of a given formula and SUP checks if an element e is superfluous, i.e., if it is independent w.r.t. the TLE.

3.2   Semantics

The semantics for our logic is structured according to the three syntactic layers. For the first layer of PFL, formulae are evaluated on a Boolean status vector $\overline{b}$ and on a tree T. Atomic formulae e are satisfied by $\overline{b}$ and T if the structure function in Definition 2 returns 1 with these $\overline{b}$ and e as input. Formally:

$$\overline{b}, T \models e \qquad\quad \text{iff } \Phi_T(\overline{b}, e) = 1$$

$$\overline{b}, T \models \neg\phi \qquad\quad \text{iff } \overline{b}, T \not\models \phi$$

$$\overline{b}, T \models \phi \wedge \phi' \quad\;\; \text{iff } \overline{b}, T \models \phi \text{ and } \overline{b}, T \models \phi'$$

$$\overline{b}, T \models \phi[e_i \mapsto 0] \text{ iff } \overline{b'}, T \models \phi \text{ with } \overline{b'} = (b'_1, \ldots, b'_n) \text{ where}$$
$$b'_i = 0 \text{ and for } j \neq i \text{ we have } b'_j = b_j$$

$$\overline{b}, T \models \phi[e_i \mapsto 1] \text{ iff } \overline{b'}, T \models \phi \text{ with } \overline{b'} = (b'_1, \ldots, b'_n) \text{ where}$$
$$b'_i = 1 \text{ and } b'_j = b_j \text{ for } j \neq i$$

$$\overline{b}, T \models \mathrm{MCS}(\phi) \;\; \text{iff } \overline{b}, T \models \phi \wedge (\neg\exists\overline{b'}.\; \overline{b'} \subset \overline{b} \wedge \overline{b'}, T \models \phi)$$

With $[\![\phi]\!]_T$ we denote the *satisfaction set* of vectors for ϕ, i.e., the set of all $\overline{b}$ that satisfy ϕ given T. Semantics for the second and third layer require the introduction of probabilities. If we consider the function $\Phi_T : \mathbb{B}^n \times \mathbf{E} \to \mathbb{B}$, we can devise an extension such that $\Phi_T : \mathbb{B}^n \times X_1 \to \mathbb{B}$, where X_1 is the set of layer-one formulae (note that $\mathbf{E} \subseteq X_1$). With a slight abuse of notation, Φ_T will now return 1 whenever the input Boolean vector satisfies the input layer-one formula. With $\phi \in X_1$, we lift the structure function to $\Phi_T^* : Dist(\mathbb{B}^n) \times X_1 \to [0, 1]$, where $Dist$ expresses a set of probability distributions, in a standard fashion, i.e.,

$$\Phi_T^*(\mu, \phi) = \sum \{\mu(\overline{b}) \mid \overline{b} \in \mathbb{B}^n \text{ for which } \Phi_T(\overline{b}, \phi) = 1\}$$

We further convert each probabilistic status vector $\overline{\rho} \in [0, 1]^n$ to a distribution $\mu_{\overline{\rho}} \in Dist(\mathbb{B}^n)$:

$$\mu_{\overline{\rho}}(b_1, \ldots, b_k) = \prod_{i=1}^{k} (b_i \times \rho_i + (1 - b_i) \times (1 - \rho_i))$$

We can then define semantics for the second syntactic layer as follows[1]:

$$\overline{\rho}, T \models \neg\psi \qquad \text{iff } \overline{\rho}, T \not\models \psi$$
$$\overline{\rho}, T \models \psi \wedge \psi' \qquad \text{iff } \overline{\rho}, T \models \psi \text{ and } \overline{\rho}, T \models \psi'$$
$$\overline{\rho}, T \models \Pr_{\bowtie p}(\phi \mid \phi') \text{ iff } \Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi')/\Phi_T^*(\mu_{\overline{\rho}}, \phi') \bowtie p$$
$$\overline{\rho}, T \models \psi[e_i \mapsto q] \text{ iff } \overline{\rho}[\rho_i \mapsto q], T \models \psi$$
$$\overline{\rho}, T \models \text{IDP}(\phi, \phi') \text{ iff } \Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi') = \Phi_T^*(\mu_{\overline{\rho}}, \phi) \cdot \Phi_T^*(\mu_{\overline{\rho}}, \phi')$$

Finally, to define semantics for the third layer we let $\mathsf{Val}_{\overline{\rho}, T}: X_3 \to [0,1]$ define an evaluation function of layer-three formulae in X_3:

$$\mathsf{Val}_{\overline{\rho}, T}(\Pr(\phi \mid \phi')) = \Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi')/\Phi_T^*(\mu_{\overline{\rho}}, \phi')$$
$$\mathsf{Val}_{\overline{\rho}, T}(\xi[e_i \mapsto q]) = \mathsf{Val}_{\overline{\rho}[\rho_i \mapsto q], T}(\xi)$$

Furthermore we write $T \models \theta$, meaning $\forall \overline{\rho} . \overline{\rho}, T \models \theta$.

4   Case Study: Examples

We showcase the potential of our logic by presenting two case studies: a COVID-19 related FT [3,37] and the FT for an oil/gas pipeline [50].

4.1   COVID-19 FT

The TLE represents a COVID-19 infected worker on site, abbreviated *IWoS*. As shown in Fig. 2, the FT considers events in several categories: COVID-19 pathogens and reservoirs (i.e., germs and objects carrying the virus); their mode of transmissions; the presence of susceptible hosts, infected objects and workers; physical contacts as well as human errors. Note that Fig. 2 contains several repeated basic events (marked with a dashed border): *IT*, *PP*, *H1* and *IW*. This
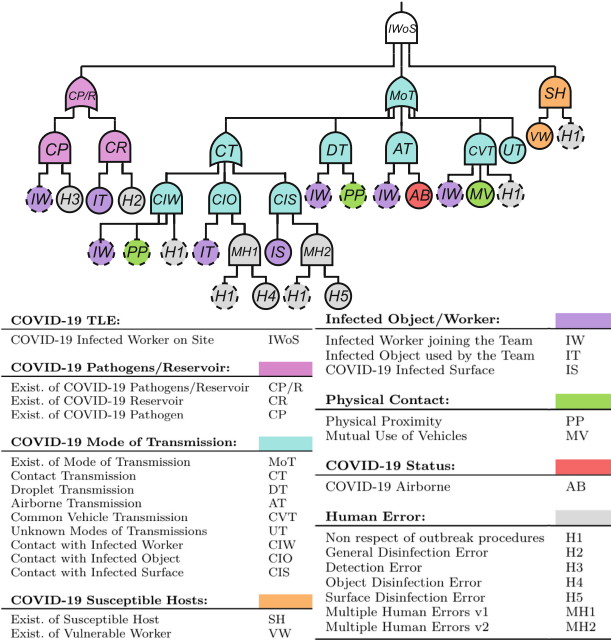

COVID-19 TLE:

COVID-19 Infected Worker on Site	IWoS

COVID-19 Pathogens/Reservoir:

Exist. of COVID-19 Pathogens/Reservoir	CP/R
Exist. of COVID-19 Reservoir	CR
Exist. of COVID-19 Pathogen	CP

COVID-19 Mode of Transmission:

Exist. of Mode of Transmission	MoT
Contact Transmission	CT
Droplet Transmission	DT
Airborne Transmission	AT
Common Vehicle Transmission	CVT
Unknown Modes of Transmissions	UT
Contact with Infected Worker	CIW
Contact with Infected Object	CIO
Contact with Infected Surface	CIS

COVID-19 Susceptible Hosts:

Exist. of Susceptible Host	SH
Exist. of Vulnerable Worker	VW

Infected Object/Worker:

Infected Worker joining the Team	IW
Infected Object used by the Team	IT
COVID-19 Infected Surface	IS

Physical Contact:

Physical Proximity	PP
Mutual Use of Vehicles	MV

COVID-19 Status:

COVID-19 Airborne	AB

Human Error:

Non respect of outbreak procedures	H1
General Disinfection Error	H2
Detection Error	H3
Object Disinfection Error	H4
Surface Disinfection Error	H5
Multiple Human Errors v1	MH1
Multiple Human Errors v2	MH2

Fig. 2. COVID-19 FT.

---

[1] When considering conditional probabilities in layer-two and layer-three formulae, we disregard the case in which $\Phi_T^*(\mu_{\overline{\rho}}, \phi') = 0$.

TLE *IWoS* is refined via an AND-gate with three children. Thus, for the TLE to occur the following must happen: COVID pathogens/COVID infected objects must exist, there has to be a susceptible host and COVID pathogens must be transmitted in some way to this host. These events are captured by corresponding subtrees: the purple OR-gate *CP/R* refines the existence of COVID pathogens/- COVID infected objects, the OR-gate *MoT* in teal refines modes of transmission and the AND-gate *SH* in orange details the presence of a susceptible host.

Properties. Following, we specify some properties using natural language and present the corresponding PFL formulae:

1) What are all the MCSs for the modes of transmission that include errors in objects and surfaces disinfection? $[\![\mathrm{MCS}(MoT) \wedge H4 \wedge H5]\!]_T$;
2) Is the probability of TLE smaller than 0.03, if physical proximity occurred? $\mathrm{Pr}_{\leq 0.03}(IWoS)[PP \mapsto 1]$;
3) Assume that the probability of an infected worker on the team equals 0.25. How does that affect the probability of TLE? $\mathrm{Pr}(IWoS)[IW \mapsto 0.25]$;
4) Assume that both COVID-19 pathogens and a vulnerable worker exist. Does this imply that $P(IWoS) \geq 0.15$? $\mathrm{Pr}_{=1}(CP) \wedge \mathrm{Pr}_{=1}(VW) \Rightarrow \mathrm{Pr}_{\geq 0.15}(IWoS)$.
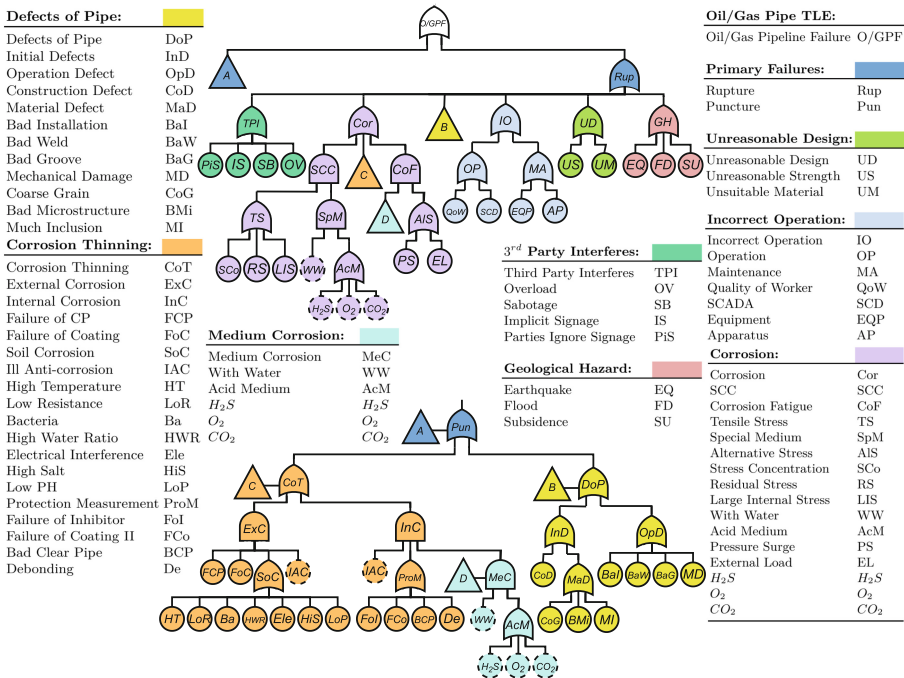
4.2   Oil/Gas Pipeline FT

Fig. 3. Oil/gas pipeline FT.

The TLE represents the failure of an oil/gas pipeline, abbreviated O/GPF. As shown in Fig. 3, the FT considers events in several categories: failures like ruptures and punctures; third party interference; different kinds of corrosion; incorrect performance of some operations (e.g., maintenance); unreasonable design choices; as well as defects on pipes. Figure 3 contains several repeated basic events (again, marked with a dashed border): WW, H_2S, O_2, CO_2 and IAC. Furthermore, multiple sub-trees are referenced/repeated in different places: those are marked using labelled triangles. The TLE O/GPF is refined via an OR-gate with two children, in **blue**. Thus, for the TLE to occur either a rupture or a puncture must happen. These two events are captured by corresponding subtrees. The **rupture** subtree (top-right of Fig. 3) is refined by an OR-gate with six children: the **green** OR-gate TPI refines possible interference by third parties; the **violet** OR-gate Cor refines modes of pipes corrosion; the **yellow** subtree B refines modes in which pipes could be defective; the **dove gray** OR-gate IO details possible incorrect operations; the **lime green** OR-gate UD details unreasonable design choices; and the **pink** OR-gate GH refines possible geological hazards. Similarly, the **puncture** subtree (bottom of Fig. 3) is refined by an OR-gate with two children: the **orange** OR-gate CoT refines modes in which corrosion can make pipes thinner—with a detailed subtree in **light blue** refining medium corrosion; and the OR-gate DoP in **yellow** that refines modes in which pipes could be defective.

Properties. We specify some properties using natural language and present the corresponding PFL formulae:

1) What are all the MPSs for pipes rupture that include the absence of water as a corrosive medium, H_2S, O_2 and CO_2? $[\![\mathrm{MPS}(Rup) \wedge \neg WW \wedge \neg H_2S \wedge \neg O_2 \wedge \neg CO_2]\!]_T$;
2) Assume that H_2S shows up in the pipes with 0.25% probability. What is the probability of pipes corrosion, if corrosion happens with water with 2% probability and that pressure surges with 1% probability? $\Pr(Cor)[H_2S \mapsto 0.0025, WW \mapsto 0.02, PS \mapsto 0.01]$;
3) Assume that the probability of pipes corrosion with acid is equal to 0.005. Assume also that pipes present defects in their construction material with 0.2% probability. Is the probability of TLE happening lower than 1.2%? $\Pr_{\leq 0.012}(O/GPF)[AcM \mapsto 0.005, MaD \mapsto 0.02]$.

5   LangPFL: A Domain Specific Language for PFL

Design of LangPFL. To ease usability of PFL, we present LangPFL, a Domain Specific Language (DSL) to specify properties in PFL. The need for a simple way to specify properties involving probability on FT was uncovered via interviews with a domain expert from industry [36]. Defining languages and tools for properties and requirements specification is common practice: in [17] the authors capture high-level requirements for a steam boiler system in a human readable

Table 1. Properties in natural language, PFL and LangPFL.

Natural Language	Property in PFL	LangPFL
What are all the MCSs for the modes of transmission that include errors in objects and surfaces disinfection?	$[\![MCS(MoT) \wedge H4 \wedge H5]\!]_T$	**assume:** **computeall:** MCS[MoT] and H4 and H5
Is the probability of TLE smaller than 0.03, if physical proximity occurred?	$\Pr_{\leq 0.03}(IWoS)[PP \mapsto 1]$	**assume:** setp PP = 1 **check:** P[IWoS] ≤ 0.03
Assume that the probability of an infected worker on the team equals 0.25. How does that affect the probability of TLE?	$\Pr(IWoS)[IW \mapsto 0.25]$	**assume:** setp IW = 0.25 **compute:** P[IWoS]
Assume that both COVID-19 pathogens and a vulnerable worker exist. Does this imply that $P(IWoS) \geq 0.15$?	$\Pr(CP) \wedge \Pr(VW)_{=1} {=1} \Rightarrow \Pr_{\geq 0.15}(IWoS)$	**assume:** setp CP = 1 setp VW = 1 **check:** P[IWoS] ≥ 0.15
What are all the MPSs for pipes rupture that include the absence of water as a corrosive medium, H_2S, O_2 and CO_2?	$[\![MPS(Rup) \wedge \neg WW \wedge \neg H_2S \wedge \neg O_2 \wedge \neg CO_2]\!]_T$	**assume:** **computeall:** MPS [Rup] and not WW and not H_2S and not O_2 and not CO_2
Assume that H_2S shows up in the pipes with 0.25% probability. What is the probability of pipes corrosion, if corrosion happens with water with 2% probability and that pressure surges with 1% probability?	$\Pr(Cor)[H_2S \mapsto 0.0025, WW \mapsto 0.02, PS \mapsto 0.01]$	**assume:** setp H_2S = 0.0025 setp WW = 0.02 setp PS = 0.01 **compute:** P[Cor]
Assume that the probability of pipes corrosion with acid is equal to 0.005. Assume also that pipes present defects in their construction material with 0.2% probability. Is the probability of TLE lower than 1.2%?	$\Pr_{\leq 0.012}(O/GPF)[AcM \mapsto 0.005, MaD \mapsto 0.02]$	**assume:** setp AcM = 0.005 setp MaD = 0.02 **check:** P[O/GPF]≤0.012

form by presenting SADL, a controlled English requirements capturing language, alongside its tool suite ASSERT. Other controlled natural languages for knowledge representation include Processable English (PENG) [49], Controlled English to Logic Translation (CELT) [39] and Computer Processable Language (CPL) [13]. LangPFL is inspired by these languages for their ease of use and close proximity to natural language. Finally, another notable example is FRETish [15], a structured natural language capturing Linear Temporal Logic (LTL). FRETish was developed at NASA and is supported by the FRET tool [23]. Other than for its usability, FRETish inspired us with the clear way in which the scope, conditions and component of specified properties are clearly separated from desired behaviours on timing and responses. LangPFL expresses only a fragment of PFL:

most notably, nesting of formulae is disallowed. By doing so, we retain most of the expressiveness of PFL while making property specification easier. In LangPFL, FT elements are referred to with their label and each operator in PFL has a counterpart in the DSL: Boolean operators, not, and, or, impl...; setting the value of FT elements to Boolean or probability values, set, setp; MCSs and MPSs, MCS[...], MPS[...]; operators to check (conditional) probability thresholds/compute (conditional) probability values, P[... | ...] ⋈ ..., P[... | ...]; and to check for independence between FT elements IDP[..., ...].

LangPFL **Templates.** Properties can be specified in LangPFL by utilizing operators inside structured templates. Assumptions on the status of FT elements can be specified under the **assume** keyword. These assumptions will be automatically integrated in the translated formula accordingly, e.g., set or setp will be translated with the according operators to set evidence, while other assumptions will be the antecedent of an implication. A second keyword separates specified formulae from the assumptions and dictates the desired result: **compute** and **computeall** compute and return desired values, i.e., probability values and lists of MCSs/MPSs respectively, while **check** establishes if a specified property holds.

Case studies. In Table 1 we showcase the properties specified in Sect. 4 and their respective translation in LangPFL.

6 Model Checking Algorithms

Layer-One Formulae. With PFL extending previous work [37], algorithms to compute satisfiability of layer-one formulae remain unchanged. In particular, it is possible to model check PFL over a FT and a Boolean vector $\overline{b}$ when considering layer-one formulae. Furthermore, we can collect all Boolean vectors $\overline{b}$ such that $\overline{b}, T \models \phi$. As noted in [37], checking if $\overline{b}, T \models \phi$ holds is trivial if ϕ is a layer-one formula that does not contain an MCS or MPS operator. In that case, we can simply substitute the values of $\overline{b}$ in ϕ and see if the Boolean expression evaluates to true. This also works to check whether $\overline{\rho}, T \models \theta$ holds for a probabilistic vector $\overline{\rho}$, a tree-shaped FT and a layer-two/three formula θ without operators for MCS or MPS. In this case, values can be computed following usual probability laws. For the other cases, the computation becomes more complex, and procedures involving binary decision diagrams (BDDs) are necessary. Algorithms for the Boolean scenarios are described in Appendix A.3 and Appendix A.4 respectively.

Layer-Two/Three Formulae. When reasoning about satisfiability of second layer formulae, algorithms present differences. As such, we present three novel algorithms for PFL: 1. Given a vector $\overline{\rho}$, a FT T and a formula ψ, check if $\overline{\rho}, T \models \psi$ (Sect. 6.4), 2. Given T and ψ, compute regions of the parameter space where $T \models \psi$ (Sect. 6.5), 3. Given a FT T and a formula ψ, check whether $T \models \psi$ for all $\overline{\rho}$ (Sect. 6.6). In continuity with previous work, all three algorithms are based on BDDs: first, FT elements that appear in a given layer-one formula are identified. Then, BDDs for these elements are selectively constructed (see

Algorithm 5) and stored to reduce computation time. Finally, these BDDs are manipulated and equipped with probabilities (see Algorithm 1) to reflect the semantics of the operators in PFL. Probability values in layer-three formulae are computed with slight variations on layer-two algorithms (see Sect. 6.4). As in standard FTA [43], we assume that BEs fail independently. A brief overview of each algorithm is given in Sect. 6.4, Sect. 6.5 and Sect. 6.6 respectively.

6.1   (Reduced Ordered) Binary Decision Diagrams

BDDs are directed acyclic graphs (DAGs) that compactly represent Boolean functions [2] by reducing redundancy. Depending on variable's ordering, BDD's size can grow linearly in the number of variables and at worst exponentially. In practice, BDDs are heavily used, including in FT analysis [4,42] and in their security-related counterpart, attack trees (ATs) [12]. Formally, a BDD is a rooted DAG $\mathbf{B}_f$ that represents a Boolean function $f \colon \mathbb{B}^n \to \mathbb{B}$ over variables $Vars = \{x_i\}_{i=1}^n$. Each nonleaf w has two outgoing arrows, labeled 0 and 1, and a label $Lab(w) \in Vars$; furthermore, each leaf has a label 0 or 1. Given a b in $\mathbb{B}^n$, the BDD is used to compute $f(b)$ as follows: starting from the top, upon arriving at a node w with $Lab(w) = x_i$, one takes the 0-edge if $b_i = 0$ and the 1-edge if $b_i = 1$. The label of the leaf one ends up in, is then equal to $f(b)$. A function f can be represented by multiple BDDs, but has a unique *reduced ordered* representative, or ROBDD [5,10], where the x_i occur in ascending order, and the BDD is reduced as much as possible by removing irrelevant nodes and merging duplicates. This is formally defined below; we let $Low(w)$ (resp. $High(w)$) be the endpoint of w's 0-edge (resp. 1-edge) and let $R_{\mathbf{B}}$ be the BDD root.

Definition 5 (*Reduced   Ordered   Binary   Decision   Diagram ((RO)BDD)*). *Let Vars be a set. A* BDD *over Vars is a tuple* $\mathbf{B} = (W, A, Lab, u)$ *where* (W, A) *is a rooted directed acyclic graph, and* $Lab \colon W \to Vars \sqcup \{0,1\}, u \colon A \to \{0,1\}$ *are maps such that: 1. Every nonleaf w has exactly two outgoing edges a, a' with $u(a) \neq u(a')$, and $Lab(w) \in Vars$; 2. Every leaf w has $Lab(w) \in \{0,1\}$. 3. Vars are equipped with a total order,* $\mathbf{B}_f$ *is thus defined over a pair* $\langle Vars, < \rangle$*; 4. the variable of a node is of lower order than its children, that is:* $\forall w \in W_n . Lab(w) < Lab(Low(w)), Lab(High(w))$*; 5. the children of nonleaf nodes are distinct nodes; 6. nodes are uniquely determined by their label, low child and high child.*

6.2   Translating FTs/Formulae to BDDs

Translations. We shortly sketch the idea of translating a layer-one formula and a (sub)tree to BDDs. As mentioned, to translate formulae to BDDs, FT elements that appear in a given formula are identified. Then, BDDs representing these elements are selectively constructed and stored to reduce computation time. Finally, operations on these BDDs are performed to reflect semantics of the operators in PFL.

Translating FTs to BDDs. As a first step, a translation from FTs to BDDs is needed [37]. These BDDs represent exactly the structure function of (sub)trees. In the following, we assume $Vars = V \,\dot\cup\, V'$, where the set of variables $V = BE$ and the set of primed variables $V' = \{e'|e \in BE\}$ (used for the BDD translation of the MCS operator, see Appendix A.2). Furthermore, we keep $Var_{\mathbf{B}} : BDD \to Vars$ to be a function that returns variables occurring in a BDD [37]. Then, our translation function $\Psi_{FT} : E \to BDD$ takes elements of a FT as input and maps them to BDDs. For an exact definition of Ψ_{FT} see Appendix A.1.

Translating Formulae. With BDDs for FTs, the next step consists in manipulating them to mirror PFL operators in layer-one. I.e., given Ψ_{FT} and a FT T, for every PFL formula ϕ in the set of PFL layer-one formulae X_1 there exists a translation to BDDs $\underline{\mathbf{B}}_T : X_1 \to BDD$ in Algorithm 5 (see Appendix A.2). The implementation of this procedure abides the dynamic programming standards: by caching, we would reuse the translation of (sub)trees and (sub)formulae between different analyses without recomputing them each time anew.

6.3 Equipping BDDs with Probabilities

Once we obtain BDDs for FTs/ϕ-formulae, we can construct a function $\Phi_T^*(\mu_{\overline{x}}, \phi)$ from $[0,1]^n$ to $[0,1]$ that computes the probability value of ϕ given probability values in $\overline{x}$, where $\overline{x}$ can be substituted with any $\overline{\rho}$. Algorithm 1 shows this procedure: first, we compute $\underline{\mathbf{B}}_T(\phi)$ via Algorithm 5, we then obtain a polynomial $poly(\underline{\mathbf{B}}_T(\phi))$ representing $\underline{\mathbf{B}}_T(\phi)$

Algorithm 1. Obtain $\Phi_T^*(\mu_{\overline{x}}, \phi)$ for $\underline{\mathbf{B}}_T(\phi)$.

Input: FT T, formula ϕ
Output: function $\Phi_T^*(\mu_{\overline{x}}, \phi) : [0,1]^n \to [0,1]$
where $x_1, \dots, x_n \in \overline{x}$ are function parameters
Method:
$\underline{\mathbf{B}}_T(\phi) \leftarrow$ Algorithm 5(T, ϕ)
$poly(\underline{\mathbf{B}}_T(\phi)) \leftarrow \mathsf{value}(R_{\underline{\mathbf{B}}_T(\phi)})$, where:
- $\mathsf{value}(w_i \notin W_t) = (1 - x_i) \cdot \mathsf{value}(Low(w_i))$
  $+ x_i \cdot \mathsf{value}(High(w_i))$
- $\mathsf{value}(\top) = 1$ and $\mathsf{value}(\bot) = 0$
return $poly(\underline{\mathbf{B}}_T(\phi))$

---

via $\mathsf{value}(R_{\underline{\mathbf{B}}_T(\phi)})$, where $\mathsf{value}(w_i \notin W_t) = (1 - x_i) \cdot \mathsf{value}(Low(w_i)) + x_i \cdot \mathsf{value}(High(w_i))$, $\mathsf{value}(\top) = 1$ and $\mathsf{value}(\bot) = 0$. $x_1, \dots, x_n \in \overline{x}$ are parameters of the constructed function and can be substituted in $poly(\underline{\mathbf{B}}_T(\phi))$ with values from an arbitrary $\overline{\rho}$ to compute the overall probability value of the BDD for FT/ϕ-formula.

6.4 Algorithm 2: Model Checking PFL over a FT and a $\overline{\rho}$

Overview. Given a specific vector $\overline{\rho}$, a FT T and a PFL layer-two formula ψ, we want to check if $\overline{\rho}, T \models \psi$. To do so, if we come across a layer-one formula ϕ we translate it to a BDD, we equip the resulting BDD with probabilities obtained from $\overline{\rho}$ and we compute whether the resulting value respects the threshold set in the given layer-two formula ψ. Boolean connectives are resolved as usual and independence is checked according to probability laws once the value for the respective BDD is computed. For the corresponding layer-three formulae ξ, we

would simply return the value computed from the BDD instead of comparing it to the given layer-two threshold.

Algorithm 2**.** This algorithm shows a procedure to check if $\overline{\rho}, T \models \psi$, given $\overline{\rho}, T$ and ψ. Boolean connectives are handled as usual via case distinction. In the same way, probability values in $\overline{\rho}$ are replaced by mappings in ψ, if any. For $\mathrm{Pr}_{\bowtie p}(\phi \mid \phi')$, we compute the BDD $\underline{\mathbf{B}}_T(\phi_n)$ for each ϕ_n of the respective layer-one formulae via Algorithm 1. Finally, we compute the conditional probability $P(\phi \mid \phi')$. If the returned value respects the threshold set in ψ we return *True*, *False* otherwise. For IDP we follow an analogous procedure: we compute probability values of needed layer-one inner formulae and we return *True* if they are stochastically independent. An algorithm for layer-three formulae ξ would simply return the conditional probability value for $\mathrm{Pr}(\phi \mid \phi')$, after potentially modifying $\overline{\rho}$ and computing $P(\underline{\mathbf{B}}_T(\phi_n))$.

Example. Let us consider the subtree in Fig. 1 and a vector with probability values for WW, H_2S, O_2 and CO_2 respectively: $\overline{\rho} = (0.002, 0.001, 0.0015, 0.002)$. Suppose we want to know if $P(MeC)$ is lower or equal to 0.0001, assuming the scenario where $P(H_2S) = 0.0023$ and $P(WW) = 0.015$, i.e., formally with $\psi = \mathrm{Pr}_{\leq 0.0001}(MeC)[H_2S \mapsto 0.0023, WW \mapsto 0.015]$. First, $\overline{\rho}$ would be modified as per the new assignments in ψ: $\overline{\rho} = (0.015, 0.0023, 0.0015, 0.002)$. Then, Algorithm 2 is called again with the modified $\overline{\rho}$ and the BDD $\underline{\mathbf{B}}_T(MeC)$ for MeC is constructed (see Fig. 4). The value for the BDD is computed via Algorithm 1. The result (0.000087) is lower than the threshold in ψ, the formula is satisfied and the algorithm returns *True*.

Algorithm 2. Check if $\overline{\rho}, T \models \psi$, given $\overline{\rho}, T$ and ψ.

Input: prob. vector $\overline{\rho}$, FT T, formula ψ
Output: *True* iff $\overline{\rho}, T \models \psi$, *False* otherwise.
Method:
if $\psi = \neg\psi'$ **then return not**(Algorithm 2($\overline{\rho}, T, \psi'$))
else if $\psi = \psi' \wedge \psi''$ **then return** Algorithm 2($\overline{\rho}, T, \psi'$) **and** Algorithm 2($\overline{\rho}, T, \psi''$)
else if $\psi = \mathrm{Pr}_{\bowtie p}(\phi \mid \phi')$ **then**
$\quad$ $P(\underline{\mathbf{B}}_T(\phi)), P(\underline{\mathbf{B}}_T(\phi')) \leftarrow$ Algorithm 1(T, ϕ)($\overline{\rho}$), Algorithm 1(T, ϕ')($\overline{\rho}$)
$\quad$ $P(\phi \mid \phi') = \frac{P(\underline{\mathbf{B}}_T(\phi)) \cdot P(\underline{\mathbf{B}}_T(\phi'))}{P(\underline{\mathbf{B}}_T(\phi'))}$
$\quad$ **return** $P(\phi \mid \phi') \bowtie p$
else if $\psi = \psi'[e_i \mapsto q]$ **then return** Algorithm 2($\overline{\rho}[\rho_i \mapsto q], T, \psi'$)
else if $\psi = \mathrm{IDP}(\phi, \phi')$ **then**
$\quad$ $P(\underline{\mathbf{B}}_T(\phi)), P(\underline{\mathbf{B}}_T(\phi')), P(\underline{\mathbf{B}}_T(\phi \wedge \phi')) \leftarrow$ Algorithm 1(T, ϕ)($\overline{\rho}$), Algorithm 1(T, ϕ')($\overline{\rho}$),
$\quad$ Algorithm 1($T, \phi \wedge \phi'$)($\overline{\rho}$)
$\quad$ **return** $P(\underline{\mathbf{B}}_T(\phi)) \cdot P(\underline{\mathbf{B}}_T(\phi')) = P(\underline{\mathbf{B}}_T(\phi \wedge \phi'))$
end if

6.5 Algorithm 3: Computing regions where ψ-formulae are satisfied

Overview. Given a FT T and a layer-two formula ψ, we want to find the region S_{yes} in $[0,1]^n$ of all $\overline{\rho}$ that satisfy ψ. Typically, such a region is defined by large polynomials, and therefore difficult to describe analytically. Instead, we provide an algorithm that approximates this region up to a given level of precision. Such an approximation is given in the definition below: it consists of a region S_{yes} where ψ is known to hold, a region S_{no} where ψ does not hold, and the remainder S_{maybe} is of limited volume.
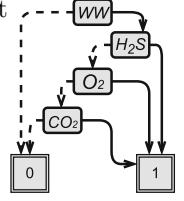
Fig. 4. BDD for Fig. 1.

Definition 6. *Let T be a FT, let $\varepsilon \in (0,1]$, and let ψ be a layer-two formula. A ε-partition for ψ is a partition $(S_{yes}, S_{no}, S_{maybe})$ of $[0,1]^n$ such that: 1. $\overline{\rho}, T \models \psi$ for all $\overline{\rho} \in S_{yes}$; 2. $\overline{\rho}, T \not\models \psi$ for all $\overline{\rho} \in S_{no}$; 3. $\text{Vol}(S_{maybe}) \leq \varepsilon$, where Vol denotes n-dimensional volume.*

Algorithm 3. Given T, find ε-partition for $\text{Pr}_{\geq p}(\phi|\phi')$.

Input: FT T, formulae ϕ, ϕ', reals $p, \varepsilon \in (0,1]$.
Output: ε-partition $(S_{\text{yes}}, S_{\text{no}}, S_{\text{maybe}})$ for $\text{Pr}_{\geq p}(\phi|\phi')$.
Method:
$\mathscr{B}_{\text{maybe}} \leftarrow \{[0,1]^n\}$; $V_{\text{maybe}} \leftarrow 1$; $S_{\text{yes}}, S_{\text{no}} \leftarrow \varnothing$
while $V_{\text{maybe}} > \varepsilon$ **do**
    Pick $B = \prod_{i=1}^n [l_i, u_i]$ from $\mathscr{B}_{\text{maybe}}$ with maximal volume
    $\mathscr{B}_{\text{maybe}} \leftarrow \mathscr{B}_{\text{maybe}} \setminus \{B\}$
    $V_{\text{maybe}} \leftarrow V_{\text{maybe}} - \text{Vol}(B)$
    $\mathscr{B}_{\text{test}} \leftarrow \left\{ \prod_{i=1}^n I_i \ \middle|\ \forall i. I_i \in \{[l_i, \frac{l_i + u_i}{2}], [\frac{l_i + u_i}{2}, u_i]\} \right\}$
    for each $B' = \prod_{i=1}^n [l'_i, u'_i] \in \mathscr{B}_{\text{test}}$ **do**
        $A \leftarrow \{\overline{\rho} \in [0,1]^n \mid \forall i. \rho_i \in \{l'_i, u'_i\}\}$
        $p_{\min} \leftarrow \min_{\overline{\rho} \in A} \frac{\text{Algorithm } 1(T, \phi \wedge \phi')(\overline{\rho})}{\text{Algorithm } 1(T, \phi)(\overline{\rho})}$
        $p_{\max} \leftarrow \max_{\overline{\rho} \in A} \frac{\text{Algorithm } 1(T, \phi \wedge \phi')(\overline{\rho})}{\text{Algorithm } 1(T, \phi)(\overline{\rho})}$
        if $p \leq p_{\min}$ **then** $S_{\text{yes}} \leftarrow S_{\text{yes}} \cup B'$
        else if $p > p_{\max}$ **then** $S_{\text{no}} \leftarrow S_{\text{no}} \cup B'$
        else $\mathscr{B}_{\text{maybe}} \leftarrow \mathscr{B}_{\text{maybe}} \cup \{B'\}$; $V_{\text{maybe}} = V_{\text{maybe}} + \text{Vol}(B')$
        end if
    end for
end while
$S_{\text{maybe}} \leftarrow \bigcup \mathscr{B}_{\text{maybe}}$
return $(S_{\text{yes}}, S_{\text{no}}, S_{\text{maybe}})$

Algorithm 3. An algorithm finding a ε-partition for formulae of the form $\psi = \text{Pr}_{\geq p}(\phi|\phi')$ is given in Algorithm 3; it works as follows. We have a set $\mathscr{B}_{\text{maybe}}$ of candidate hypercubes, which starts as the singleton $\{[0,1]^n\}$. One by one, we take hypercubes B from $\mathscr{B}_{\text{maybe}}$, and divide them into 2^n smaller hypercubes. For each of the smaller hypercubes B', we check whether $\overline{\rho}, T \models \psi$ for all $\psi \in B'$; if so, we add B' to S_{yes}. If $\overline{\rho}, T \not\models \psi$ for all $\psi \in B'$, we add B' to S_{no}. If neither is true, then we add B' to $\mathscr{B}_{\text{maybe}}$, so that later it is split up again. The algorithm

stops when the joint volume of all hypercubes in $\mathscr{B}_{\mathrm{maybe}}$ is at most ε. Algorithm 3 has the argument ε to ensure that it terminates, as one can go on partitioning hypercubes indefinitely. The algorithm can easily be adapted to other stopping conditions, such as a maximal number of hypercubes. Literature in the area of parametric model checking explored this technique, also w.r.t. Markov decision processes (MDPs) [18, 22, 24, 30, 31]. However, we leverage the specific situation presented here to devise a less generic but more convenient algorithm. In fact, to check $\forall \overline{\rho} \in B'.\overline{\rho}, T \models \psi$, we use Theorem 1 (proof in Appendix B.1), which says that the minimum of $\frac{\Phi_T^*(\overline{\rho}, \phi \wedge \phi')}{\Phi_T^*(\overline{\rho}, \phi')}$ $\left(\text{computed as } \frac{\text{Algorithm } 1(T, \phi \wedge \phi')(\overline{\rho})}{\text{Algorithm } 1(T, \phi)(\overline{\rho})} \right)$ on B' is attained at one of its vertices. This means that we only need to check whether $\overline{\rho}, T \models \psi$ for the set A of vertices of B'. The same holds for checking $\forall \overline{\rho} \in B'.\overline{\rho}, T \not\models \psi$.

Theorem 1. *Let ϕ, ϕ' be layer-one formulae, and let $B \subseteq [0,1]^n$ be a hyperrectangle. Then $\frac{\Phi_T^*(\overline{\rho}, \phi \wedge \phi')}{\Phi_T^*(\overline{\rho}, \phi')}$ attains its minimum and maximum (as a function of $\overline{\rho}$) at one of the vertices of B.*

So far, we have assumed $\psi = \mathrm{Pr}_{\geq p}(\phi | \phi')$. Formulae of the form $\mathrm{Pr}_{=p}(\phi | \phi')$ and $\mathrm{IDP}(\phi, \phi')$ generally define hypersurfaces in $[0,1]^n$ rather than regions; these can be approximated by considering the set S_{maybe} of a ε-partition, which forms an open neighborhood of the actual hypersurface. Furthermore, one finds regions for $\neg \psi$ and $\psi \wedge \psi'$ by considering complements and intersections, respectively.

6.6   Algorithm 4: Checking PFL ψ-formulae over a FT for all $\overline{\rho}$

Overview. Given a FT T a layer-two formula ψ, we want to check if $T \models \psi$ for all $\overline{\rho}$. In this section we discuss two different approaches to answer this question, one derived from Algorithm 3 and one employing SAT solving.

Algorithm 4. Leveraging Algorithm 3, one could check whether $T \models \psi$ for all $\overline{\rho}$ by checking the parameter space in order to show that the negated formula $\neg \psi$ is unsatisfiable. If, on the other hand, we manage to find a candidate hypercube B' from $\mathscr{B}_{\mathrm{maybe}}$ such that $\forall \overline{\rho} \in B'.\overline{\rho}, T \models \neg \psi$ then we can exhibit a region that serves as a counterexample for our initial question. This procedure would be bound to approximate to a given level of precision, as previously discussed.

The second possibility is to resort to SMT solving. Again, our aim is to check if the negation of the given formula is unsatisfiable. First, we translate each of the inner ϕ_n layer-one formulae (e.g., inside $\mathrm{Pr}_{\bowtie p}(\phi \mid \phi')$ or $\mathrm{IDP}(\phi, \phi')$ operators) to BDDs, to then obtain representations of these BDDs as polynomials (see Algorithm 1). By comparing these to bounds set in $\mathrm{Pr}_{\bowtie p}(\phi \mid \phi')$ operators or to the semantics of $\mathrm{IDP}(\phi, \phi')$, one can represent the original negated formula $\neg \psi$ via (in)equalities between polynomials. We then use already available SMT solvers - such as SMT-RAT [16] - as a black box to handle such an encoding. If the input representation is satisfiable, the SMT solver returns an assignment of variables to values, i.e., a counterexample probability vector for our original question.

7 Conclusion and Future Work

Conclusion. We presented PFL, a probabilistic logic for FTs that enables the construction of complex queries that capture many relevant scenarios. Furthermore, we introduced LangPFL, a domain specific language for PFL to ease property specification. We showcased their usefulness with an application of PFL and LangPFL to a COVID19-related FT and to a FT for an oil/gas pipeline. Specified properties can then be checked via the model checking algorithms, that we presented alongside relevant theorems.

Future Work. Our work opens several relevant perspectives for future research. First, it would be interesting to extend PFL to consider timed behaviours to further extend quantitative analysis capabilities. Secondly, it would be possible to extend PFL in order to consider dynamic gates in FTs. This further validates our first point: to handle dynamic gates in dynamic FTs it would be very natural to have a logic that can express temporal properties, moving more in the direction of LTL [40] or CTL [14] or their timed variants TLTL [41] and TCTL [1]. Moreover, it is foreseeable to extend the proposed framework to security variants of FTs, attack trees (ATs) [7,12,27,44], and to their combinations, e.g., attack-fault trees (AFTs) [32]. Another relevant area is concerned with automatic inference of FTs: further research could explore inference on PFL formulae, e.g. based on genetic algorithms [29] or dedicated methods [35]. Lastly, developing an implementation of this logic could further propel usability of PFL and LangPFL by providing hands-on feedback from domain experts acquainted with FTA.

A Appendix: Algorithms and Additional Definitions for Layer One Formulae

Following, operations between BDDs are represented by **bold** operands e.g., $\wedge, \vee$. Algorithms to conduct these operations on BDDs can be found in [2,5]. Given a set of variables $V = \{v_1, \ldots, v_n\}$, existential quantification (needed to translate part of the semantics of MCS operator) can be defined as follows: $\exists v.\mathbf{B} = \text{RESTRICT}(\mathbf{B}, v, 0) \vee \text{RESTRICT}(\mathbf{B}, v, 1)$; $\exists V.\mathbf{B} = \exists v_1.\exists v_2. \ldots \exists v_n.\mathbf{B}$.

A.1 Translating FTs to BDDs

Ψ_{FT} is defined as follows:

Definition 7. *The translation function of a FT T is a function $\Psi_{FT_T} \colon \mathtt{E} \to \mathtt{BDD}$ that takes as input an element $e \in \mathtt{E}$. With $e' \in ch(e)$, we can define Ψ_{FT_T}:*

$$\Psi_{FT_T}(e) = \begin{cases} \overline{\mathbf{B}}(e) & if\ e \in \mathtt{BE} \\ \bigvee \Psi_{FT_T}(e') & if\ e \in \mathtt{IE}\ and\ t(e) = \mathtt{OR} \\ \bigwedge \Psi_{FT_T}(e') & if\ e \in \mathtt{IE}\ and\ t(e) = \mathtt{AND} \\ \bigvee_{\substack{n_1,\ldots,n_k \\ n_1 < \ldots < n_k}} \bigwedge_{i=1}^{k} \Psi_{FT_T}(e'_{n_i}) & if\ e \in \mathtt{IE}\ and\ t(e) = \mathtt{VOT}(k/N) \end{cases}$$

where $\overline{\mathbf{B}}(v)$ is a BDD with a single node in which $Low(v) = \mathtt{0}$ and $High(v) = \mathtt{1}$.

A.2   Algorithm 5: Translating FTs/Formulae to BDDs

Following, the recursion scheme taken from [37] to translate FTs and layer one formulae is presented.

Algorithm 5. Given ϕ and T, compute $\underline{\mathbf{B}}_T(\phi)$

Input: FT T, formula ϕ

Output: $\underline{\mathbf{B}}_T(\phi)$

Method: Compute $\underline{\mathbf{B}}_T(\phi)$ according to the recursion scheme below. Store intermediate results $\underline{\mathbf{B}}_T(\cdots)$ and $\Psi_{FT_T}(\cdots)$ in a cache in case they are used several times.

> **Recursion scheme:**
>
> $\underline{\mathbf{B}}_T(e):$ $\qquad\qquad \Psi_{FT_T}(e)$
>
> $\underline{\mathbf{B}}_T(\neg\phi):$ $\qquad\quad \neg(\underline{\mathbf{B}}_T(\phi))$
>
> $\underline{\mathbf{B}}_T(\phi \wedge \phi'):$ $\qquad \underline{\mathbf{B}}_T(\phi) \wedge \underline{\mathbf{B}}_T(\phi')$
>
> $\underline{\mathbf{B}}_T(\phi[e_i \mapsto 0]):$ $\quad \text{RESTRICT}(\underline{\mathbf{B}}_T(\phi), e_i, 0)$
>
> $\underline{\mathbf{B}}_T(\phi[e_i \mapsto 1]):$ $\quad \text{RESTRICT}(\underline{\mathbf{B}}_T(\phi), e_i, 1)$
>
> $\underline{\mathbf{B}}_T(\text{MCS}(\phi)):$ $\quad \underline{\mathbf{B}}_T(\phi) \wedge (\neg\exists \mathtt{V}'.\underline{\mathbf{B}}_T(\mathtt{V}' \subset \mathtt{V})\wedge$
>
> $\qquad\qquad\qquad\quad \underline{\mathbf{B}}_T(\phi)[\mathtt{V} \curvearrowright \mathtt{V}'])$ where:
>
> $\qquad\qquad\qquad\quad \underline{\mathbf{B}}_T(\mathtt{V}' \subset \mathtt{V}) \equiv \underline{\mathbf{B}}_T(\bigwedge_k v'_k \Rightarrow v_k)\wedge$
>
> $\qquad\qquad\qquad\quad \underline{\mathbf{B}}_T(\bigvee_k v'_k \neq v_k)$

---

where $\underline{\mathbf{B}}_T(\phi)[\mathtt{V} \curvearrowright \mathtt{V}']$ indicates the BDD $\underline{\mathbf{B}}_T(\phi)$ in which every variable $v_k \in \mathtt{V}$ is renamed to its primed $v'_k \in \mathtt{V}'$.

A.3   Algorithm 6: Model Checking PFL over a FT and a $\overline{b}$

Overview. As per [37], given a specific vector $\overline{b}$, a FT T and a layer one formula ϕ, this algorithm showcases how to check if $\overline{b}, T \models \phi$. To do so, we translate the given formula to a BDD and then we walk down the BDD from the root node following truth assignments given in the specific vector $\overline{b}$.

Algorithm 6. Algorithm 6 shows an algorithm to check whether $\overline{b}, T \models \phi$, given a status vector $\overline{b}$, a FT T and a formula ϕ. A BDD for the formula ϕ is computed with regard to the structure function of the given FT T i.e., we compute $\underline{\mathbf{B}}_T(\phi)$ as per Algorithm 5. Subsequently, the algorithm walks down the BDD following the Boolean assignments given in $\overline{b}$: if the i-th element of $\overline{b}$ is set to 0 then the next node in the path will be given by $Low(w_i)$, if it is set to 1 then the next node will be $High(w_i)$. When the algorithm reaches a terminal node it returns *True* if its value is one - i.e., if $\overline{b}, T \models \phi$ - and *False* otherwise.

Algorithm 6. Check if $\overline{b}$, $T \models \phi$, given $\overline{b}$, T and ϕ.

Input: boolean vector $\overline{b}$, FT T and a formula ϕ
Output: *True* iff $\overline{b}$, $T \models \phi$, *False* otherwise.
Method: compute $\underline{\mathbf{B}}_T(\phi)$
Starting from BDD root,
while current node w_i of $\underline{\mathbf{B}}_T(\phi) \notin W_t$ **do:**
   if $b_i \in \overline{b} = 0$ **then:**
      $w_i = Low(w_i)$
   else if $b_i \in \overline{b} = 1$ **then:**
      $w_i = High(w_i)$
   end if
end while
if $w_i = 0$ **then:**
   return False
else if $w_i = 1$ **then:**
   return True
end if

A.4 Algorithm 7: Computing all Satisfying Vectors

Overview. Given a FT T and a formula ϕ, we now want to compute all vectors $\overline{b}$ such that $\overline{b}$, $T \models \phi$. In this scenario no Boolean vector is given. Thus, we need to construct the BDD for the given formula and then collect every path that leads to the terminal 1 to compute all satisfying vectors $[\![\overline{b}]\!]_T$ for the given formula.

Algorithm 7. To achieve the desired outcome we will construct the BDD $\underline{\mathbf{B}}_T(\phi)$ for the given formula following Algorithm 5. Then, the algorithm will walk down the BDD and store all the paths that lead to the terminal node 1. These paths represent all the status vectors that satisfy our formula ϕ. The value for the elements of each vector is set to 0 or 1 if the stored path follows respectively the low or high edge of the collected elements of the BDD. After computing the BDD for a given ϕ, ALLSAT [2] will achieve the desired outcome. This algorithm returns exactly all the satisfying assignments for a given BDD, i.e., in our case, all the Boolean vectors that satisfy our formula.

B Appendix: Proofs

B.1 Proof for Theorem 1

Proof. For a layer one formula ϕ and $\overline{\rho} \in B$, one can express

$$\Phi_T^*(\mu_{\overline{\rho}}, \phi) = \sum_{\substack{b \in \mathbb{B}^n : \\ \Phi_T(b, \phi) = 1}} \prod_{i=1}^{n} \rho_i^{b_i} (1 - \rho_i)^{1 - b_i}. \tag{1}$$

This is a polynomial in the n variables ρ_i. Each summand has degree 1 in each ρ_i, hence $\Phi_T^*(\mu_{\overline{\rho}}, \phi)$ can be written as

$$\Phi_T^*(\mu_{\overline{\rho}}, \phi) = \sum_{w \in \{0,1\}^n} c_w^h \prod_{i=1}^n \rho_i^{w_i} \tag{2}$$

for some constants $c_w^h \in \mathbb{R}$. Now fix an i, and let ϕ, ϕ' be two Boolean formulae; then we can write $\frac{\Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}}, \phi')} = \frac{A\rho_i + B}{C\rho_i + D}$ for some polynomials A, B, C, D in the variables $\rho_1, \ldots, \rho_{i-1}, \rho_{i+1}, \ldots, \rho_n$. In particular, we have

$$\frac{\partial}{\partial \rho_i} \frac{\Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}}, \phi')} = \frac{AD - BC}{(C\rho_i + D)^2}. \tag{3}$$

The sign of this partial derivative does not depend on the value of ρ_i. In particular, when all other $\rho_{i'}$ are fixed, this expression is maximized on an interval when ρ_i is at one of the boundary points of that interval.

Now let us return to the setting of the Theorem; we will prove it for the maximum only as the minimum is proved analogously. Let Let $B = \prod_i [l_i, u_i]$ and let $\overline{\rho} \in \prod_i [l_i^{-1}, l_i^+]$; our aim is to find a vertex $\overline{\rho}'$ such that $\frac{\Phi_T^*(\mu_{\overline{\rho}}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}}, \phi')} \leq \frac{\Phi_T^*(\mu_{\overline{\rho}'}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}'}, \phi')}$. To do so, we construct a sequence $\overline{\rho}_0, \overline{\rho}_1, \ldots, \overline{\rho}_n$ with the following properties:

1. $\overline{\rho}_0 = \overline{\rho}$;
2. $\frac{\Phi_T^*(\mu_{\overline{\rho}_i}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_i}, \phi')} \leq \frac{\Phi_T^*(\mu_{\overline{\rho}_{i+1}}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_{i+1}}, \phi')}$ for $i < n$;
3. $\rho_{i,i'} \in \{l_{i'}, u_{i'}\}$ for $i' \leq i \leq n$.

This ensures that $\overline{\rho}' := \overline{\rho}_n$ has the required property. We define each $\overline{\rho}_i$ from $\overline{\rho}_{i-1}$ as follows: define $\overline{\rho}_i^-, \overline{\rho}_i^+ \in [l_i, u_i]$ by

$$\overline{\rho}_{i,i'}^\bullet = \begin{cases} l_i, & \text{if } \bullet = - \text{ and } i' = i, \\ u_i, & \text{if } \bullet = + \text{ and } i' = i, \\ \rho_{i-1,i'}, & \text{if } i' \neq i. \end{cases}$$

By the discussion following (3), one has $\frac{\Phi_T^*(\mu_{\overline{\rho}_i}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_i}, \phi')} \leq \max$ $\left\{ \frac{\Phi_T^*(\mu_{\overline{\rho}_{i+1}^-}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_{i+1}^-}, \phi')}, \frac{\Phi_T^*(\mu_{\overline{\rho}_{i+1}^+}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_{i+1}^+}, \phi')} \right\}$. Take $\overline{\rho}_{i+1} \in \{\overline{\rho}_{i+1}^-, \overline{\rho}_{i+1}^+\}$ to maximize $\frac{\Phi_T^*(\mu_{\overline{\rho}_{i+1}}, \phi \wedge \phi')}{\Phi_T^*(\mu_{\overline{\rho}_{i+1}}, \phi')}$, then this satisfies conditions 1–3 above.

References

1. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Inf. Comput. **104**(1), 2–34 (1993)

2. Andersen, H.R.: An introduction to binary decision diagrams. Lecture notes, available online, IT University of Copenhagen, p. 5 (1997)
3. Bakeli, T., Hafidi, A.A., et al.: COVID-19 infection risk management during construction activities: an approach based on fault tree analysis (FTA). J. Emerg. Manage. **18**(7), 161–176 (2020)
4. Basgöze, D., Volk, M., Katoen, J., Khan, S., Stoelinga, M.: BDDs strike back - efficient analysis of static and dynamic fault trees. In: Deshmukh, J.V., Havelund, K., Perez, I. (eds.) NFM 2022. LNCS, vol. 13260, pp. 713–732. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-06773-0_38
5. Ben-Ari, M.: Mathematical Logic for Computer Science. Springer, Heidelberg (2012). https://doi.org/10.1007/978-1-4471-4129-7
6. Bieber, P., Castel, C., Seguin, C.: Combination of fault tree analysis and model checking for safety assessment of complex system. In: Bondavalli, A., Thevenod-Fosse, P. (eds.) EDCC 2002. LNCS, vol. 2485, pp. 19–31. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-36080-8_3
7. Bobbio, A., Egidi, L., Terruggia, R.: A methodology for qualitative/quantitative analysis of weighted attack trees. IFAC Proc. Vol. **46**(22), 133–138 (2013). https://doi.org/10.3182/20130904-3-UK-4041.00007
8. Boudali, H., Crouzen, P., Stoelinga, M.: Dynamic fault tree analysis using input/output interactive Markov chains. In: DSN, pp. 708–717. IEEE Computer Society (2007). https://doi.org/10.1109/DSN.2007.37
9. Bozzano, M., Cimatti, A., Katoen, J., Nguyen, V.Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended AADL models. Comput. J. **54**(5), 754–775 (2011). https://doi.org/10.1093/comjnl/bxq024
10. Brace, K., Rudell, R., Bryant, R.: Efficient implementation of a BDD package. In: 27th ACM/IEEE Design Automation Conference, pp. 40–45 (1990). https://doi.org/10.1109/DAC.1990.114826
11. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9
12. Budde, C.E., Stoelinga, M.: Efficient algorithms for quantitative attack tree analysis. In: 2021 IEEE 34th Computer Security Foundations Symposium (CSF), pp. 1–15 (2021). https://doi.org/10.1109/CSF51468.2021.00041
13. Clark, P., Harrison, P., Jenkins, T., Thompson, J.A., Wojcik, R.H., et al.: Acquiring and using world knowledge using a restricted subset of English. In: Flairs Conference, pp. 506–511 (2005)
14. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Logic of Programs 1981. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982). https://doi.org/10.1007/BFb0025774
15. Conrad, E., Titolo, L., Giannakopoulou, D., Pressburger, T., Dutle, A.: A compositional proof framework for FRETish requirements. In: Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs, pp. 68–81 (2022)
16. Corzilius, F., Kremer, G., Junges, S., Schupp, S., Ábrahám, E.: `SMT-RAT`: an open source `C++` toolbox for strategic and parallel SMT solving. In: Heule, M., Weaver, S. (eds.) SAT 2015. LNCS, vol. 9340, pp. 360–368. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24318-4_26

17. Crapo, A., Moitra, A., McMillan, C., Russell, D.: Requirements capture and analysis in ASSERT (TM). In: 2017 IEEE 25th International Requirements Engineering Conference (RE), pp. 283–291. IEEE (2017)
18. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J.P., Topcu, U.: Convex optimization for parameter synthesis in MDPs. IEEE Trans. Autom. Control **67**, 6333–6348 (2021)
19. Déharbe, D., Shankar, S., Clarke, E.M.: Model checking VHDL with CV. In: Gopalakrishnan, G., Windley, P. (eds.) FMCAD 1998. LNCS, vol. 1522, pp. 508–514. Springer, Heidelberg (1998). https://doi.org/10.1007/3-540-49519-3_33
20. Dutuit, Y., Rauzy, A.: A linear-time algorithm to find modules of fault trees. IEEE Trans. Reliab. **45**(3), 422–425 (1996)
21. Ericson, C.A.: Fault tree analysis. In: System Safety Conference, vol. 1, pp. 1–9 (1999)
22. Gainer, P., Hahn, E.M., Schewe, S.: Accelerated model checking of parametric Markov chains. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 300–316. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_18
23. Giannakopoulou, D., Mavridou, A., Rhein, J., Pressburger, T., Schumann, J., Shi, N.: Formal requirements elicitation with FRET. In: International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ-2020). No. ARC-E-DAA-TN77785 (2020)
24. Hahn, E.M., Han, T., Zhang, L.: Synthesis for PCTL in parametric Markov decision processes. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 146–161. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-20398-5_12
25. Hansen, K.M., Ravn, A.P., Stavridou, V.: From safety analysis to software requirements. IEEE Trans. Software Eng. **24**(7), 573–584 (1998)
26. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994). https://doi.org/10.1007/BF01211866
27. Hermanns, H., Krämer, J., Krčál, J., Stoelinga, M.: The value of attack-defence diagrams. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 163–185. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49635-0_9
28. International Standardization Organization: ISO/DIS 26262: Road vehicles, functional safety (2018). https://www.iso.org/standard/68383.html
29. Jimenez-Roa, L., Heskes, T., Tinga, T., Stoelinga, M.: Automatic inference of fault tree models via multi-objective evolutionary algorithms. IEEE Trans. Dependable Secure Comput., 1–12 (2021). https://doi.org/10.1109/TDSC.2022.3203805
30. Junges, S., et al.: Parameter synthesis for Markov models. arXiv preprint arXiv:1903.07993 (2019)
31. Katoen, J.P.: The probabilistic model checking landscape. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, pp. 31–45 (2016)
32. Kumar, R., Stoelinga, M.: Quantitative security and safety analysis with attack-fault trees. In: Proceedings of the 18th IEEE International Symposium on High Assurance Systems Engineering (HASE 2017), pp. 25–32. HASE, IEEE, USA (2017). https://doi.org/10.1109/HASE.2017.12
33. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_47

34. Moszkowski, B.: A temporal logic for multi-level reasoning about hardware. Technical report, STANFORD UNIV CA (1982)
35. Nauta, M., Bucur, D., Stoelinga, M.: LIFT: learning fault trees from observational data. In: McIver, A., Horvath, A. (eds.) QEST 2018. LNCS, vol. 11024, pp. 306–322. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99154-2_19
36. Nicoletti, S., Hahn, E., Stoelinga, M.: A logic to reason about fault trees. Interview Report. https://www.utwente.nl/en/eemcs/fmt/research/files/ft-logic-interview-domain-expert.pdf
37. Nicoletti, S., Hahn, E., Stoelinga, M.: BFL: a logic to reason about fault trees. In: (DSN), pp. 441–452. IEEE/EUCA (2022). https://doi.org/10.1109/DSN53405.2022.00051
38. Ognjanovic, Z.: Discrete linear-time probabilistic logics: completeness, decidability and complexity. J. Log. Comput. **16**(2), 257–285 (2006). https://doi.org/10.1093/logcom/exi077
39. Pease, A., Murray, W.: An English to logic translator for ontology-based knowledge representation languages. In: 2003 Proceedings of the International Conference on Natural Language Processing and Knowledge Engineering, pp. 777–783. IEEE (2003)
40. Pnueli, A.: The temporal logic of programs. In: 18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October–1 November 1977, pp. 46–57. IEEE Computer Society (1977). https://doi.org/10.1109/SFCS.1977.32
41. Raskin, J.F.: Logics, automata and classical theories for deciding real time. Ph.D. thesis, Facultés universitaires Notre-Dame de la Paix, Namur (1999)
42. Rauzy, A.: New algorithms for fault trees analysis. Reliab. Eng. Syst. Saf. **40**(3), 203–211 (1993). https://doi.org/10.1016/0951-8320(93)90060-C
43. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15–16**, 29–62 (2015). https://doi.org/10.1016/j.cosrev.2015.03.001
44. Schneier, B.: Attack trees. Dr. Dobb's J. **24**(12), 21–29 (1999)
45. Stamatelatos, M., Vesely, W., Dugan, J., Fragola, J., Minarick, J., Railsback, J.: Fault tree handbook with aerospace applications. Prepared for NASA Office of Safety and Mission Assurance (2002)
46. Thums, A., Schellhorn, G.: Model checking FTA. In: Araki, K., Gnesi, S., Mandrioli, D. (eds.) FME 2003. LNCS, vol. 2805, pp. 739–757. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45236-2_40
47. Volk, M., Junges, S., Katoen, J.: Fast dynamic fault tree analysis by model checking techniques. IEEE Trans. Ind. Inform. **14**(1), 370–379 (2018). https://doi.org/10.1109/TII.2017.2710316
48. Walker, M.D.: Pandora: a logic for the qualitative analysis of temporal fault trees. Ph.D. thesis, The University of Hull (2009)
49. White, C., Schwitter, R.: An update on PENG light. In: Proceedings of the Australasian Language Technology Association Workshop 2009, pp. 80–88 (2009)
50. Yuhua, D., Datao, Y.: Estimation of failure probability of oil and gas transmission pipelines by fuzzy fault tree analysis. J. Loss Prev. Process Ind. **18**(2), 83–88 (2005)