FDS-ML: A New Modeling Formalism for Probabilistic Risk and Safety Analyses

Liu Yang(✉) and Antoine Rauzy

Department of Mechanical and Industrial Engineering (MTP),
Norwegian University of Science and Technology (NTNU),
Trondheim, Norway
liu.yang@ntnu.no

Abstract. In this article, we present FDS-ML, a new modeling formalism dedicated to probabilistic risk and safety analyse. FDS-ML relies on the notion of finite degradation structures, an algebraic framework recently introduced by the authors. FDS-ML provides a simple and clear way to design combinatorial models.

The assessment of FDS-ML models relies on the decision diagram technology. Classical concepts defined for fault trees, such as those of minimal cutsets, availability, reliability and importance measures, can be lifted up to finite degradation structures and computed by means of decision diagram algorithms.

The article aims at presenting the most important ideas underlying FDS-ML and its implementation. It illustrates the practical interest of the proposed approach by means of a case study stemmed from the ISO/TR 12489 standard.

Keywords: Probabilistic risk and safety analyses · Modeling language · Finite degradation structures · Combinatorial models · Decision diagrams

1 Introduction

Probabilistic risk and safety analyses are used in virtually all industries to determine whether the risk of operating complex technical systems (aircraft, nuclear power plants, offshore platforms...) is low enough to be socially acceptable. A large number of modeling formalisms have been proposed to carry out these analyses. They can be roughly split into two categories: combinatorial formalisms and stochastic discrete event systems. The first category gathers Boolean formalisms such as fault trees [11], reliability block diagrams [4] as well as so-called multistate systems [12]. In combinatorial formalisms, the state of the system is described as a combination of the states of its components. The second category gathers formalisms such as Markov chains, stochastic Petri nets, stochastic automata networks as well as high level modeling languages such as AltaRica 3.0 [1]. They provide analysts with a much higher expressive power than the former, but the price to pay is a dramatic increase of the computational complexity of assessments.

Finite degradation structures (FDS) have been recently introduced by the authors as a unified algebraic framework for combinatorial models [13,14]. FDS generalize existing combinatorial formalisms (both for Boolean and multistate systems) at no algorithmic cost. Classical concepts defined for fault trees—minimal cutsets, availability, reliability, importance measures,...—can be lifted up to FDS.

FDS-ML is a small domain specific modeling language designed on top of FDS. It makes it possible to define domains (finite degradation structures), operators, variables, formulas and eventually sets of equations.

We developed a prototype assessment engine for FDS-ML models. Algorithms implemented in this prototype rely on the decision diagram technology. As fault trees, the assessment process works in two steps: first, a decision diagram is built for the (equivalent of the) top event of the model. Second minimal cutsets and probabilistic indicators are calculated by traversing this diagram.

This article aims at presenting theoretical foundations of FDS-ML, as well as the current version of the language. It describes also assessment algorithms. Finally, it shows the interest of the proposed approach by means of a use case stemmed from the ISO/TR 12489 standard [6].

The remainder of this article is structured as follows. Section 2 introduces the use case we shall throughout the article to illustrate the concepts and algorithms. Section 3 presents FDS. Section 4 presents the language FDS-ML. Section 5 describes assessment algorithms and the data structures they rely on. Section 6 presents some experimental results obtained on the case study. Finally, Sect. 7 concludes the article.

2 Illustrative Use Case

2.1 Presentation

Safety instrumented systems (SIS) are designed to keep an equipment under control in a safe state when some abnormal conditions occur. As illustrative use case, we shall consider the TA4 system of ISO/TR 12489 [6], which pictured Fig. 1.

The objective of this SIS is to protect a pipe section from overpressures. It involves seven main components: three sensors (S1, S2 and S3), two logic solvers (LS1 and LS2) and two actuators (the isolation valves V1 and V2) which are activated via the solenoid valves (SV1, SV2 and SV3). When the sensors detect an overpressure in the protected section, the logic solvers send a control signal to the solenoid valves which close isolation valves so to release the pressure. The logic solver LS2 works according to a 1-out-of-2 logic, i.e. that it sends the order to close the valves if at least one out of two sensors S2 and S3 detects an overpressure.

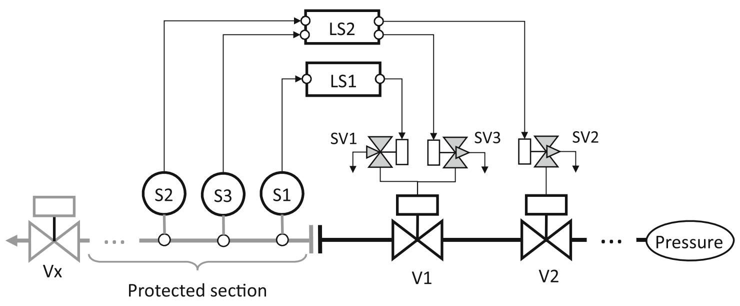

Fig. 1. Architecture of the safety instrumented system in TA4 of ISO/TR 12489

According to the standard IEC61508 [5], failure modes of the components of a SIS can be classified along two directions: safe versus dangerous failure modes and detected versus undetected failure modes.

In our example, safe failure modes are those which contribute to close the isolation valves, even though there is no overpressure (spurious triggers), while dangerous faire modes are those which contribute to keep the isolation valves open, even though there is an overpressure. Logic solvers embed autotest facilities so that their failures are immediately detected. On the contrary, failure of valves remain undetected between two maintenance interventions. Failures of sensors may be detected or not.

ISO/TR 12489 makes the additional following assumptions.

– The three solenoid valves are perfectly reliable.
– All other components may fail (independently). Their probabilities of failure follow negative exponential distributions. The parameters of these distributions are given Table 1. Safe failures are always detected.
– The system is maintained once a year (once in 8760 h). The production is stopped during the maintenance. Components are as good as new after the maintenance.

Table 1. TA4 reliability parameters

Parameter	Sensor	Logic solver	Isolation valve
Dangerous undetected failure rate	$3.0 \times 10^{-7}\,\text{h}^{-1}$	NA	$2.9 \times 10^{-6}\,\text{h}^{-1}$
Dangerous detected failure rate	$3.0 \times 10^{-5}\,\text{h}^{-1}$	$6.0 \times 10^{-7}\,\text{h}^{-1}$	NA
Safe failure rate	$3.0 \times 10^{-5}\,\text{h}^{-1}$	$3.0 \times 10^{-5}\,\text{h}^{-1}$	2.9×10^{-4}

It is not possible to compare safe failures and dangerous failures, because the risk they represent, both in terms of frequency of occurrence and severity of consequences, are very different On the one hand, spurious triggers of SIS have a strong economic impact, but indeed no impact on safety. On the other hand, dangerous failures have an impact on safety. If they remain undetected, they may lead to a catastrophic accident.

2.2 Modeling

According to what precedes, we shall consider three failure modes: safe-failure, dangerous-detected-failure and dangerous-undetected-failure.

Usually, the different failure modes are analysed one-by-one. In our case, this means that one would design a dedicated fault tree to describe safe-failures of the system, another one for dangerous-detected-failures and a third one for dangerous-undetected-failures.

The modeling framework presented in this article makes it possible to study different failure modes by means of a unique model. This model makes in turn possible to study, for instance, the combination of a safe-failure of a sensor and a dangerous-detected-failure of a valve. To the best of authors' knowledge, such combinations have not been formally defined in the standard nor in any other previous work.

We can read Fig. 1 as a block diagram. Each component can be seen as a basic block, with an internal state, some input and some output flows. Failures propagates through the block diagram. Therefore, both states and flows may take one of the four values: W (working), safe-failure (Fs), dangerous-detected-failure (Fdd) and dangerous-undetected-failure (Fdu).

Two fundamental operations are performed on states and flows: series composition, denoted by $\ogreaterthan$, and parallel composition, denoted by $\|$. These operators are defined Table 2.

Table 2. Definition of $\ogreaterthan$ and $\|$

$u \ogreaterthan v$		v			
		W	Fs	Fdd	Fdu
u	W	W	Fs	Fdd	Fdu
	Fs	Fs	Fs	Fdd	Fdd
	Fdd	Fdd	Fs	Fdd	Fdd
	Fdu	Fdu	Fs	Fdd	Fdu

$u \parallel v$		v			
		W	Fs	Fdd	Fdu
u	W	W	Fs	W	W
	Fs	Fs	Fs	Fs	Fs
	Fdd	W	Fs	Fdd	Fdu
	Fdu	W	Fs	Fdu	Fdu

It is easy to verify that the series operator $\ogreaterthan$ is not commutative but associative and that the parallel operator $\|$ is both commutative and associative.

Using $\ogreaterthan$ and $\|$, the model for the whole SIS could be as sketched Fig. 2.

In the remaining part of this article, we shall study how to implement the above ideas in the framework of FDS-ML.

$$
\boxed{
\begin{array}{rlrl}
S1.in & := W & S1.out & := S1.in \ogreaterthan S1.state \\
S2.in & := W & S2.out & := S2.in \ogreaterthan S2.state \\
S3.in & := W & S3.out & := S3.in \ogreaterthan S3.state \\
LS1.in & := S1.out & LS1.out & := LS1.in \ogreaterthan LS1.state \\
LS2.in & := S2.out \parallel S3.out & LS2.out & := LS2.in \ogreaterthan LS2.state \\
V1.in & := LS1.out \parallel LS2.out & V1.out & := V1.in \ogreaterthan V1.state \\
V2.in & := LS2.out & V2.out & := V2.in \ogreaterthan V2.state \\
SIS & := V1.out \parallel V2.out & &
\end{array}
}
$$

Fig. 2. Model for the SIS TA4 of ISO/TR 12489

3 Finite Degradation Structures

3.1 Definition

Finite degradation structures rely on the algebraic notion partially ordered sets.

A *partially ordered set* (poset) is pair $\langle D, \sqsubseteq \rangle$, where D is a set and $\sqsubseteq$ is a binary relation over D, such that $\forall a, b, c \in D$:

- $a \sqsubseteq a$ (*Reflexivity*);
- if $a \sqsubseteq b$ and $b \sqsubseteq c$, then $a \sqsubseteq c$ (*Transitivity*);
- if $a \sqsubseteq b$ and $b \sqsubseteq a$, then $a = b$ (*Antisymmetry*).

A *finite degradation structure* is such poset $\langle D, \sqsubseteq \rangle$. The elements in D represent the states of a component, while the partial order $\sqsubseteq$ represents the *degradation order* amongst these states, interpreted informally as "less or equally degraded than". For instance, a working state W is less degraded than the failed state F, therefore $W \sqsubset F$.

We require moreover the poset $\langle D, \sqsubseteq \rangle$ to have a unique least element, denoted $\bot$, that represents the initial working state. In other words, a finite degradation structure is a *(meet-)semi-lattice.*

Four FDS are graphically represented Fig. 3. These diagrams are called Hasse diagrams. Vertices represent states and the relation $a \sqsubset b$ is represented by drawing as a line segment that goes upward from a to b. For simplicity, we name the FDS in (a), (b), (c) and (d) by **WF**, **WDF**, **SWF** and **W3F**.

W3F is essentially the FDS we used Sect. 2. Its least element is the working state W. The degradation order is described by the inequalities $W \sqsubset Fs$, $W \sqsubset Fdd$ and $Fdd \sqsubset Fdu$. Fs is incomparable with Fdd and Fdu since they correspond to radically different situations. We have $Fdd \sqsubset Fdu$ because an undetected failure is always more dangerous than a detected one.

W3F is indeed not the only way to describe the states of SIS.

3.2 Products and Abstractions

Let $\mathcal{S} : \langle D_S, \sqsubseteq_S, \bot_S \rangle$ and $\mathcal{T} : \langle D_T, \sqsubseteq_T, \bot_T \rangle$ be two FDS. Then the product $\mathcal{S} \otimes \mathcal{T}$ of $\mathcal{S}$ and $\mathcal{T}$ is the FDS $\langle D, \sqsubseteq, \bot \rangle$ such that,

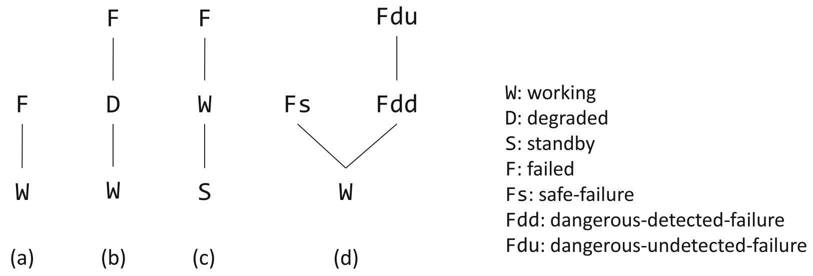

Fig. 3. Graphical representation of FDS.

– $D = D_S \times D_T$, where $\times$ stands for the Cartesian product.
– $\forall \langle x_S, x_T \rangle, \langle y_S, y_T \rangle \in D$, $\langle x_S, x_T \rangle \sqsubseteq \langle y_S, y_T \rangle \Leftrightarrow x_S \sqsubseteq_S y_S \wedge x_T \sqsubseteq_T y_T$.
– $\bot = \langle \bot_S, \bot_T \rangle$.

Let $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ be three FDS. It is easy to check that $\mathcal{R} \otimes \mathcal{S}$ and $\mathcal{S} \otimes \mathcal{R}$ on the one hand, $\mathcal{R} \otimes (\mathcal{S} \otimes \mathcal{T})$ and $(\mathcal{R} \otimes \mathcal{S}) \otimes \mathcal{T}$ on the other hand are equal up to an isomorphism. In this sense, the product of FDS is commutative and associative.

Let $\mathcal{S} : \langle D_S, \sqsubseteq_S, \bot_S \rangle$ and $\mathcal{T} : \langle D_T, \sqsubseteq_T, \bot_T \rangle$ be two FDS, then $\mathcal{T}$ is an *abstraction* of $\mathcal{S}$, which is denoted $\mathcal{S} \twoheadrightarrow \mathcal{T}$, if there exists a surjective structure preserving mapping from $\mathcal{S}$ to $\mathcal{T}$, i.e. a function $\varphi : \mathcal{S} \rightarrow \mathcal{T}$ such that:

– $x \sqsubseteq_S y \Rightarrow \varphi(x) \sqsubseteq_T \varphi(y)$ for all $x, y \in D_S$.
– $\varphi(\bot_S) = \bot_T$.
– $\forall y \in D_T, \exists x \in D_S$ such that $\varphi(x) = y$.

Let $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ be three FDS. It is easy to check that if $\mathcal{R} \twoheadrightarrow \mathcal{S}$ and $\mathcal{S} \twoheadrightarrow \mathcal{T}$ then $\mathcal{R} \twoheadrightarrow \mathcal{T}$ (the composition of abstraction is an abstraction).

Taken together products and abstractions make possible to define the state of a system as a combination of the states of its component.

3.3 Finite Degradation Models

Let $\mathbf{O}$ be a set of operators defined over finite degradation structures and let $\mathbf{V}$ be a set of variables. We assume that each variable v of $\mathbf{V}$ takes its value into some finite degradation structure, called the domain of v and denoted $\text{dom}(v)$.

Formulas over $\mathbf{O}$ and $\mathbf{V}$ are built as usual, verifying that they are well-typed, i.e. that each operator has the correct number of arguments and that its arguments are of the correct types.

We denote by $\text{var}(f)$ the set of variables showing up in the formula f.

From now, we shall assume that $\mathbf{V}$ is decomposed into two distinct subsets $\mathbf{S}$ and $\mathbf{F}$, i.e. $\mathbf{V} = \mathbf{S} \uplus \mathbf{F}$. Variables of $\mathbf{S}$ and $\mathbf{F}$ are called respectively state and flow variables. State variables play the role of basic events in fault trees while flow variables play the role of intermediated events.

A *finite degradation model* (FDM) ϕ over $\mathbf{O}$ and $\mathbf{V}$ is a set of equations of the form:

$$\phi : \left\{ \begin{array}{c} w_1 := f_1 \\ w_2 := f_2 \\ \vdots \\ w_n := f_n \end{array} \right\} \tag{1}$$

such that:

- the w_i's are variables of $\mathbf{F}$;
- the f_i's are well-typed formulas over $\mathbf{O}$ and $\mathbf{V}$;
- for any $w \in \mathbf{F}$, there is exactly one equation $w := f$ in the set whose left hand side member is w. We say that this equation defines w and that f is the definition of w.

Let w be a flow variable defined by the equation $w := f$ and let v be a variable. We say that f depends on the variable v if either $v \in \mathrm{var}(f)$ or there is a flow variable $u \in \mathrm{var}(f)$ such that u depends on v.

A finite degradation model is data-flow if no variable depend on itself. In the sequel, we shall only consider data-flow models.

The set of equations presented Fig. 2 is thus a finite degradation model. The variables $X.state$ are state variables and the variables $X.in$ and $X.out$ are flow variables. All of the variables of this model take their values into the finite degradation structure $\mathbf{W3F}$.

A finite degradation model over $\mathbf{O}$ and $\mathbf{V} = \mathbf{S} \uplus \mathbf{F}$ can thus be interpreted as a function from $\bigotimes_{v\in\mathbf{S}} \mathrm{dom}(v)$ into $\bigotimes_{w\in\mathbf{F}} \mathrm{dom}(w)$ (the data-flow property warranties that this construction is possible and uniquely defined).

If operators are correctly chosen, i.e. if they are abstractions, then the model itself is an abstraction.

It is easy to verify that both operators $\circleddash$ and $\|$ are abstractions. Therefore the model presented Fig. 2 can be seen as an abstraction $(\mathbf{W3F})^7 \twoheadrightarrow (\mathbf{W3F})^{15}$, as it involves 7 state variables and 15 flow variables.

3.4 Minimal Cutsets

Let $\mathbf{M}$ be a finite degradation model built over $\mathbf{O}$ and $\mathbf{V} = \mathbf{S} \uplus \mathbf{F}$.

Conventionally, we call the flow variable on which the current analysis is focused on the *observer* of the analysis. Observers play the role of top-events in fault trees.

Let $w \in \mathbf{F}$ be the observer of the analysis. According to w, the model $\mathbf{M}$ can be interpreted as an abstraction $\phi|_w : \bigotimes_{v\in\mathbf{S}} \mathrm{dom}(v) \twoheadrightarrow \mathrm{dom}(w)$. Then, $\forall\, y \in \mathrm{dom}(w)$, we define the set of *cutsets* of w for y, denoted by $\mathbf{CS}(w, y)$, as follows.

$$\mathbf{CS}(w, y) \overset{def}{=} \{\overline{\mathbf{v}} | \overline{\mathbf{v}} \in \bigotimes_{v\in\mathbf{S}} \mathrm{dom}(v), \phi|_w(\overline{\mathbf{v}}) = y\} \tag{2}$$

A cutset $\mathbf{CS}(w, y)$ represents a combination of the states of components that leads the state of the observer w to be y. Therefore, the set of *minimal cutsets*, denoted by $\mathbf{MCS}(w, y)$, is defined as follows:

$$\mathbf{MCS}(w, y) \stackrel{def}{=} \{\overline{\mathbf{v}} \in \mathbf{CS}(w, y), \nexists \overline{\mathbf{u}} \in \mathbf{CS}(w, y), \overline{\mathbf{u}} \sqsubset \overline{\mathbf{v}}\} \tag{3}$$

The minimality of cutsets is captured by the degradation order defined in $\bigotimes_{v \in \mathbf{S}} \mathrm{dom}(v)$. In this sense, a minimal cutset of w and y represents one of the least degraded composition of components' states that degrades the state of the observer w from its least element $\perp$ to y. The extension of the concept of minimal cutsets from Boolean systems into multistate systems is one of the most important contributions of FDS.

3.5 Probabilistic Indicators

Let $\mathcal{S} : \langle D, \sqsubseteq, \perp \rangle$ be a FDS. We can equip $\mathcal{S}$ with a probability measure p, i.e. a function $p : D \rightarrow [0, 1]$ such that $\sum_{d \in D} p(d) = 1$.

The probability measure could also be a function of time, i.e. $p : D \times \mathbb{R}^+ \rightarrow [0, 1]$, where $p(d, t)$ represents the probability of being in the state $d \in D$ at time $t \in \mathbb{R}^+$. However, as it makes no difference in terms computationally speaking, we keep the above simplest definition.

Now, let $\mathcal{S} : \langle D_S, \sqsubseteq_S, \perp_S \rangle$ and $\mathcal{T} : \langle D_T, \sqsubseteq_T, \perp_T \rangle$ be two FDS equipped respectively with probability measures p_S and p_T.

Then, their product $\mathcal{S} \otimes \mathcal{T}$ can be equipped with the natural probability measure p defined as follows. $\forall \langle x, y \rangle \in D_S \times D_T$,

$$p\left(\langle x, y \rangle\right) \stackrel{def}{=} p_S(x) \times p_T(y)$$

It is easy to verify that p is actually a probability measure on $\mathcal{S} \otimes \mathcal{T}$. Its construction assumes indeed that the events represented by $\mathcal{S}$ and $\mathcal{T}$ are statistically independent.

Let $\mathcal{S} : \langle D_S, \sqsubseteq_S, \perp_S \rangle$ and $\mathcal{T} : \langle D_T, \sqsubseteq_T, \perp_T \rangle$ be two FDS. Assume that $\mathcal{S}$ is equipped with p_S and that $\mathcal{T}$ is an abstraction of $\mathcal{S}$. Then, the natural probability measure p_T is defined as follows. $\forall y \in D_T$

$$p_T(y) = \sum_{x \in \varphi^{-1}\{y\}} p_S(x)$$

The above two natural constructions make it possible to lift-up probabilistic indicators defined for fault trees to finite degradation models.

4 FDS-ML

FDS-ML stands for Finite Degradation Structures - Modeling Language. In its current version, which is purely textual, this small domain specific modeling language provides constructs to declare domains (finite degradation structures) and operators on the one hand, state and flow variables and equations on the other hand. We shall review these constructs in turn.

4.1 Domains and Operators

The syntax of FDS-ML is rather straightforward and is strongly inspired from the one of AltaRica 3.0 [1]. Therefore, we shall present it on example.

Figure 4 shows the FDS-ML code that declares the FDS **W3F** and the operators $\oslash$ and $\|$ involved in the model described Fig. 2.

```
domain W3F {W, Fs, Fdd, Fdu} (W<Fs, W<Fdd, Fdd<Fdu)

operator series(W3F, W3F) return W3F
    *, W -> *
    *, Fs -> Fs
    *, Fdd -> Fdd
    W, Fdu -> Fdu
    Fs, Fdu -> Fdd
    Fdd, Fdu -> Fdd
    Fdu, Fdu -> Fdu
end

operator parallel(W3F, W3F) return W3F
    W, * -> W
    Fs, * -> Fs
    *, Fs -> Fs
    *, W -> W
    Fdd, Fdd -> Fdd
    Fdd, Fdu -> Fdu
    Fdu, Fdd -> Fdu
    Fdu, Fdu -> Fdu
end
```

Fig. 4. Declarations of the FDS **W3F** and the operators $\oslash$ and $\|$.

The declaration **W3F** is self-explanatory.

The declaration of operators is just a bit more tricky. The first part consists in giving a name to the operator, and to declare the type of its arguments and its output. In the current version of FDS-ML, operators can return only one value.

The body of the declaration is a list of statements that are read in order. The first one that matches the values of the argument is taken. `*` matches any value.

Declarations of domains and operators can be reused from model to model. One of our objectives is to develop domain specific libraries of such declarations.

4.2 Variables and Equations

In FDS-ML, a model is declared as a `block`, i.e. a prototype in the sense of object-oriented theory.

```
block TA4
    W3F S1.state, S2.state, S3.state (W=..., Fs=..., Fdd=...)
    W3F S1.in, S1.out, S2.in, S2.out, S3.in, S3.out
    ...
    assertion
        S1.in := W
        S1.out := series(S1.in, S1.state)
        ...
        SIS := parallel(V1.out, V2.out)
    observer top = SIS
end
```

Fig. 5. Model of the SIS TA4 of ISO/TR 12489 written in FDS-ML.

This block is made of two parts: first variables are declared, then equations are given. Figure 5 sketches the model of the SIS.

The FDM for the SIS, as written in FDS-ML, is given in Fig. 5. The domains (i.e. FDSs) and the operators should be defined separately before use. The model is written in the part of `block`, where state variables should be declared and assigned with probabilities in parentheses `(W=..., Fs=..., Fdd=...)`. Formulas are written in the part of `assertion` and the `observer` should be declared right after.

5 Algorithms

The implementation of FDS-ML is programmed in Python. Only the main algorithms are presented in this section.

In the implementation, formulas are encoded by binary trees, which is the same as fault trees. The leaves of a formula tree are state variables. Each internal node $(\diamondsuit, f_l, f_r)$ encodes the formula $f_l \diamondsuit f_r$.

The *decision diagram* (DD) used in this article is a particular type of binary decision diagrams (BDD) that represent multi-valued functions in binary way [7,8]. Algorithms presented in this article is similar to those for BDD.

In the DD in this article, each internal node (s, v, n_d, n_r) is labelled with state s, variable v, down-child n_d and right-child n_r. The terminal node $(s, /, /, /)$ is only labelled with state s.

The DD is built for the top of a formula.

If the formula contains only a variable v without any operator, its DD is called a one-level DD. The one-level DD of v such that $\text{dom}(v) = \mathbf{W3F}$ is shown in Fig. 6. For every $s \in dom(v)$, we create an internal node (s, v, n_d, n_r) where n_d represents the resulting node if $v = s$. These internal nodes are connected successively by their right-child n_r in a chain. We fix the order of states in such chain for a given variable v.

The algorithms of building DD for formulas are given in Fig. 7. The input of `BuildDD` is the node of formula tree. The function n`.IsTermi()` returns true if n is a terminal node while n`.IsInter()` is true if n is a internal node.

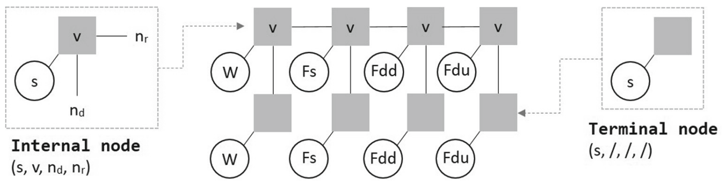

Fig. 6. The one-level DD of v such that $\text{dom}(v) = \mathbf{W3F}$.

Note that the `Combine` algorithms, as well as the `Prob` algorithms in Fig. 9, use caching [3]. Caching makes it possible to not redo an operation that has been already done.

```
BuildDD($v$)                  ← One-Level-DD $v$                   if $v$ is a state variable
BuildDD(($\diamondsuit, f_1, f_2$)) ← Operate($\diamondsuit, n_1, n_2$)
                                $n_1$ ← BuildDD($f_1$)
                                $n_2$ ← BuildDD($f_2$)

Operate($\diamondsuit, n_1, n_2$)    ← Combine($\diamondsuit, n_1, n_2$)          if $n_1$.IsInter() and $n_2$.IsInter()
                              ← Value($\diamondsuit, n_1, n_2$)            if $n_1$.IsTermi() and $n_2$.IsTermi()
                              ← ($n_1.s, n_1.v, n_d, n_r$)         if $n_1$.IsInter() and $n_2$.IsTermi()
                                $n_d$ ← Operate($\diamondsuit, n_1.n_d, n_2$)
                                $n_r$ ← Operate($\diamondsuit, n_1.n_r, n_2$)
                              ← ($n_2.s, n_2.v, n_d, n_r$)         if $n_1$.IsTermi() and $n_2$.IsInter()
                                $n_d$ ← Operate($\diamondsuit, n_1, n_2.n_d$)
                                $n_r$ ← Operate($\diamondsuit, n_1, n_2.n_r$)

Combine($\diamondsuit, n_1, n_2$)    ← Combine($\diamondsuit, n_2, n_1$)          if $v_2 \prec v_1$
                              ← ($n_1.s, n_1.v, n_d, n_r$)         otherwise
                                $n_r$ ← Operate($\diamondsuit, n_1.n_r, n_2$)
                                $n_d$ ← Operate($\diamondsuit, n_1.n_d, n'$)
                                    $n'$ ← $n_2$       if $v_1 \prec v_2$
                                       ← $n_2.n_d$ if $v_1 = v_2$

Value($\diamondsuit, n_1, n_2$)      ← ($s$, /, /, /)
                                $s$ ← $\diamondsuit(n_1.s, n_2.s)$               The value assignment by $\diamondsuit$
```

Fig. 7. Recursive algorithm of building DD for formulas.

The symbol $\prec$ in the algorithm represents the variable ordering of DD. It is worth noting that if all the operators used in the model are commutative, then the variable ordering is arbitrary. Otherwise, for instance $u \circleddash v$, the local ordering of u, v should be $u \prec v$. Note that only the state variables in the model need to be ordered.

Figure 8 shows the DD built for $u \otimes v$, where $\mathrm{dom}(u) = \mathrm{dom}(v) = \mathbf{W3F}$.

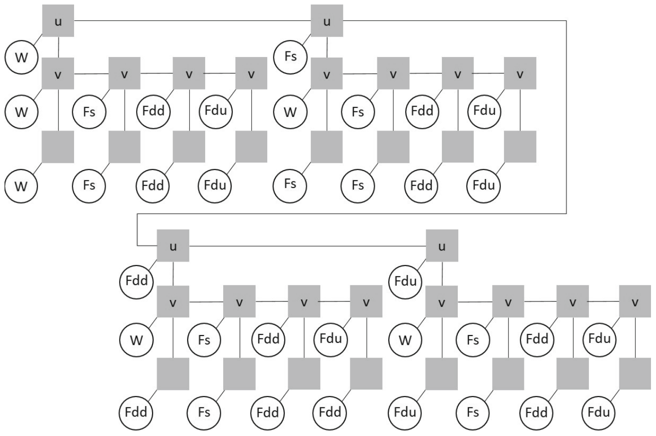

Fig. 8. DD of the formula $u \otimes v$, where $\mathrm{dom}(u), \mathrm{dom}(v) = \mathbf{W3F}$.

Once the DD is built, we can calculate the probabilistic indicators for the flow variable defined by the formula associated to this DD. The algorithms are given in Fig. 9. For any internal node $n = (s, v, n_d, n_r)$, $p_n = p(s) \in [0, 1]$ where $s \in \mathrm{dom}(v)$ and p is the probability measure defined in $\mathrm{dom}(v)$.

$$\texttt{Prob}(n, y) \leftarrow \begin{cases} 0 & \text{if } n.\texttt{IsTermi()} \text{ and } n.s \neq y \\ 1 & \text{if } n.\texttt{IsTermi()} \text{ and } n.s = y \\ p_n \times \texttt{Prob}(n.n_d, y) + \texttt{Prob}(n.n_r, y) & \text{if } n.\texttt{IsInter()} \end{cases}$$

Fig. 9. Algorithms of calculating probabilities from DD.

6 Experiments

In this section, we provide the assessment results of the model of SIS presented in Sect. 2.

The flow variable SIS of the model in Fig. 2 is selected as the observer of the analysis.

The variable ordering in this case is not arbitrary as $\otimes$ is not commutative. According to the model in Fig. 2, we select a valid variable ordering: $S1.state \prec LS1.state \prec V1.state \prec S2.state \prec S3.state \prec LS2.state \prec V2.state$.

As inputs, the state probabilities of each type of the components are calculated according to the failure rates given in Table 1. For those with NA (not applicable), the probability is set to be zero.

The calculation results of the number of cutsets $|\mathbf{CS}(SIS, y)|$ and the number of minimal cutsets $|\mathbf{MCS}(SIS, y)|$ for each state $y \in dom(SIS)$ are given in Table 3.

As illustration, the seven minimal cutsets in $\mathbf{MCS}(SIS, Fdd)$ are listed in Table 4, which are the least degraded scenarios that SIS is degraded from W to Fdd.

Table 3. The number of cutsets $|\mathbf{CS}(SIS, y)|$ and minimal cutsets $|\mathbf{MCS}(SIS, y)|$ for each state $y \in dom(SIS)$.

y	W	Fs	Fdd	Fdu
$\|\mathbf{CS}(SIS, y)\|$	433	9623	4169	2159
$\|\mathbf{MCS}(SIS, y)\|$	1	7	7	17

Table 4. The minimal cutsets in $\mathbf{MCS}(SIS, Fdd)$.

$S1.state$	$LS1.state$	$V1.state$	$S2.state$	$S3.state$	$LS2.state$	$V2.state$
Fdd	W	W	W	W	Fdd	W
W	W	Fdd	W	W	W	Fdd
Fdd	W	W	Fdd	Fdd	W	W
W	Fdd	W	W	W	Fdd	W
W	Fdd	W	Fdd	Fdd	W	W
W	W	Fdd	W	W	Fdd	W
W	W	Fdd	Fdd	Fdd	W	W

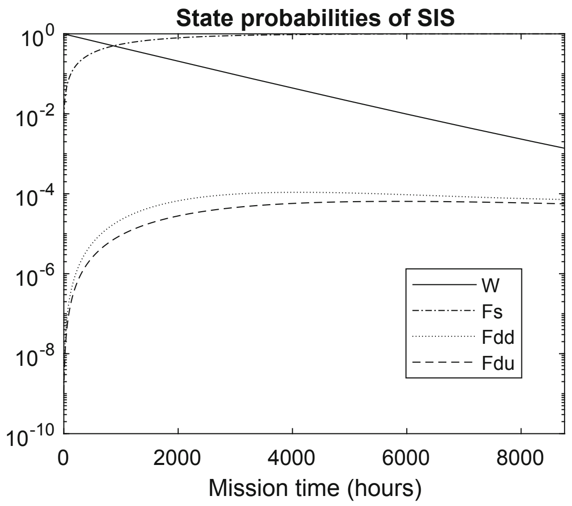

Fig. 10. The results of the probability of each state in $dom(SIS)$.

For probabilistic indicators, the results of the state probabilities in $dom(SIS)$ are pictured Fig. 10. Numerically, the average probabilities P_{avg} within the mission time (8760 h for each state are: $P_{avg}(W) = 1.449 \times 10^{-1}$, $P_{avg}(Fs) = 8.550 \times 10^{-1}$, $P_{avg}(Fdd) = 7.6916 \times 10^{-5}$ and $P_{avg}(Fdu) = 4.6073 \times 10^{-5}$.

7 Conclusion

In this article, we introduced a new modeling formalism, so-called FDS-ML, dedicated to the design of combinatorial probabilistic risk assessment models. We presented decision diagram based algorithms to assess FDS-ML models and we showed by means of a use case stemmed from ISO/TR 12489 standard the interest of the proposed approach.

FDS-ML relies on the notion of finite degradation structures. Finite degradation structures can be seen as the most general mathematical framework to design combinatorial probabilistic risk assessment models. As of today, the language is rather simple: it just provides constructs to define domains (finite degradation structures) and operators as well as to declare variables and equations. This is necessary and sufficient for basic uses, but our ambition is to make FDS-ML a full object-oriented language, using the S2ML+X paradigm [2,10]. Here X would stand for the current FDS-ML. Object-orientation, in the sense of S2ML, is a key enabler for the design of reusable modeling patterns, as demonstrated with AltaRica 3.0 [1]. The design of such patterns for finite degradation models is of primary importance for their industrial deployment as it makes it possible to hide, to some extent, the mathematical difficulties: with suitable, domain-specific libraries of modeling patterns, analysts can design their models by copying existing ones and adjusting them to their particular needs.

Regarding the implementation, much remains also to do. So far, our prototype is implemented in Python, which is indeed not ideal in terms of efficiency. We plan to move to C++ as soon as concepts and methods will be sufficiently stable. Decision diagram algorithms are now relatively mature. We plan to implement also bottom-up algorithms generalizing those designed for fault tree assessment [9].

References

1. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica 3.0 in 10 modeling patterns. Int. J. Crit. Comput.-Based Syst. **9**(1–2), 133–165 (2018). https://doi.org/10.1504/IJCCBS.2019.098809
2. Batteux, M., Prosvirnova, T., Rauzy, A.: From models of structures to structures of models. In: IEEE International Symposium on Systems Engineering, ISSE 2018. IEEE, Roma, October 2018. https://doi.org/10.1109/SysEng.2018.8544424. Best paper award
3. Brace, K.S., Rudell, R.L., Bryant, R.S.: Efficient implementation of a BDD package. In: Proceedings of the 27th ACM/IEEE Design Automation Conference, pp. 40–45. IEEE, Orlando (1990). https://doi.org/10.1145/123186.123222

4. Guo, H., Yang, X.: A simple reliability block diagram method for safety integrity verification. Reliab. Eng. Syst. Saf. **92**(9), 1267–1273 (2007). https://doi.org/10.1016/j.ress.2006.08.002
5. International IEC standard IEC61508 - functional safety of electrical/electronic/programmable safety-related systems (E/E/PE, or E/E/PES). Standard, International Electrotechnical Commission, Geneva, Switzerland, April 2010
6. ISO/TR 12489:2013 petroleum, petrochemical and natural gas industries - reliability modelling and calculation of safety systems. Standard, International Organization for Standardization, Geneva, Switzerland, November 2013
7. Minato, S.I.: Zero-suppressed BDDs for set manipulation in combinatorial problems. In: Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC 1993, pp. 272–277. IEEE, Dallas (1993). https://doi.org/10.1145/157485.164890
8. Minato, S.I.: Binary Decision Diagrams and Applications for VLSI CAD. Kluwer Academic Publishers, Dordrecht (1996)
9. Rauzy, A.: Anatomy of an efficient fault tree assessment engine. In: Virolainen, R. (ed.) Proceedings of International Joint Conference PSAM 2011/ESREL 2012, June 2012
10. Rauzy, A., Haskins, C.: Foundations for model-based systems engineering and model-based safety assessment. J. Syst. Eng. (2018). https://doi.org/10.1002/sys.21469
11. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput. Sci. Rev. **15**, 29–62 (2015). https://doi.org/10.1016/j.cosrev.2015.03.001
12. Ushakov, I.: Probabilistic Reliability Models. Wiley, Hoboken (2012)
13. Yang, L., Haskins, C., Rauzy, A.: Finite degradation structures: a formal framework to support the interface between MBSE and MBSA. In: IEEE International Symposium on Systems Engineering, ISSE 2018. IEEE, Roma, October 2018. https://doi.org/10.1109/SysEng.2018.8544411
14. Yang, L., Rauzy, A.: Reliability modeling using finite degradation structures. In: Proceedings of the 3rd International Conference on System Reliability and Safety (ICSRS), pp. 168–175. IEEE, Barcelona, November 2018. https://doi.org/10.1109/ICSRS.2018.00035