information

MDPI

Article

Decision Diagram Algorithms to Extract Minimal Cutsets of Finite Degradation Models

Antoine Rauzy * and Liu Yang

Department of Mechanical and Industrial Engineering (MTP), Norwegian University of Science and Technology, NO-7491 Trondheim, Norway; liu.yang@ntnu.no

* Correspondence: Antoine.Rauzy@ntnu.no

Received: 19 August 2019; Accepted: 20 November 2019; Published: 25 November 2019

Abstract: In this article, we propose decision diagram algorithms to extract minimal cutsets of finite degradation models. Finite degradation models generalize and unify combinatorial models used to support probabilistic risk, reliability and safety analyses (fault trees, attack trees, reliability block diagrams...). They formalize a key idea underlying all risk assessment methods: states of the models represent levels of degradation of the system under study. Although these states cannot be totally ordered, they have a rich algebraic structure that can be exploited to extract minimal cutsets of models, which represent the most relevant scenarios of failure. The notion of minimal cutsets we introduce here generalizes the one defined for fault trees. We show how algorithms used to calculate minimal cutsets can be lifted up to finite degradation models, thanks to a generic decomposition theorem and an extension of the binary decision diagrams technology. We discuss the implementation and performance issues. Finally, we illustrate the interest of the proposed technology by means of the use case stemmed from the oil and gas industry.

Keywords: combinatorial reliability models; minimal cutsets; finite degradation structures

1. Introduction

Since the famous WASH report [1] that followed the Three Mile Island nuclear accident, probabilistic risk, reliability and safety analyses have been developed considerably. They are nowadays applied to virtually all industrial systems the operation of which presents a risk for the systems themselves, their operators or their environment.

Probabilistic risk, reliability and safety assessment models describe how a system may degrade and eventually fail under the occurrence of random events such as failures of mechanical components, sudden changes in environmental conditions or human errors. They can be split into three nested categories: combinatorial models, state automata and process algebras [2].

Combinatorial models are the simplest, but also by far the most widely used. In these models, the state of the system is described as a combination of the states of its components. As the number of possible states considered for each component is finite, and in general small, the number of possible states of the system is also finite, although subject to an exponential blow-up. Boolean models, i.e., fault trees, reliability block diagrams and event trees, belong to this category, see e.g., [3,4] for reference textbooks. So-called multistate systems [5–7], i.e., extensions of Boolean models to the case where components can be in more than two—but still a finite number of—states, enter also in this category.

Intuitively, combinatorial models take a snapshot of the system at a given point in time. They do not consider the trajectory that lead the system from its initial state to its current state. Consequently, they can only approximate time dependencies induced for instance by cold-redundancies or reconfigurations. To take into account such dependencies, one needs more expressive models such as Markov chains [8], stochastic Petri nets [9], or guarded transition systems [10,11]. These models

enter either in the category of state automata or in the category of process algebras. However, the more expressive the model, the more computationally expensive the calculation of probabilistic indicators from that model. Consequently, probabilistic risk, reliability and safety assessment models result always of a tradeoff between the accuracy of the description of the system under study and the ability to perform calculations on this description, see reference [2] for an in-depth discussion. Moreover, the more expressive the model, the harder it is to design, to validate and to maintain. This is the reason why combinatorial models, and more specifically Boolean models, dominate the industrial practice.

It remains that, with the steadily increasing complexity of technical systems, the more expressive power we can have for the models, the better, with the proviso that the added expressive power does not degrade the computational complexity of assessments. This is why lots of efforts have been made in the recent years on the study of multistate systems. This is also what led us to introduce the algebraic concept of finite degradation structures [12]. Finite degradation models generalize and unify both Boolean models and multistate systems. They can be seen as the most general mathematical framework that can be used to design probabilistic risk assessment models, while staying in the realm of combinatorial models.

Calculations performed on Boolean models belong formally to two distinct categories: qualitative assessments and quantitative assessments. The former aim primarily at extracting and classifying minimal cutsets. Minimal cutsets represent minimal scenarios of failure. The latter aim primarily at calculating probabilistic risk indicators such as the unavailability of the system (probability that the system is failed at time t), safety integrity levels or importance measures. In practice, this distinction is blurred because there are so many minimal cutsets that only the most relevant, i.e., the most probable, ones can be extracted. Moreover, the best way of calculating probabilistic indicators is to assess them via minimal cutsets. This is the reason why the extraction of the most probable minimal cutsets plays a central role in probabilistic risk and safety analyses.

Finite degradation structures formalize what is probably the most fundamental idea underlying all risk assessment methods: the states of a model represent essentially various levels of degradation of the system under study. Although these states cannot be totally ordered, they have rich algebraic structure that can be exploited both by analysts and by assessment algorithms: when studying a given level of degradation of the system, one is interested in the minimal conditions that produce this degradation level. These conditions are in turn characterized by the minimal states, according to the degradation order, in which the system is in the given degradation level. In other words, the notion of minimal cutset, or minimal state, of finite degradation models generalizes and sheds a new light on the one defined for fault trees.

There are essentially two families of algorithms to extract minimal cutsets. The first one consists of top-down algorithms. MOCUS [13] and its modern versions such as the one implemented in RiskSpectrum [14] and the one implemented in XFTA [15], belong to this family. The second family consists of bottom-up algorithms based on the binary decision diagrams technology [16–18]. We show here how to extend this technology so to handle finite degradation structures, via ideas introduced by Minato [19] (in a different context). For this purpose, we introduce the notion of implicative normal form for formulas encoding discrete functions. We demonstrate a generic decomposition theorem making it possible the extraction of the most probable minimal states. This theorem generalizes the one of reference [16], which is at core of minimal cutsets calculations.

The contribution of this article is thus fourfold. First, it shows how to generalize the concept of minimal cutsets to finite degradation models. Second, it demonstrates a decomposition theorem that makes possible to lift-up Boolean algorithms proposed in reference [16] to finite degradation models. Third, it shows how to extend the binary decision diagram technology so to handle finite degradation models. Last, it discusses approximation schemes in the context of finite degradation models.

The remainder of this article is organized as follows. Section 2 presents an illustrative example stemmed from the oil and gas industry. Section 3 introduces finite degradation structures and finite degradation models. Section 4 shows how to lift-up the notion of minimal cutset. Section 5 shows

how to extend the binary decision diagrams technology. Section 6 discusses approximation schemes. Section 7 provides some experimental results. Finally, Section 8 concludes the article.

Appendix A presents the full model of the use case.

2. Illustrative Example

This section presents an example we shall use throughout the article to illustrate the proposed approach.

2.1. Description

The high pressure protection system pictured in Figure 1 is referred to as TA4 in ISO/TR 12489 standard [20].

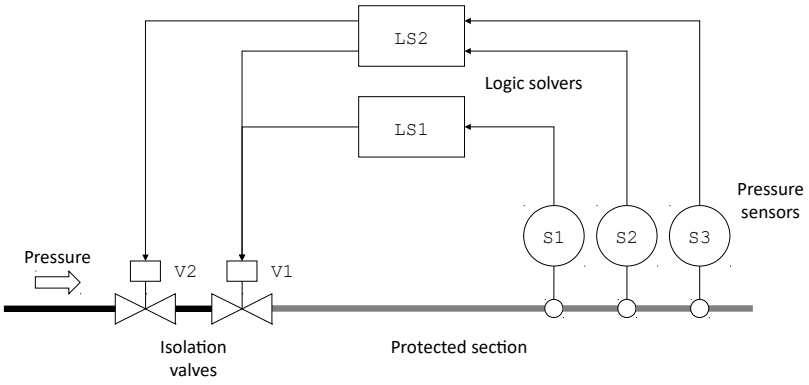

Figure 1. A high pressure protection system taken from ISO/TR 12489 (system TA4).

The objective of this safety instrumented system 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`). When the sensors detect an overpressure in the protected section, the logic solvers order the isolation valves to close 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.

As the failure of the TA4 HIPPS may result in a catastrophic accident, it is of importance to assess its reliability. This is classically done by means of a fault tree such as the one pictured in Figure 2.

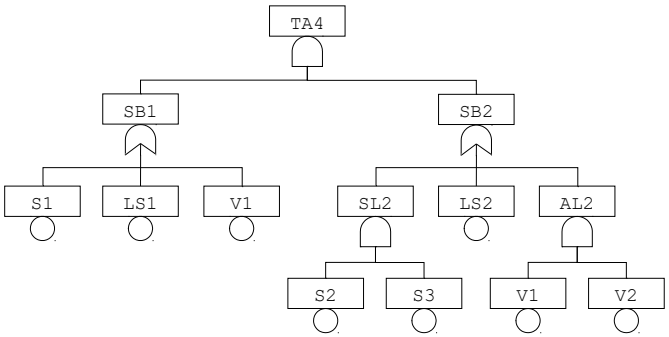

Figure 2. A fault tree describing the failures of the TA4 HIPPS.

2.2. Classification of Failures

The fault tree pictured in Figure 2 does not distinguish however the different types of failures. According to the standard IEC61508 [21], individual failures of components and global failures of a safety instrumented system can be classified along two directions: safe versus dangerous failures, and

detected versus undetected failures. One finds similar classifications in other safety standards such as ISO 26262 [22].

In our example, safe failures are those which contribute to close the isolation valves, even though there is no overpressure (spurious triggers), while dangerous failures are those which contribute to keep the isolation valves open, even though there is an overpressure. These two types of failures have indeed very different consequences: the former hampers the profitability of the production while the latter may result in catastrophic accidents. Moreover, their frequencies may be very different.

If we look more specifically at dangerous failures, we need to distinguish detected and undetected ones. 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.

- All other components may fail (independently). Their probabilities of failure follow negative exponential distributions. The parameters of these distributions are given in Table 1. Safe failures are always detected.
- The system is maintained once a year (once in 8760 hours). 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
Safe failure rate	$3.00 \times 10^{-6}\ \mathrm{h}^{-1}$	$3.00 \times 10^{-7}\ \mathrm{h}^{-1}$	NA
Dangerous detected failure rate	$3.00 \times 10^{-7}\ \mathrm{h}^{-1}$	$6.00 \times 10^{-7}\ \mathrm{h}^{-1}$	$2.90 \times 10^{-6}\ \mathrm{h}^{-1}$
Dangerous undetected failure rate	$3.00 \times 10^{-7}\ \mathrm{h}^{-1}$	NA	$2.90 \times 10^{-7}\ \mathrm{h}^{-1}$

2.3. Discussion

The Boolean logic at work in fault trees and related formalisms does not make it possible to distinguish the different types of failures. More exactly, it would be possible to design a fault tree for each type of failure, but this would be both tedious and error prone. Hence the idea to use multi-valued logics, in which variables and formulas can take more than two, but still a finite number of values [7]. This raises however two conceptual and practical issues:

- Which algebraic properties such a logic should they obey so that usual notions of reliability engineering (minimal cutsets, top-event probability, importance measures, safety integrity levels...) can be lifted-up into the logic?
- Once the mathematical foundations have been established, how to perform efficiently calculations of risk indicators?

Finite degradation structures provide a general mathematical framework to answer the first question. In this article, we provide a concrete answer to the second one.

3. Finite Degradation Models

This section introduces finite degradation models which unify and generalize the various types of combinatorial models used in the framework of probabilistic risk, reliability and safety analyses.

3.1. Finite Degradation Structures

A finite degradation structure is a triple $\langle C, \leq_C, \bot_C \rangle$ where:

- C is a finite set of constants.
- $\leq_C$ is a partial order over C.

- $\bot_C$ is the unique lower bound of C, i.e., the unique element such that $\bot_C \leq_C x$ for all $x \in C$.

Technically, finite degradation structures are thus semi-lattices. C is called the support set of the finite degradation structure.

The idea behind finite degradation structures is that each element of C represents a possible state of the system under study. The partial order $\leq_C$ sorts states according to their level of degradation, i.e., $s \leq_C t$ if the system is less degraded in the state s than in the state t. Finally, $\bot_C$ represents the initial, nominal state of the system.

Figure 3 shows Hasse diagrams representing some typical finite degradation structures. In these diagrams, the degradation order is represented bottom-up.

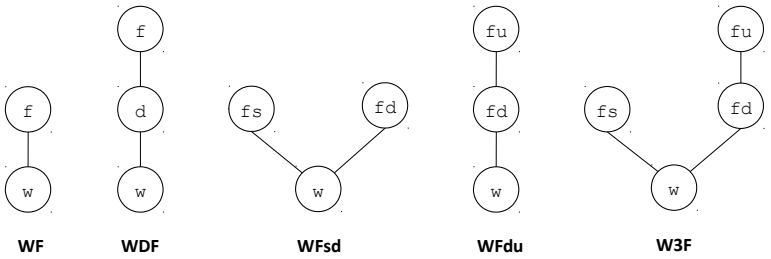

Figure 3. Hasse diagrams for some typical finite degradation structures.

The finite degradation structure **WF** is the classical Boolean domain: `w` and `f` stand respectively for working and failed. Indeed, `f` is more degraded than `w`. The finite degradation structure **WDF** introduces the intermediate value `d`, which stands for degraded. Hence the natural degradation order `w` $<$ `d` $<$ `f`. In the finite degradation structure **WFsd**, the failure state `fs` (failed safe) is not comparable with the failure state `fd` (failed dangerous) although both are more degraded than the working state `w`. The finite degradation structure **WFdu** distinguishes two types of dangerous failures: detected ones (`fd`) and undetected ones (`fu`). Of course, undetected failures are more problematic than detected ones. Hence the order `w` $<$ `fd` $<$ `fu`. Last but not least, the finite degradation structure **W3F** combines **WFsd** and **WFsd**. We shall come back on this finite degradation structure that is, implicitly, at the core of safety standards such as IEC61508 [21] and ISO/TR 12489 [20].

Note that **WDF** and **WFdu** are isomorphic. However, operations on these two finite degradation structures are different. It is thus worth to consider them as distinct.

3.2. Product

Now, let $\mathcal{C} : \langle C, \leq_C, \bot_C \rangle$ and $\mathcal{D} : \langle D, \leq_D, \bot_D \rangle$ be two finite degradation structures, the product of $\mathcal{C}$ and $\mathcal{D}$, denoted by $\mathcal{C} \otimes \mathcal{D}$, is the triple $\langle C \times D, \leq_{C\times D}, \bot_{C\times D} \rangle$, where:

- $C \times D$ stands for the Cartesian product of C and D.
- For all $x, x' \in C$ and $y, y' \in D$, $\langle x, y \rangle \leq_{C\times D} \langle x', y' \rangle$ if and only if $x \leq_C x'$ and $y \leq_D y'$.
- $\bot_{C\times D} = \langle \bot_C, \bot_D \rangle$.

It is easy to verify that $\mathcal{C} \otimes \mathcal{D}$ is a finite degradation structure. This is by no means surprising as the category of partial orders has a product [23].

The product $\otimes$ is commutative and associative up to an isomorphism, i.e., $\mathcal{C} \otimes \mathcal{D} \cong \mathcal{D} \otimes \mathcal{C}$ and $\mathcal{C} \otimes (\mathcal{D} \otimes \mathcal{E}) \cong (\mathcal{C} \otimes \mathcal{D}) \otimes \mathcal{E}$ for all finite degradation structures $\mathcal{C}$, $\mathcal{D}$ and $\mathcal{E}$. The finite degradation structure $\mathbf{1} = \langle \{\bot\}, \bot \leq \bot, \bot \rangle$ is a neutral element for this product: $\mathbf{1} \otimes \mathcal{C} \cong \mathcal{C} \otimes \mathbf{1} \cong \mathcal{C}$ for any finite degradation structure $\mathcal{C}$.

The existence of a product over finite degradation structures makes it possible to describe the state of system as the combination of the states of its components.

Consider, for instance, a system made of two components `A` and `B`. Assume that both `A` and `B` are represented by means of the finite degradation structure **WDF**. Then, the state space of the system is represented by the finite degradation structure $\mathbf{WDF}^2$ pictured in Figure 4.

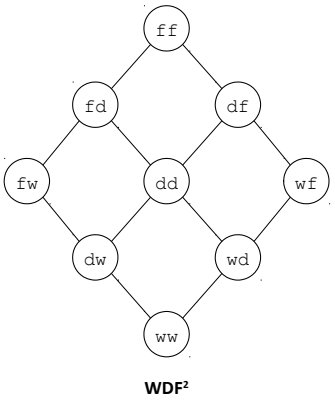

Figure 4. Hasse diagram for the finite degradation structure $\mathbf{WDF}^2 = \mathbf{WDF} \otimes \mathbf{WDF}$.

Note that although **WDF** encodes a total order over its elements, $\mathbf{WDF}^2$ encodes only a partial order.

3.3. Formulas

In the previous example, the state of the system is represented by a pair. Now if, for instance, the components A and B are in series, we can map each pair onto an element of **WDF**, e.g. $(\mathtt{w}, \mathtt{w})$ is mapped onto $\mathtt{w}$, $(\mathtt{w}, \mathtt{d})$ is mapped onto $\mathtt{d}$, $(\mathtt{w}, \mathtt{f})$ is mapped onto $\mathtt{f}$, and so on. There are indeed several ways to define such a mapping. Moreover the target finite degradation structure does not need to be the same as the source ones, which themselves may differ one another. In other words, it is possible to define a collection of operators on finite degradation structures.

In the sequel, we shall thus assume a finite set $\mathcal{U}$ of finite degradation structures and a finite set $\mathcal{O}$ of symbols called operators.

Each operator o of $\mathcal{O}$ is associated with a mapping $[\![o]\!]$ from $\bigotimes_{1 \leq i \leq n} s_i$, $n \geq 0$, into s, where both the s_i's and s are finite degradation structures. n is the arity of o. $\bigotimes_{1 \leq i \leq n} s_i$ is called the domain of o and is denoted $dom(o)$. s is called the codomain of o and is denoted $codom(o)$. The signature of o, which characterizes its arguments and return types, is denoted $\bigotimes_{1 \leq i \leq n} s_i \to s$.

It is often, but not always, the case that operators are abstractions. Abstractions are surjective structure preserving mappings, i.e., an abstraction $\alpha : \bigotimes_{1 \leq i \leq n} s_i \to s$ obeys the following conditions:

- For all $\bar{u}, \bar{v} \in \bigotimes_{1 \leq i \leq n} s_i$, $\bar{u} \leq \bar{v} \Rightarrow \alpha(\bar{u}) \leq \alpha(\bar{v})$.
- $\alpha(\bigotimes_{1 \leq i \leq n} \bot_{s_i}) = \bot_s$.
- $\forall v \in s, \exists \bar{u} \in \bigotimes_{1 \leq i \leq n} s_i$ such that $\alpha(\bar{u}) = v$.

As abstractions are epimorphisms in the sense of category theory, we shall use in the sequel the usual symbol $\twoheadrightarrow$ for their signature, i.e., $\bigotimes_{1 \leq i \leq n} s_i \twoheadrightarrow s$.

It is easy to check that the notion of abstraction generalizes the notion of monotone Boolean function.

As an illustration, one can consider the two binary operators $\circledcirc$ and $\|$ from $\mathbf{W3F} \otimes \mathbf{W3F}$ into **W3F**. The operator $\circledcirc$ represents the series composition. It generalizes the Boolean operator $\vee$ (or). The operator $\|$ represents the parallel composition. It generalizes the Boolean operator $\wedge$ (and). These operators are defined in Table 2.

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

We can now define formulas of the finite degradation calculus.

Let $\mathcal{U}$ be a finite set of finite degradation structures, let $\mathcal{O}$ be a finite set of operators on $\mathcal{U}$ defined as above, and finally, let $\mathcal{V}$ be a finite set of symbols called variables. Each variable V of $\mathcal{V}$ is assumed to take its value in the support set of one of the finite degradation structures of $\mathcal{U}$. This finite degradation structure is called the domain of V and is denoted $dom(V)$.

Table 2. Definition of operators $\circledcirc$ and $\|$.

$U \circledcirc V$		V			
		w	**fs**	**fd**	**fu**
U	**w**	w	fs	fd	fu
	fs	fs	fs	fd	fd
	fd	fd	fs	fd	fd
	fu	fu	fs	fd	fu

$U \| V$		V			
		w	**fs**	**fd**	**fu**
U	**w**	w	fs	w	w
	fs	fs	fs	fs	fs
	fd	w	fs	fd	fu
	fu	w	fs	fu	fu

The set of well formed (typed) formulas over $\mathcal{U}$, $\mathcal{V}$ and $\mathcal{O}$ is the smallest set such that:

- Constants, i.e., members of support sets of finite degradation structures of $\mathcal{U}$, are well formed formulas. The type of a constant is the finite degradation structure it comes from.
- Variables of $\mathcal{V}$ are well-formed formulas. The type of a variable V is simply its domain.
- If o is an operator of $\mathcal{O}$ such that $[\![o]\!] : \bigotimes_{1 \leq i \leq n} s_i \to s$, and $f_1, \ldots f_n$ are well formed formulas of types $s_1, \ldots s_n$, then $o(f_1, \ldots, f_n)$ is a well formed formula of type s.

In the sequel, we shall say simply formula instead of a well-formed formula.

The set of variables occurring in the formula f is denoted $var(f)$.

A variable valuation of f is a mapping from $var(f)$ into $\prod_{V \in var(f)} dom(V)$, i.e., a function that associates with each variable a value of its domain.

f is interpreted as a mapping $[\![f]\!] : \bigotimes_{V \in var(f)} dom(V) \to s$ where s is the codomain of the outmost operator of f, by lifting up as usual variable valuations. Let σ be a variable valuation of $var(f)$, then:

- If f is reduced to a constant c, then $[\![f]\!](\sigma) = c$.
- If f is reduced to a variable V, then $[\![f]\!](\sigma) = \sigma(V)$.
- If f is in the form $o(f_1, \ldots, f_n)$, then $[\![f]\!](\sigma) = [\![o]\!]([\![f_1]\!](\sigma), \ldots, [\![f_n]\!](\sigma))$.

3.4. Finite Degradation Models

Finite degradation models are obtained by lifting up fault tree constructions to the finite degradation calculus. Namely, a finite degradation model is a pair $\langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ where:

- $\mathcal{S} = \{V_1, \ldots, V_m\}$, $m \geq 1$, is a finite set of state variables.
- $\mathcal{F} = \{W_1, \ldots, W_n\}$, $n \geq 1$, is a finite set of flow variables.
- $\mathcal{E} = \{W_1 := f_1, \ldots, W_n := f_n\}$ is a finite set of equations such that for $1 \leq j \leq n$:
  - W_j is the jth variable of $\mathcal{F}$.
  - f_j is a formula built over $\mathcal{V}$.
  - $codom(f_j) = dom(W_j)$

We say that the flow variable W_j depends on the (state or flow) variable V if either $V \in var(f_j)$ or there exists a variable $U \in var(f_j)$ that depends on V.

A finite degradation model is looped if one of its flow variable depends on itself. It is loop-free otherwise.

A finite degradation model is uniquely rooted if there is only one variable that occurs in no right member of an equation. Such a variable is a called the root of the model.

From now, we shall consider only well-typed, loop-free, uniquely rooted models.

It is easy to see that finite degradation models generalize fault trees: state and flow variables play respectively the roles of basic and internal events, while equations play the role of gates. Moreover, the root variable plays the role of the top event. The terms "state" and "flow" comes from guarded transition systems [10].

As an illustration, consider TA4 HIPPS presented Section 2. We can use the finite degradation structure `W3F` and the operators defined in Table 2 to represent this system.

The fault tree pictured in Figure 2 can be re-interpreted in this logic. Equations implementing this fault tree are given in Table 3.

Table 3. Equations implementing the fault tree given in Table 2 in the **W3F** logic.

`TA4`	:=	`SB1`$\|$`SB2`
`SB1`	:=	`S1`⊘`LS1`⊘`V1`
`SB2`	:=	`SL2`⊘`LS2`⊘`AL2`
`SL2`	:=	`S2`$\|$`S3`
`AL2`	:=	`V1`$\|$`V2`

In this model, `S1`, `S2`, `S3`, `LS1`, `LS2`, `V1` and `V2` are state variables, while `TA4`, `SB1`, `SB2`, `SL2` and `AL2` are flow variables.

Note that `LS1` and `LS2` cannot take the value `fu` as we assumed that failures of logic solvers always detected. Similarly, `V1` and `V2` cannot take the value `fs` as we assumed that spurious closures of valves, if any, are immediately fixed.

3.5. Semantics

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be a finite degradation model. A variable valuation of $\mathcal{V}$ is a function $\sigma : \mathcal{V} \to dom(\mathcal{V})$, i.e., a function that associates with each variable V of $\mathcal{V}$ a value in $dom(V)$.

A variable valuation σ is admissible in the model $\mathcal{M} : \langle \mathcal{V}, \mathcal{E} \rangle$ if $\sigma(W_j) = \sigma(f_j)$ for each equation $W_j := f_j$ of $\mathcal{E}$.

The following property holds, thanks to the fact that the model is loop-free.

Lemma 1 (Unicity of admissible variable valuations)**.** *Let $\mathcal{M}$ be a finite degradation model and σ be a partial variable valuation that assigns values only to state variables of $\mathcal{M}$. Then there is a unique way to extend σ into an admissible total assigment σ' of variables of $\mathcal{M}$.*

Note, σ' is simply calculated bottom-up by propagating values in equations.

In the sequel, we shall denote $\overrightarrow{\sigma}_{\mathcal{M}}$, or simply $\overrightarrow{\sigma}$ when the model $\mathcal{M}$ is clear from the context, this unique extension.

Consider for instance the model given in Table 3 and the valuation of state variables σ : `S1` = `w`, `S2` = `fu`, `S3` = `fd`, `LS1` = `w`, `LS2` = `w`, `V1` = `fu` and `V2` = `w`. Then, $\overrightarrow{\sigma}$ is the total variable valuation such that `SL2` = `fu`, `AL2` = `w`, `SB1` = `fu`, `SB2` = `fu` and finally `TA4` = `fu`.

There is indeed a one-to-one correspondence between variable valuations and elements of $\bigotimes_{V \in \mathcal{V}} dom(V)$. We shall thus make no distinction between them in the sequel.

It follows from the above lemma, that we can interpret a model $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ as a mapping:

$$[\![\mathcal{M}]\!] : \bigotimes_{V \in \mathcal{S}} dom(V) \to \bigotimes_{W \in \mathcal{F}} dom(W)$$

Note that flow variables include in particular the root variable R of the model. Therefore, if we restrict $[\![\mathcal{M}]\!]$ to its R component, we can interpret it as a mapping:

$$[\![\mathcal{M}]\!]|_R : \bigotimes_{V \in \mathcal{S}} dom(V) \to dom(R)$$

This is indeed true for any other flow variable.

The following result holds.

Lemma 2 (Coherent finite degradation models)**.** *Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be a uniquely rooted, loop-free finite degradation model, whose root variable is $R \in \mathcal{F}$. If all operators used to build equations of $\mathcal{E}$ are abstractions, then $\mathcal{M}$ defines an abstraction:*

$$[\![\mathcal{M}]\!]|_R : \bigotimes_{V \in \mathcal{S}} dom(V) \twoheadrightarrow dom(R)$$

Lemma 2 generalizes the well-known result that a sufficient (but not necessary) condition for a fault tree to be coherent is that it is written using only $\wedge$, $\vee$ and k-out-of-n connectives.

The finite degradation model given in Table 3 is thus a mapping: $\mathbf{W3F}^3 \otimes \mathbf{WFsd}^2 \otimes \mathbf{WFdu}^2 \rightarrow \mathbf{W3F}$ as the states of the three sensors are described with the finite degradation structure **W3F**, the states of the two logic solvers are described with the finite degradation structure **WFsd**, the states of the two valves are described with the finite degradation structure **WFdu**, and the state of the unique root variable (`TA4`) is described with the finite degradation structure **W3F**.

Note that this model is not an abstraction, because the operator $\ominus$ introduces non-coherencies. Consider for instance the case where the three sensors are failed dangerous undetected and all other components are working. In this case, the HIPPS as a whole is failed dangerous undetected. Now, consider the case where the three sensors are failed dangerous undetected as previously but, in addition, the logic solver `LS1` is failed safe. In this case, the closure of valves is triggered, even in absence of an overpressure. Consequently, the HIPPS as a whole is failed safe. The latter state is however clearly more degraded than the former. It is often the case that partial orders induce this kind of non-monotone behavior. This is one of the reason why handling different types of failures, as suggested by safety standards such as IEC61508, is far from easy when staying in the realm of Boolean models.

From now and for the sake of the simplicity, we shall omit brackets $[\![.]\!]$ when it is clear from the context that we speak about the interpretation of a model $\mathcal{M}$ and not the model $\mathcal{M}$ itself.

3.6. Probability Measures

Let $\mathcal{C} : \langle C, \sqsubseteq_C, \bot_C \rangle$ be a finite degradation structure. We can associate $\mathcal{C}$ with a probability measure p, i.e., a function $p : C \rightarrow [0, 1]$ such that $\sum_{s \in C} p(s) = 1$.

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

Now, let $\mathcal{C} : \langle C, \sqsubseteq_C, \bot_C \rangle$ and $\mathcal{D} : \langle D, \sqsubseteq_D, \bot_D \rangle$ be two finite degradation structures equipped respectively with probability measures p_C and p_D.

Then, their product $\mathcal{C} \otimes \mathcal{D}$ can be associated with the natural probability measure p defined as follows. $\forall \langle s, t \rangle \in C \times D$,

$$p\left(\langle s, t \rangle\right) \stackrel{def}{=} p_C(s) \times p_D(t).$$

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

Let $\mathcal{C} : \langle C, \sqsubseteq_C, \bot_C \rangle$ and $\mathcal{D} : \langle D, \sqsubseteq_D, \bot_D \rangle$ be two finite degradation structures. Assume moreover that $\mathcal{C}$ is associated with the probability measure p_C. Let $\varphi : \mathcal{C} \rightarrow \mathcal{D}$ be a mapping from $\mathcal{C}$ into $\mathcal{D}$. Then, the natural probability measure p_D over $\mathcal{D}$ is defined as follows. $\forall t \in D$:

$$p_D(t) = \sum_{s \in \varphi^{-1}\{t\}} p_C(s).$$

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

4. Minimal Cutsets

The notion of minimal cutset plays a central role in system reliability theory, as well as in practical probabilistic risk analyses. Intuitively, in fault trees, a minimal cutset represents a minimal set of component failures that induces a failure of the system as a whole. In other words, minimal cutsets represent the most significant and in general the most probable scenarios of failure. This intuitive idea works fine for coherent models for which the notion of minimal cutset coincide with the classical notion of prime implicant. However, it needs to be refined to handle non-coherent ones [16].

In this section, we shall lift-up the notion of minimal cutset to finite degradation models. We shall also characterize minimal cutsets in terms of states of finite degradation structures.

4.1. Definition

One of the main objective of probabilistic risk and safety analyses is to characterize dangerous states of the system under study as well as the cumulated probability of these states. Dangerous states are represented by biggest elements of finite degradation structures. We can formalize this idea by considering upper sets.

Let $\mathcal{C} : \langle C, \leq, \bot \rangle$ be a finite degradation structure, an upper set of $\mathcal{C}$ is a subset U of C such that $\forall s, t \in C$, if $s < t$ and $s \in U$, then $t \in U$. Upper sets are sometimes called upward closed sets [24].

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model with a root variable R and let U be an upper set of $dom(R)$. We call the predicate $R \in U$ an observer of $\mathcal{M}$.

As an illustration, in the finite degradation model given Table 3, we could consider the observers $\texttt{TA4} \in \{\texttt{fs}\}$ (the system is failed safe), $\texttt{TA4} \in \{\texttt{fu}\}$ (the system is failed dangerous undetected), $\texttt{TA4} \in \{\texttt{fd}, \texttt{fu}\}$ (the system is failed dangerous) and $\texttt{TA4} \in \{\texttt{fs}, \texttt{fd}, \texttt{fu}\}$ (the system is failed).

We are interested in characterizing the subset of valuations σ of the state variables whose admissible extension $\vec{\sigma}$ verifies $\vec{\sigma}\ (R) \in U$, i.e., that satisfies the observer $R \in U$. To do so, we shall lift-up the notions of literals, product and minterm.

A literal over $\mathcal{S}$ is an equality of the form $V = c$, where V is a variable of $\mathcal{S}$ and c is constant of $dom(V)$.

A product over $\mathcal{S}$ is a conjunct of literals over $\mathcal{S}$ in which each variable of $\mathcal{S}$ occurs at most once.

A minterm over $\mathcal{S}$ is a product over $\mathcal{S}$ in which each variable of $\mathcal{S}$ occurs exactly once.

We denote by $\text{Minterms}\,(\mathcal{S})$ the set of minterms that can be built over a set $\mathcal{S}$ of variables.

Minterms one-to-one correspond with variable valuations of $\mathcal{S}$: the variable valuation assigns the value c to the variable V if and only if the literal $V = c$ occurs in the corresponding minterm. Products one-to-one correspond thus with partial variable valuations.

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model and $R \in U$ be an observer for $\mathcal{M}$.

For each valuation σ of the state variables, the root variable R takes a certain value $\vec{\sigma}\ (R)$. We do not care which specific value it takes, but whether this value belongs to U or not. Consequently, we can simply consider $\vec{\sigma}$ as a function from $dom(\mathcal{S})$ into $\{0, 1\}$, i.e., as the characteristic function of the set of state variable valuations σ such that $\vec{\sigma}\ (R) \in U$, or equivalently as the set of minterms verifying the same condition. We denote by $\mathcal{M}_{R \in U}$ this latter set:

$$\mathcal{M}_{R \in U} \stackrel{def}{=} \{\sigma \in \text{Minterms}\,(\mathcal{S})\,;\, \vec{\sigma}\ (R) \in U\}$$

Let π be a product over $\mathcal{S}$, we can extend π to obtain a minterm over $\mathcal{S}$ as follows.

$$\pi^c \stackrel{def}{=} \pi \wedge \bigwedge_{V \notin var(\pi)} V = \bot_{dom(V)}$$

For instance, in our example, if $\pi = \texttt{S2} = \texttt{fd} \wedge \texttt{S3} = \texttt{fu} \wedge \texttt{V1} = \texttt{fu}$, then π^c is as follows.

$$\pi^c \quad = \quad \texttt{S1} = \texttt{w} \wedge \texttt{S2} = \texttt{fd} \wedge \texttt{S3} = \texttt{fu} \wedge \texttt{LS1} = \texttt{w} \wedge \texttt{LS2} = \texttt{w} \wedge \texttt{V1} = \texttt{fu} \wedge \texttt{V2} = \texttt{w}$$

We can now define minimal cutsets:

- A product π is a cutset for the observer $R \in U$ if $\overrightarrow{\pi^c}(R) \in U$.
- A cutset π for $R \in U$ is minimal if none of its proper subproducts is a cutset.

We denote by $\text{MCS}(\mathcal{M}_{R \in U})$ the set of minimal cutsets for the observer $R \in U$ (in the model $\mathcal{M}$).

For instance, in the model given Table 3, the product $\pi = \texttt{S2} = \texttt{fd} \wedge \texttt{S3} = \texttt{fu} \wedge \texttt{V1} = \texttt{fu}$ is a cutset for the observer $\texttt{TA4} \in \{\texttt{fu}\}$ because $\overrightarrow{\pi^c}(\texttt{TA4}) = \texttt{fu}$. However, it is not minimal because, $\rho = \texttt{S2} = \texttt{fd} \wedge \texttt{S3} = \texttt{fd} \wedge \texttt{V1} = \texttt{fu}$ verifies also $\overrightarrow{\rho^c}(\texttt{TA4}) = \texttt{fu}$ and $\rho < \pi$. It is easy to verify that ρ is minimal if the valve V1 is not failed undetected, then the first safety barrier SB1 is not failed undetected, and if either sensors S2 or S3 are not failed dangerous, then the second safety barrier SB2 is not failed dangerous, therefore it cannot produce a dangerous undetected failure of the HIPPS in combination with the dangerous undetected failure of the valve V1.

4.2. Minimal States

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model and $R \in U$ be an observer for $\mathcal{M}$.

Let σ be a state of $dom(\mathcal{S})$, or equivalently a variable valuation of $\mathcal{S}$. We say that σ is minimal for $R \in U$, if the following conditions hold.

- $\overrightarrow{\sigma}(R) \in U$.
- There is no other state ρ of $dom(\mathcal{S})$ such that $\overrightarrow{\rho}(R) \in U$ and $\rho < \sigma$.

For instance, in the model given Table 3, the state $\overrightarrow{\rho^c}$, where ρ is defined as above, is minimal for the observer $\texttt{TA4} \in \{\texttt{fu}\}$.

We denote by $\text{MinimalStates}_{\mathcal{M}}(R \in U)$ the set of minimal states of $dom(\mathcal{S})$ for the observer $R \in U$.

The following theorem follows from the definitions.

Theorem 3 (Minimal Cutsets versus Minimal States)**.** *Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model and $R \in U$ be an observer for $\mathcal{M}$. Then,*

$$MCS(\mathcal{M}_{R \in U}) \quad \cong \quad MinimalStates_{\mathcal{M}}(R \in U)$$

The above theorem presents no difficulty: it asserts that minimal cutsets one-to-one correspond with minimal states for the chosen degradation order (the sets of minimal cutsets and minimal states are isomorphic), i.e., they are essentially the same thing. It is actually easy to check that the product π is a minimal cutset if and only if its extension π^c is a minimal state. Despite of its simplicity, this theorem is probably the most important of system reliability theory, as it shows that the core notion of minimal cutsets arises naturally from the more fundamental notion of degradation order. Finite degradation structures capture the latter. This is the reason why they are a central concept of system reliability theory, at least for what concerns combinatorial models.

4.3. Decomposition Theorem

Decomposition theorems are recursive principles at the core of algorithms to extract prime implicants [25] and minimal cutsets [16]. We shall now extend the latter. But before doing so, we need to introduce a few notations and results on sets of products.

Let $\mathcal{P}$ be a set of products built over a set of variables $\mathcal{V}$, let $V \in \mathcal{V}$ and $W \notin \mathcal{V}$ be two variables and finally let c and d be two constants. Then ($\setminus$ stands for the set difference),

$$\begin{aligned}
\mathcal{P}|_{V=c} &\stackrel{def}{=} \{\pi; V = c \wedge \pi \in \mathcal{P}\} \\
\mathcal{P}|_{V\leq c} &\stackrel{def}{=} \{\pi; V = d \wedge \pi \in \mathcal{P}, d \leq c\} \\
\mathcal{P}|_{V\neq c} &\stackrel{def}{=} \mathcal{P} \setminus \mathcal{P}|_{V=c} \\
\mathcal{P}|_{\widetilde{V}} &\stackrel{def}{=} \{\pi \in \mathcal{P}; V \notin var(\pi)\} \\
W = d \oslash \mathcal{P} &\stackrel{def}{=} \{W = d \wedge \pi; \pi \in \mathcal{P}\}
\end{aligned}$$

The following equality holds.

Lemma 4 (Pivot decomposition)**.** *Let $\mathcal{P}$ be a set of products built over a set of variables $\mathcal{V}$, let $V \in \mathcal{V}$ be a variable and finally let c be a constant. Then,*

$$\mathcal{P} \quad = \quad V = c \oslash \mathcal{P}|_{V=c} \cup \mathcal{P}|_{V\neq c}$$

We extend the degradation order to products as follows. Let π and ρ be two products built over a set of variables $\mathcal{V}$, then $\pi \sqsubset \rho$ if either of the two following condition holds.

- $\pi^c < \rho^c$, or
- $\pi^c = \rho^c$ and $var(\pi) \subset var(\rho)$.

The second condition makes it possible to distinguish π and $\rho = (V = \perp_{dom(V)}) \wedge \pi$.

Now, let $\mathcal{P}$ and $\mathcal{Q}$ be two sets of products built over the set of variables $\mathcal{V}$. The operator $\div$ generalizes the Boolean without operator introduced in [16].

$$\mathcal{P} \div \mathcal{Q} \quad \stackrel{def}{=} \quad \{\pi \in \mathcal{P}; \nexists\, \rho \in \mathcal{Q} \text{ s.t. } \rho \sqsubseteq \pi\}$$

The following decomposition theorem holds.

Theorem 5 (Decomposition of without operator)**.** *Let $\mathcal{P}$ and $\mathcal{Q}$ be two sets of products built over the set of variables $\mathcal{V}$, let $V \in \mathcal{V}$ and let c be a constant. If $\mathcal{Q}$ does not contain products of the form $V = d \wedge \pi$ with $d > c$, then,*

$$\mathcal{P} \div \mathcal{Q} \quad = \quad V = c \oslash \left((\mathcal{P}|_{V=c} \div \mathcal{Q}|_{V\leq c}) \div \mathcal{Q}|_{\widetilde{V}}\right) \cup \mathcal{P}|_{V\neq c} \div \mathcal{Q}|_{V\neq c}$$

The proof follows from the definitions and the remark that for any three sets of products $\mathcal{P}$, $\mathcal{Q}$ and $\mathcal{R}$, $\mathcal{P} \div (\mathcal{Q} \cup \mathcal{R}) = (\mathcal{P} \div \mathcal{Q}) \div \mathcal{R}$.

The following decomposition theorem holds, which is at the core of the decision diagrams algorithm to extract minimal cutsets (the proof follows from the definitions and previous lemmas).

Theorem 6 (Decomposition of minimal cutsets)**.** *Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model, $R \in U$ be an observer for $\mathcal{M}$, let $V \in \mathcal{S}$ and $c \in dom(V)$. Moreover, let $\mathcal{P} = MCS\left(\mathcal{M}_{R\in U}|_{V=c}\right)$ and $\mathcal{Q} = MCS\left(\mathcal{M}_{R\in U}|_{V\neq c}\right)$. If $\mathcal{Q}$ does not contain products of the form $V = d \wedge \pi$ with $d > c$, then,*

$$MCS\left(\mathcal{M}_{R\in U}\right) \quad = \quad V = c \oslash \left((\mathcal{P} \div \mathcal{Q}|_{V<c}) \div \mathcal{Q}|_{\widetilde{V}}\right) \cup \mathcal{Q}$$

The proof follows again from the definitions.

The theoretical framework is now in place, we can move to algorithmic issues.

5. Decision Diagram Algorithms

The modern implementation of binary decision diagrams has been introduced by Bryant and his colleagues [26]. We shall rely here on this implementation as well as on ideas introduced by Minato [19]. The algorithms presented in this section generalize to finite degradation models those developed by Rauzy for Boolean models [16].

5.1. Implicative Normal Form

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model. Each equation $W := f$ of $\mathcal{E}$ describes eventually a function from $dom(\mathcal{S})$ into $dom(W)$.

We want to write a logical formula that encodes this function. The implicative normal form is a way to do so. To define this normal form, we need a few additional definitions.

Two products π and ρ are disjoint if they contain two different literals built over the same variable, e.g., $\pi = (V = c) \wedge \pi'$ and $\rho = (V = d) \wedge \rho'$.

An implication is a formula in the form $\pi \to W = c$, where π is a product and $W = c$ is a literal. The product π is the hypothesis of the implication. The literal $W = c$ is the consequence of the implication.

A formula f is in implicative normal form if it is a conjunct of implications such that:

- Consequences of the implications of f are literals built over the same variable.
- Hypotheses of the implications of f are pairwisely disjoint products.

Let f be a formula in implicative normal form and let $\mathcal{H}$ be the set of variables that show up in the hypotheses of the implications of f, i.e., $var(f) = \mathcal{H} \cup \{W\}$, where W is the variable on which consequences of the implications of f are built. Then f is complete if for any valuation σ of the variables of f, there exists an implication $\pi \to W = d$ of f such that $\sigma(\pi) = 1$. As the hypotheses of f are pairwisely disjoint, this implication, if it exists, is unique.

Consider for instance the variable `SL2` of the model given in Table 3. Table 4 gives a complete formula in implicative normal form that defines this variable. This formula is not the simplest one may imagine *a priori*, but we shall see that it can be encoded efficiently by means of a decision diagram.

Finally, we need to lift-up the definition of the operator $\oslash$ introduced in the last section to formulas in an implicative normal form. Let f be a formula in implicative normal form, let V be a variable and let c be a constant, then:

$$V = c \oslash f \quad \stackrel{def}{=} \quad \bigwedge_{\pi \to W = d \in f} V = c \wedge \pi \to W = d.$$

Table 4. A complete formula in implicative normal form that encodes the definition of `SL2`.

	`S2=w∧S3=w` → `SL2=w`
∧	`S2=w∧S3=fs` → `SL2=fs`
∧	`S2=w∧S3=fd` → `SL2=w`
∧	`S2=w∧S3=fu` → `SL2=w`
∧	`S2=fs` → `SL2=fs`
∧	`S2=fd∧S3=w` → `SL2=w`
∧	`S2=fd∧S3=fs` → `SL2=fs`
∧	`S2=fd∧S3=fd` → `SL2=fd`
∧	`S2=fd∧S3=fu` → `SL2=fu`
∧	`S2=fu∧S3=w` → `SL2=w`
∧	`S2=fu∧S3=fs` → `SL2=fs`
∧	`S2=fu∧S3=fd` → `SL2=fu`
∧	`S2=fu∧S3=fu` → `SL2=fu`

5.2. Decision Diagrams

Let $\mathcal{M} : \langle \mathcal{V} = \mathcal{S} \uplus \mathcal{F}, \mathcal{E} \rangle$ be finite degradation model. Each equation $W := f$ of $\mathcal{E}$ describes a function from $dom(\mathcal{S})$ into $dom(W)$. The decision diagram associated with the variable W encodes this function. To build this decision diagram, one needs first to build the decision diagrams associated with all variables of $var(f)$, then to build the decision diagram encoding f. The first step consists thus in building the decision diagrams associated with each variable by traversing bottom-up the model.

Building these decision diagrams requires choosing an order over the literals $V = c$, where $V \in \mathcal{S}$ and $c \in dom(V)$, i.e., associating each literal $V = c$ with an index $\iota(V = c)$. This indexing must fulfill the following constraints.

- Indices of the literals built over a variable $V \in \mathcal{S}$ must be consecutive, i.e., it is not allowed to intertwine indices associated with literals built over different variables.
- For any variable $V \in \mathcal{S}$ and $c, d \in dom(V)$, if $c < d$, then $\iota(V = c) > \iota(V = d)$.

The justification for these two constraints will appear in the next section.

A decision diagram is a directed acyclic graph with three types of nodes:

- The leaf `nil`.
- Other leaves, which are labeled with constants.
- Internal nodes, which are labeled with the index of a literal and have two out-edges: the then out-edge and the else out-edge.

In the sequel, we shall denote $\square(c)$ the leaf labeled with the constant c, and $\Delta(i, m, n)$ the internal node labeled with the index i and whose then and else out-edges point respectively to nodes m and n. m and n are called respectively the then-child and the else-child of the node $\Delta(i, m, n)$.

Decision diagrams are ordered, i.e., that the index labeling an internal node is always smaller than the indices labeling its children (if they are internal nodes).

A decision diagram encodes a formula in implicative normal form. The semantics of decision diagrams is defined then recursively as follows.

$$\begin{aligned} [\![\texttt{nil}]\!] &\stackrel{def}{=} 1 \\ [\![\square(c)]\!] &\stackrel{def}{=} 1 \to W = c \\ [\![\Delta(i, m, n)]\!] &\stackrel{def}{=} V = c \oslash [\![m]\!] \wedge [\![n]\!] \quad \text{where } \iota(V = c) = i \end{aligned}$$

Decision diagrams are built bottom-up, i.e., that nodes m and n are always built before the node $\Delta(i, m, n)$. This makes it possible to store nodes into a unique table. When one needs the leaf $\square(c)$ or the internal node $\Delta(i, m, n)$, the unique table is looked up. If the node is already present in the table, it is returned. Otherwise, it is created and inserted in the table. This technique, introduced in reference [26], ensures the unicity of each node in the table. It is one of the main reasons, if not the main reason, of the efficiency of the decision diagrams technology. The insertion and look-up in the unique table are implemented by means of hashing techniques, so to ensure a nearly linear complexity.

The semantics of decision diagrams makes it possible to simplify the representation. The node $\Delta(i, \texttt{nil}, n)$ encodes actually the formula $V = c \oslash 1 \wedge [\![n]\!]$, which is equivalent to $[\![n]\!]$. $\Delta(i, \texttt{nil}, n)$ is thus useless and can be replaced by n. A decision diagram is reduced if it contains no node of the form $\Delta(i, \texttt{nil}, m)$. This reduction generalizes to the one of Minato's zero-suppressed binary decision diagrams [19].

From now, we shall simply say the decision diagram, but we shall mean the reduced, ordered decision diagram.

5.3. Operations

The calculation of the decision diagram encoding a formula f is performed bottom-up. There are four cases:

- $f = c$, for some constant c. In this case, the unique table is looked up for the node $\square(c)$, which is added if it did not exist before.
- $f = V$, for some state variable V. In this case, the decision diagram associated with this variable is built. This decision diagram is made of a series of internal nodes in the form $\Delta(i, \square(c), m)$, where i is the index of the literal $V = c$, as illustrated in Figure 5 for the variable `S2` of our example.
- $f = W$, for some flow variable W. In this case, the decision diagram associated with W is calculated (via the equation defining W) and returned.
- $f = op(f_1, \ldots, f_k)$ for some operator op. In this case, the decision diagrams for the arguments f_1, $\ldots$, f_k are calculated, then the operator op is applied onto these decision diagrams. The application of op may require the applications of more primitive operators. For instance, the calculation of $f \wedge g \wedge h$ requires first the calculation of $r = g \wedge h$, then the calculation of $f \wedge r$.

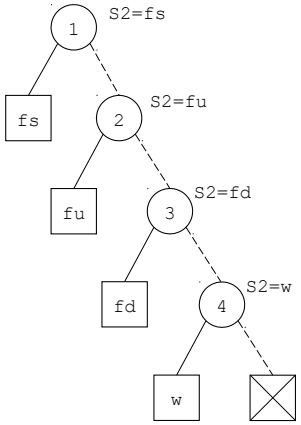

Figure 5. The decision diagram encoding the state variable `S2`.

Algorithm 1 gives the pseudo-code of the function that performs binary operations on decision diagrams. The algorithms for operations with different arities (unary, ternary,...) can be easily derived for this one.

The first step of the algorithm (line 2) consists in checking whether the result can be calculated directly, e.g., if the first argument is the leaf $\square(f)$ while computing a $\circledcirc$. If so, the result is returned. In general, when one of the arguments is `nil`, `nil` is returned.

Then, one looks whether the result on the operation has been cached (line 5). If so, the result is returned. Otherwise, it is computed and inserted into the cache (line 30). Caching techniques, which have been introduced for binary decision diagrams in the seminal article by Bryant et al. [26], are the second pillar of the efficiency of the technology. As unique tables, caches are managed by means of hashing techniques. In case binary operators are commutative, the caching policy can be improved by setting an order over the arguments.

The core of the algorithm consists in a case study on the type of arguments. In the case where the first argument is a node $m : \Delta(i, m_1, m_0)$ and the second one a node $n : \Delta(j, n_1, n_0)$ such that the literals corresponding to indices i and j, $i < j$ are respectively $V = c$ and $V = d$ (line 17), the function must be recursively called on m_1. However, it cannot be called on n, because the then-branch of n assigns also the variable V. This is the reason why the function `GetOtherwiseChild` is called.

In Algorithm 1, the function `NewNode` that creates a new node (e.g., line 29), checks whether the node `r` is `nil`, in which case it returns `r0`.

Algorithm 1 Pseudo-code for the function that performs binary operation on decision diagrams.

```
define PerformBinaryOperation(op: operation, m: Node, n: Node)
r = CalculateTerminalResult(op, m, n)
if r!=None:
return r
r = LookForEntryInCache(op, m, n)
if r!=None:
return r
if m==□(c) and n==Δ(j,n1,n0):
r1 = PerformBinaryOperation(op, m, n1)
r0 = PerformBinaryOperation(op, m, n0)
r = NewNode(j, r1, r0)
else if m==Δ(i,m1,m0) and n==□(d):
r1 = PerformBinaryOperation(op, m1, n)
r0 = PerformBinaryOperation(op, m0, n)
r = NewNode(i, r1, r0)
else if m==Δ(i,m1,m0) and n==Δ(j,n1,n0) and i<j:
k = GetOtherwiseChild(i, n)
r1 = PerformBinaryOperation(op, m1, k)
r0 = PerformBinaryOperation(op, m0, k)
r = NewNode(i, r1, r0)
else if m==Δ(i,m1,m0) and n==Δ(j,n1,n0) and i>j:
k = GetOtherwiseChild(j, m)
r1 = PerformBinaryOperation(op, k, n1)
r0 = PerformBinaryOperation(op, k, n0)
r = NewNode(i, r1, r0)
else m==Δ(i,m1,m0) and n==Δ(i,n1,n0)
r1 = PerformBinaryOperation(op, m1, n1)
r0 = PerformBinaryOperation(op, m0, n0)
r = NewNode(i, r1, r0)
AddEntryToCache(op, m, n, r)
return r

define GetOtherwiseChild(i: index, m: Node)
if m==Node(j, m1, m0) and Variable(i)==Variable(j)
return GetOtherwiseChild(i, m0)
return m
```

Discussion

The algorithm to build the decision diagrams associated with variables of a finite degradation model is thus similar to the one to build the binary decision diagrams associated with variables of a Boolean model, e.g., fault tree or reliability block diagram. However, two nodes are used for each Boolean variable: one for the value 1 and one for the value 0. This induces an overcost compared to the algorithm dedicated to the binary case. However, this overcost is acceptable, especially when truncations are applied (see next section).

All complexity results that hold for binary decision diagrams see e.g., [27] for a review, hold for decision diagrams as well. In particular, caching ensures that the worst case complexity of the function `PerformBinaryOperation` is in the product of the sizes of its arguments (without caching, it is exponential).

It is well known that the size of binary decision diagrams, and therefore the efficiency of the whole method, depends heavily on the chosen variable ordering, see again [27]. The same applies indeed to decision diagrams. As the calculation of an optimal ordering is computationally intractable, heuristics are applied which are essentially the same as for the Boolean case, see e.g., [28]. The only difference regards dynamic reordering of decision diagrams, notably sifting [29]. As indices of literals built over the variable must be consecutive, group sifting has to be applied rather than individual sifting.

Note finally that efficient algorithms exist to calculate probabilistic risk indicators such as the top-event probability or importance measures [30]. These algorithms can be lifted-up to decision diagrams. The key idea is to extend the pivotal decomposition: assume that we have a decision diagram encoding a function $dom(\mathcal{S}) \to dom(W)$ and that we want to calculate the probability $p(W \in U)$ that $W \in U$ for some subset U of $dom(W)$. Then the recursive equations to do so are as follows.

$$\begin{aligned} p(\texttt{nil}) &= 0 \\ p(\square(c)) &= \begin{cases} 1 & \text{if } c \in U \\ 0 & \text{otherwise} \end{cases} \\ p(\Delta(i,m,n)) &= p(i) \times p(m) + p(n). \end{aligned}$$

The mathematical definition of importance measures for finite degradation models is, however, non-trivial and will be the subject of a forthcoming article.

5.4. Extraction of Minimal Cutsets

As in the Boolean case [16], the principle of the extraction of minimal cutsets is to build a second decision diagram encoding the minimal cutsets from the one associated with the root variable the model. Technically, this second decision diagram is a zero-suppressed binary decision diagram. Zero-suppressed binary decision diagrams have been introduced by Minato [19]. They encode sets of products and not Boolean functions:

- They have two leaves: the leaf $\square(1)$ which encodes the set containing only the empty product and the leaf $\square(0)$ which encodes the empty set of products.

$$\begin{aligned} [\![\square(1)]\!] &= \{1\} \\ [\![\square(0)]\!] &= \emptyset \end{aligned}$$

- An internal node $\Delta(i, n_1, n_0)$ encodes the set of products defined by the pivot decomposition (Lemma 4), i.e.,

$$[\![\Delta(i,n_1,n_0)]\!] = (V = c) \oslash [\![n_1]\!] \cup [\![n_0]\!]$$

  where $\iota(V = c) = i$.

For this reason, they have a different reduction rule, namely $\Delta(i, \square(0), n) \equiv n$, hence their name. This semantics is actually almost the same as the one we have given for decision diagrams: the leaf `nil` of decision diagrams plays the role of the leaf $\square(0)$ of the zero-suppressed binary decision diagrams, all what we need is another leaf to play the role of the leaf $\square(1)$, we shall denote it $\square(1)$.

The algoritm to extract minimal cutsets follows the two decomposition Theorems 5 and 6 demonstrated Section 4. Algorithm 2 gives the pseudo-code for the functions extracting minimal cutsets. The operator $\div$ (without) is implemented by means of two mutually recursive functions. Algorithms 3 and 4 give their respectective pseudo-codes.

The code for the function `ExtractMinimalCutsets` is essentially the same in the Boolean case. The only difference stands in the test whether the constant labeling a leaf belongs to the upper set or not.

The code for the without operator is trickier than in the Boolean case. One has actually to take into account the degradation order. It is worth noticing that the algorithm relies heavily on the order chosen over the literals built over the same variable. The algorithm works because we are sure that else-branches do not contain nodes labeled with literal smaller than the literal labeling the node.

Algorithm 2 Pseudo-code for the function extracting the minimal cutsets.

```
ExtractMinimalCutsets(U: UpperSet, m: Node)
if m==□(c):
if c in U:
return NewLeaf(1)
return NewLeaf(0)
r = LookForEntryInCache(ExtractMinimalCutsets, U, m)
if r!=None:
return r
m==Δ(i,m1,m0)
r0 = ExtractMinimalCutsets(U, m0)
r1 = ExtractMinimalCutsets(U, m1)
r1 = Prune(i, r1, r0)
r = NewZBDDNode(i, r1, r0)
AddEntryToCache(ExtractMinimalCutsets, U, m, r)
return r
```

Algorithm 3 Pseudo-code for the function `Prune` implementing the without operator.

```
Prune(i: Index, m: Node, n: Node)
if n==nil:
return m
if n==□(1):
return nil
n==Δ(j,n1,n0)
if Variable(i)!=Variable(j):
r = Without(m, n)
else if Constant(i)>Constant(j):
r = Without(m, n1)
r = Prune(i, r, n0)
else:
r = Prune(i, m, n0)
return r
```

Algorithm 4 Pseudo-code for the function `Without` implementing the without operator.

```
Without(m: Node, n: Node)
if m==nil or n==□(1):
return nil
if m==□(1) or n==nil:
return m
m==Δ(i,m1,m0)
n==Δ(j,n1,n0)
if i>j:
return Without(m, n0)
r = LookForEntryInCache(Without, m, n)
if r!=None:
return r
if i<j:
r0 = Without(m0, n)
r1 = Prune(i, m1, n)
else: // i==j
r0 = Without(m0, n0)
r1 = Without(m1, n1)
r1 = Prune(i, r1, n0)
r = NewZBDDNode(i, r1, r0)
AddEntryToCache(Without, m, n, r)
return r
```

6. Approximation Schemes

6.1. Rational

Industrial probabilistic risk assessment models may have a very large number of minimal cutsets. As of today, the largest models are event trees designed to assess nuclear safety. These models involve thousands of basic events and gates. Their number of minimal cutsets is just astronomical. However, most of these minimal cutsets have an extremely low probability. Intuitively, the average probability of each minimal cutset must decrease as the number of minimal cutsets increases. Minimal cutsets with a low probability should be ignored, at least for practical reasons. First, they represent a negligible risk. Second, given the uncertainties on basic event probabilities, ranking them according to their probability is hopeless. Third, there are anyway too many of them to expect to retrieve any valuable information out of their extraction. Fourth, extracting millions and millions of minimal cutsets would require much too much computation resources, both in terms of computation time and in terms of computer memory. This is the reason why algorithms implemented in assessment tools such as risk spectrum [14] or XFTA [15], are designed to extract only the most probable minimal cutsets, i.e., with a probability higher than a given predefined threshold. These algorithms can be seen as model pruners as they eliminate irrelevant parts of models, i.e., those that induce minimal cutsets with a low probability [31]. It has been observed that up to 95% of the basic events do not show up in the extracted minimal cutsets [32].

From a theoretical point of view, these approximations cannot be warrantied: it may be the case that, although the risk is high, each individual minimal cutset has a probability lower than the chosen threshold. This phenomenon is called the model refinement paradox in reference [2]. The more one refines a model, i.e., the more scenarios of failure are decomposed, the more minimal cutsets go under the algorithmic horizon, which eventually leads to eliminate all of them and conclude (falsely) that there is no risk at all. As demonstrated by Valiant [33], reliability problems are #P-hard, which means concretely that no efficient algorithm to assess them is expected to exist, even if we accept an approximate, but warrantied answer.

Nevertheless, approximations based on probabilistic cutoffs give satisfying results in practice. The question is thus how to implement such approximations within the decision diagram technology. Figure 6 helps to understand the problem at stake.

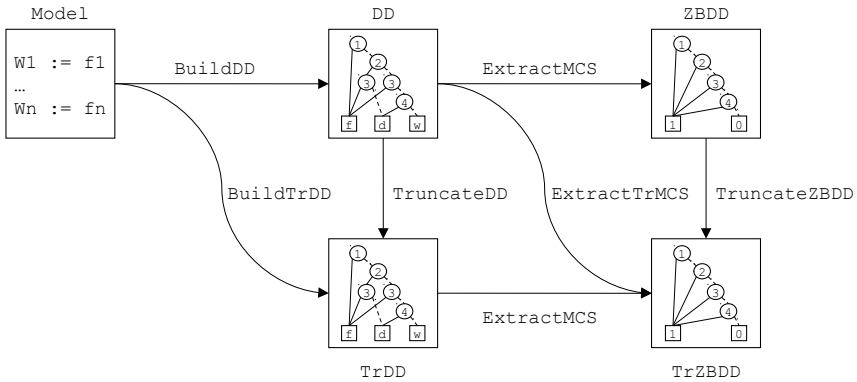

Figure 6. Approximation schemes for the extraction of minimal cutsets.

The minimal cutsets extraction process we described in the previous section is represented by the upper line of the figure: first, the decision diagram encoding the model is built by means of the `BuildDD` algorithm, then the zero-suppressed binary decision diagram encoding the minimal cutsets of the observer is calculated by means of the `ExtractMCS` algorithm. The problem with this process stands in the sizes of decision diagram and zero-suppressed binary decision diagram, which may be exponential in the number of literals showing up in the model.

It is, of course, possible to filter the zero-suppressed binary decision diagram so to keep only minimal cutsets with a probability higher than a given threshold, by means of a `TruncateZBDD` algorithm, but this does not solve the above problem.

A first approach consists in extracting only the most probable minimal cutsets from the decision diagram, combining the `ExtractMCS` and the `TruncateZBDD` algorithms into a `ExtractTrMCS` algorithm. This solution, although it still does not solve the exponential blow-up problem, has nevertheless some virtues: exact values of probabilistic indicators can be calculated from the decision diagram, while (most probable) minimal cutsets can be used for qualitative assessment purposes.

A more radical approach consists in performing the approximation onto the decision diagram itself by means of an algorithm `BuildTrDD` that combines the algorithm `BuildDD` performing the regular construction of decision diagram with the algorithm `TruncateDD` that truncates this decision diagram. The idea is that the truncated decision diagram encodes not the model itself, but a model whose minimal cutsets are those of the original model that have a probability higher than the given threshold. In other words, the diagram given Figure 6 commutes:

$$\begin{aligned} \texttt{TruncateZBDD} \circ \texttt{ExtractMCS} \circ \texttt{BuildDD} &= \texttt{ExtractTrMCS} \circ \texttt{BuildDD} \\ &= \texttt{ExtractMCS} \circ \texttt{TruncateDD} \circ \texttt{BuildDD} \\ &= \texttt{ExtractMCS} \circ \texttt{BuildTrDD} \end{aligned}$$

We shall see that the last combination warranties a polynomial worst case complexity of the process.

6.2. Implementation

Algorithm 5 gives the pseudo-code to truncate a decision diagram for a given probability threshold.

Several important remarks must be made about this algorithm.

First, $\bot$ stands for the bottom element of the domain of the flow variable the decision diagram is associated with. If the decision diagram is a zero-suppressed decision diagram, then $\bot = \emptyset$. In the form given Algorithm 5, the algorithm works only for monotone functions (coherent models). If non-monotone functions must be represented, the best solution consists in introducing as special constant $\star$ and to return $\square(\star)$ when the decision diagram is truncated. This reduces the reduction of decision diagrams and forces to take into account the value $\star$ in all operations. But in some sense, it is mathematically "cleaner" than just using $\bot$. Moreover, it makes it possible to calculate bounds for the probability of an observer, rather than just an approximation: a lower bound is obtained by setting the probability of $\square(\star)$ to 0, while a upper bound is obtained by setting it to 1.

Algorithm 5 Pseudo-code for the truncation algorithm.

```
define TruncateDecisionDiagram(m: Node, pr: probability)
if m==nil:
return nil
if pr<threshold:
return NewLeaf(⊥)
if m==□(c):
return NewLeaf(c)
m==Δ(i,m1,m0)
r1 = TruncateDecisionDiagram(m1, pr⋆p(i))
r0 = TruncateDecisionDiagram(m0, pr)
r = NewNode(i, r1, r0)
return r
```

Second, conversely to the algorithm we have seen so far, `TruncateDecisionDiagram` does not cache intermediate results. The reason is that it is hard to make truncation and caching compatible.

Attempts to discretize probabilities, e.g., by caching $\lfloor -\log(\texttt{pr}) \rfloor$ rather than `pr`, although nice conceptually, do not bring much in practice: one tends to store too many intermediate results in the cache, which makes lose here what we win there.

Third, this algorithm is intended to be applied to truncate decision diagrams and zero-suppressed decision diagrams calculated from formulas, i.e., at a relatively coarse grain. The reason is again to make the truncation compatible with other operations.

Fourth, the algorithm does not take into account probabilities of literals $V = \bot_{dom(V)}$. The reason is that it is the only way to make the algorithm insensitive to the literal ordering. We shall come back on this point.

6.3. Complexity

To discuss the complexity of the approximate assessment process, we shall state a general theorem of the size of binary trees.

The size and the depth of a binary tree are defined as usual as, respectively, the number of nodes and the maximum number of levels of the tree.

The Horton-Strahler measure $HS(t)$ of a binary tree t is defined as follows [34]:

$$HS(t) \stackrel{def}{=} \begin{cases} 0 & \text{if } t \text{ is a leaf.} \\ HS(t_1) + 1 & \text{if } t \text{ is an internal node } \Delta(t_1, t_2) \text{ and } HS(t_1) = HS(t_2) \\ \max(HS(t_1), HS(t_2)) & \text{if } t \text{ is an internal node } \Delta(t_1, t_2) \text{ and } HS(t_1) \neq HS(t_2) \end{cases}$$

The Horton-Strahler measure is almost like depth, except that one adds one to the children depth only if both children have the same depth, while for depth, one adds one (to the maximum depth of children) in any case. For this reason, the Horton–Strahler measure is always smaller than the depth.

The theorem goes as follows.

Theorem 7 (Size of binary trees)**.** *Let t be a binary tree of depth n and of Horton–Strahler measure k, $k \leq n$. Then, the size of t is in $\mathcal{O}\left(n^k\right)$*

Proof. We can assume, without a loss of generality, that the right branches of the tree are always at least as deep as the left branches. Now, we can make a proof by induction. If $HS(t) = 1$, then the left-child of the root must be a leaf and the right branch can involve at most n nodes. If $HS(t) = k$, then, by induction hypothesis, the left branches can contain at most n^{k-1} nodes and there is less than n such branches.

It is in general the case that components of a system are relatively reliable, i.e., the probability they are working correctly at time t is (much) greater than $\frac{1}{2}$. This means in turn that the probabilities of the other states must be less than $\frac{1}{2}$. Therefore, the probability of a product of k such literals must be less than $\frac{1}{2^k}$. Consequently, if we put a threshold τ on the minimum probability of the products we consider, the size of these products must be smaller than $-\log_2(\tau)$. Hence the following corrolary of Theorem 7. □

Corrolary 8 (Number of recursive calls to `TruncateDecisionDiagram`)**.** *Let d be the decision diagram representing a function built over n variables $V_1, \ldots V_n$. Assume that $p(V_i = \bot_{dom(V_i)}) \geq \frac{1}{2}$ for $i = 1, \ldots n$. Then, the number of recursive calls to `TruncateDecisionDiagram` on d for a threshold τ is in $\mathcal{O}\left(n^{-\log_2(\tau)}\right)$, i.e., polynomial in n if τ is fixed independently of the problem. Consequently, the size of the returned decision diagram is also polynomial in n.*

As we have seen, all of the other algorithms are polynomial in the size of their arguments, thanks to caching mechanisms. Hence the following second corollary.

Corrolary 9 (Complexity of the approximation scheme)**.** *Let $\mathcal{M}$ be a finite degradation model, $R \in U$ be an observer for $\mathcal{M}$, and $0 < \tau < 1$ be a thresold. Then, the process of calculating $\{\pi \in MCS\,(\mathcal{M}_{R \in U})\,; p(\pi) \geq \tau\}$ is $\mathcal{O}\left(n^{-\log_2(\tau)}\right)$, i.e., polynomial in n.*

The algorithm `ExtractMCS` ∘ `BuildTrDD` is thus efficient in the usual sense of computational complexity theory.

7. Experimental Results

This section reports a few experimental results we got on the HIPPS TA4 described Section 2.

7.1. Modeling

Table 3 gives the equations implementing a fault tree-like model (pictured in Figure 2) for the HIPPS TA4. This fault tree makes it possible to study different types of failures by means of a unique model, both at component and at system level. To the best of authors' knowledge, such combinations have not been formally defined in the standard nor in any other previous work.

Fault trees are not the only way of describing the failures of a system as a combination of the failures of its components. It is sometimes more convenient to use reliability block diagrams because they reflect better the architecture of the system and the way flows of information, matter and energy circulate in the network of components. With that respect, Figure 1 can be seen as a block diagram. Each component can be seen as a basic block, with an internal state, some input and some output flows. Failures propagate through the block diagram. Using operators $\circledcirc$ and $\|$, the model for the whole safety instrumented could be as described in Table 5.

The sets of equations described in Table 3 and in Table 5 are actually equivalent.

For the experiments reported below, we implemented a block-diagram model using the S2ML+FDS domain-specific modeling language. S2ML+FDS is one of the languages of the S2ML+X family [35]. This model is described in Appendix A. S2ML+FDS enhances the descriptive power of finite degradation structures with structuring constructs stemmed from object-oriented programming. For this reason, it is much more powerful than the (implicit) internal modeling languages of tools such as HiP-HOPS [36].

Table 5. Equations implementing a block diagram model for the HIPPS TA4.

`S1.input`	:=	`w`	`S1.output`	:=	`S1.input` ⊚ `S1.state`
`S2.input`	:=	`w`	`S2.output`	:=	`S2.input` ⊚ `S2.state`
`S3.input`	:=	`w`	`S3.output`	:=	`S3.input` ⊚ `S3.state`
`LS1.input`	:=	`S1.output`	`LS1.output`	:=	`LS1.input` ⊚ `LS1.state`
`LS2.input`	:=	`S2.output` ‖ `S3.output`	`LS2.output`	:=	`LS2.input` ⊚ `LS2.state`
`V1.input`	:=	`LS1.output` ‖ `LS2.output`	`V1.output`	:=	`V1.input` ⊚ `V1.state`
`V2.input`	:=	`LS2.output`	`V2.output`	:=	`V2.input` ⊚ `V2.state`
`state`	:=	`V1.output` ‖ `V2.output`			

7.2. Results

Results reported in this section have been obtained on the model described Appendix A with Emmy, a prototypical implementation of decision diagram algorithms to assess S2ML+FDS models.

Sensors can be in 4 different states, logic solvers and valves in 3. Therefore, the total number of possible states for the system is $4^3 \times 3^4 = 5184$. The decision diagram associated with the variable representing the global state of the safety instrumented system is made of 147 nodes. It encodes 3584 implications.

Table 6 reports the probabilities calculated at $t = 4380$ (six months) and $t = 8760$ (one year) for the system to be failed safe, failed dangerous and failed dangerous undetected, calculated from the decision diagram.

Table 6. Probabilities of failure of the TA4 system.

Model	$t = 4380$	$t = 8760$
Failed safe {`fs`}	6.31×10^{-2}	1.21×10^{-1}
Failed dangerous {`fd, fu`}	2.43×10^{-4}	9.57×10^{-4}
Failed dangerous undetected {`fu`}	3.85×10^{-5}	1.46×10^{-4}

The five minimal cutsets describing the conditions under which the system is failed safe are given Table 7. They are encoded by a decision diagram made of seven7 nodes. This result is not surprising: if any of the electronic component fails safe, the production is stopped.

Table 7. Minimal cutsets of the observer `failedSafe`.

Probabilities		Minimal Cutsets
$t = 4380$	$t = 8760$	
1.31×10^{-2}	2.59×10^{-2}	`S1.state=fs`
1.31×10^{-2}	2.59×10^{-2}	`LS1.state=fs`
1.31×10^{-2}	2.59×10^{-2}	`S2.state=fs`
1.31×10^{-2}	2.59×10^{-2}	`S3.state=fs`
1.31×10^{-2}	2.59×10^{-2}	`LS2.state=fs`

The minimal cutsets describing the conditions under which the system is failed dangerous are given in Table 8. They are encoded by a decision diagram made of 13 nodes.

Table 8. Minimal cutsets of the observer `failedDangerous`.

Probabilities		Minimal Cutsets
$t = 4380$	$t = 8760$	
2.26×10^{-9}	1.81×10^{-8}	`S1.state=fd, S2.state=fd, S3.state=fd`
3.45×10^{-6}	1.38×10^{-5}	`S1.state=fd, LS2.state=fd`
4.53×10^{-9}	3.61×10^{-8}	`LS1.state=fd, S2.state=fd, S3.state=fd`
6.89×10^{-6}	2.75×10^{-5}	`LS1.state=fd, LS2.state=fd`
2.18×10^{-8}	1.73×10^{-7}	`S2.state=fd, S3.state=fd, V1.state=fd`
3.31×10^{-5}	1.32×10^{-4}	`LS2.state=fd, V1.state=fd`
1.59×10^{-4}	6.29×10^{-4}	`V1.state=fd, V2.state=fd`

Note that none of these minimal cutsets involve a safe or undetected failure. This follows from the partial order over states.

Finally, the minimal cutsets describing the conditions under which the system is failed dangerous undetected are given in Table 9. They are encoded by a decision diagram made of 23 nodes.

Table 9. Minimal cutsets of the observer `failedUndetected`.

Probabilities		Minimal Cutsets
$t = 4380$	$t = 8760$	
2.26×10^{-9}	1.81×10^{-8}	`S1.state=fu, S2.state=fd, S3.state=fd`
3.45×10^{-6}	1.38×10^{-5}	`S1.state=fu, LS2.state=fd`
2.26×10^{-9}	1.81×10^{-8}	`S1.state=fd, S2.state=fu, S3.state=fd`
2.26×10^{-9}	1.81×10^{-8}	`S1.state=fd, S2.state=fd, S3.state=fu`
4.53×10^{-9}	3.61×10^{-8}	`LS1.state=fd, S2.state=fu, S3.state=fd`
4.53×10^{-9}	3.61×10^{-8}	`LS1.state=fd, S2.state=fd, S3.state=fu`
2.18×10^{-8}	1.73×10^{-7}	`S2.state=fu, S3.state=fd, V1.state=fd`
2.18×10^{-8}	1.73×10^{-7}	`S2.state=fd, S3.state=fu, V1.state=fd`
2.19×10^{-9}	1.75×10^{-8}	`S2.state=fd, S3.state=fd, V1.state=fu`
3.33×10^{-6}	1.33×10^{-5}	`LS2.state=fd, V1.state=fu`
1.60×10^{-5}	6.36×10^{-5}	`V1.state=fu, V2.state=fd`
1.60×10^{-5}	6.36×10^{-5}	`V1.state=fd, V2.state=fu`

8. Conclusions

In this article, we presented decision diagram algorithms to extract minimal cutsets of finite degradation models. These algorithms rely on three pillars:

- The concept of finite degradation structure, which is the core of combinatorial models used in reliability engineering.
- Two decomposition theorems, one for minimal cutsets and one for the "without" operator, which generalize those proposed by the first author for the Boolean case.
- The technology of decision diagrams, which generalizes Minato's zero-suppressed binary decision diagrams so to handle discrete variables and functions.

We implemented these algorithms into a prototype tool, which aims primarily at testing ideas. Nevertheless, experiments we have made on systems described in safety standards such as ISO/TR 12489 showed the interest and the feasibility of the approach.

Finite degradation models bring a significant additional expressive power compared to Boolean models. The key issue is the definition of suitable finite degradation structures, such as **W3F** we used to deal with the HIPPS TA4. This finite degradation structure makes actually possible to deal with a very large class of systems, typically those addressed by the IEC61508 standard. Other interesting finite degradation structures will probably emerge with experience.

As future work, we plan to perform a wide series of experiments and to lift-up to finite degradation models the calculation of risk indicators such as safety integrity levels or importance measures.

Author Contributions: This article results of a join work of the authors. Liu Yang is doing her PhD under the supervision of Antoine Rauzy.

Funding: This research received no external funding

Acknowledgments: Although this research received no external funding, Antoine Rauzy benefits since several years a strong support from the SAFRAN group, via the sponsoring of the chair Blériot-Fabre at CentraleSupélec (Paris, France).

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. S2ML + FDS Model for the Illustrative Use Case

We implemented the model for HIPPS TA4 in the S2ML+FDS domain specific modeling language. S2ML stands for system structure modeling language. S2ML is a versatile and complete set of constructs to structure models stemmed from object-oriented and prototype-oriented programming [35].

Languages of the S2ML+X family are the combination of S2ML with a mathematical framework (the X) to describe the behaviors of the systems under study. In our case, this mathematical framework consists of finite degradation models.

The model for the safety instrumented system pictured in Figure 1 can be splitted into three parts.

The first part, given in Figure A1 describes the domains (the finite degradation structures) as well as the operators that will be used.

```
domain W3F {w, fs, fd, fu} w<fs, w<fd, fd<fu;
domain WFsd {w, fs, fd} w<fs, w<fd;
domain Wfdu {w, fd, fu} w<fd, fd<fu;

operator Series arg1 arg2 =
    (if (or (eq arg1 w) (eq arg2 fs) (eq arg2 fd))
        arg2
        (if (or (eq arg1 fd) (eq arg1 fu))
            arg1
            (if (eq arg2 w) fs fd)));

operator Parallel arg1 arg2 =
    (if (or (eq arg1 fs) (eq arg2 fs))
        fs
        (if (or (eq arg1 w) (eq arg2 w))
            w
            (if (or (eq arg1 fu) (eq arg2 fu)) fu fd)));
```

Figure A1. The S2ML + FDS model for the TA4 system. Part 1: domains and operators.

Languages of the S2ML+X family use the infix form for expressions. For instance, the expression `(eq arg1 w)` denotes the equality `arg1 = w`. Operators `and`, `or` and `if` (if-then-else) have their usual meaning.

```
class AbstractComponent
    W3F state(init = w);
    W3F input, output(reset = w);
    assertion
        output := (Series input state);
end

class Sensor
    extends AbstractComponent;
    probability state fs = (exponentialDistribution lambdaFs (missionTime));
    probability state fd = (exponentialDistribution lambdaFd (missionTime));
    probability state fu = (exponentialDistribution lambdaFu (missionTime));
    parameter Real lambdaFs = 3.0e-6;
    parameter Real lambdaFd = 3.0e-7;
    parameter Real lambdaFu = 3.0e-7;
end

class LogicSolver
    extends AbstractComponent;
    WFsd state(init = w);
    probability state fs = (exponentialDistribution lambdaFs (missionTime));
    probability state fd = (exponentialDistribution lambdaFd (missionTime));
    parameter Real lambdaFs = 3.0e-6;
    parameter Real lambdaFd = 6.0e-7;
end

class IsolationValve
    extends AbstractComponent;
    WFdu state(init = w);
    probability state fd = (exponentialDistribution lambdaFd (missionTime));
    probability state fu = (exponentialDistribution lambdaFu (missionTime));
    parameter Real lambdaFd = 2.9e-6;
    parameter Real lambdaFu = 2.9e-7;
end
```

Figure A2. The S2ML + FDS model for the TA4 system. Part 2: classes for the different types of components.

The second part, given in Figure A2 implements the classes used to describe the different types of components, i.e., sensors, logic solvers and valves. All these classes derive from an abstract class `AbstractComponent` which can be seen as the implementation of a basic block of a block diagram. State variables are introduced by means of the attribute `init` (setting their initial or default value) while flow variables are introduced by means of the attribute `reset`.

Finally the third part, given in Figure A3 implements the description of the safety instrumented system itself.

```
block HIPPS
    Sensor S1;
    Sensor S2;
    Sensor S3;
    LogicSolver LS1;
    LogicSolver LS2;
    IsolationValve V1;
    IsolationValve V2;
    W3F state(reset = w);
    assertion
        S1.input := w;
        S2.input := w;
        S3.input := w;
        LS1.input := S1.output;
        LS2.input := (Parallel S2.output S3.output);
        V1.input := (Parallel LS1.output LS2.output);
        V2.input := LS2.output;
        state := (Parallel V1.output V2.output);
    observer failedSafe = state in {fs};
    observer failedDangerous = state in {fd, fu};
    observer failedUndetected = state in {fu};
end
```

Figure A3. The S2ML + FDS model for the TA4 system. Part 3: description of the HIPPS.

It declares first instances of classes `Sensor`, `LogicSolver` and `IsolationValve` as well as a variable `state` to represent the state of the system as a whole. Then, it connects the different components. Finally, it declares observers, which are the predicates for which minimal cutsets are calculated.

The above model illustrates the power of S2ML + FDS: domains and operators are easily implemented. Classes can be defined to describe generic components, and models are designed by assembling components in a straightforward way.

References

1. Rasmussen, N.C. *Reactor Safety Study. An Assessment of Accident Risks in U.S. Commercial Nuclear Power Plants*; Technical Report WASH 1400, NUREG-75/014; U.S. Nuclear Regulatory Commission: Rockville, MD, USA, 1975.
2. Rauzy, A. Notes on Computational Uncertainties in Probabilistic Risk/Safety Assessment. *Entropy* **2018**, *20*, 162. [CrossRef]
3. Andrews, J.D.; Moss, R.T. *Reliability and Risk Assessment*, 2nd ed.; ASM International: Materials Park, OH, USA, 2002.
4. Kumamoto, H.; Henley, E.J. *Probabilistic Risk Assessment and Management for Engineers and Scientists*; IEEE Press: Piscataway, NJ, USA, 1996.
5. Lisnianski, A.; Levitin, G. *Multi-State System Reliability*; Quality, Reliability and Engineering Statistics, World Scientific: London, UK, 2003.
6. Natvig, B. *Multistate Systems Reliability Theory with Applications*; Wiley: Hoboken, NJ, USA, 2010.
7. Zaitseva, E.; Levashenko, V. Reliability analysis of multi-state system with application of multiple-valued logic. *Int. J. Qual. Reliab. Manag.* **2017**, *34*, 862–878. [CrossRef]

8. Stewart, W.J. *Introduction to the Numerical Solution of Markov Chains*; Princeton University Press: Princeton, NJ, USA, 1994.
9. Ajmone-Marsan, M.; Balbo, G.; Conte, G.; Donatelli, S.; Franceschinis, G. *Modelling with Generalized Stochastic Petri Nets*; Wiley Series in Parallel Computing; John Wiley and Sons: New York, NY, USA, 1994.
10. Rauzy, A. Guarded Transition Systems: A new States/Events Formalism for Reliability Studies. *J. Risk Reliab.* **2008**, *222*, 495–505. [CrossRef]
11. Batteux, M.; Prosvirnova, T.; Rauzy, A. AltaRica 3.0 in 10 Modeling Patterns. *Int. J. Crit.-Comput.-Based Syst.* **2019**, *9*, 133–165. [CrossRef]
12. Rauzy, A.; Yang, L. Finite Degradation Structures. *J. Appl. Logics Ifcolog J. Logics Their Appl.* **2019**, *6*, 1471–1495.
13. Fussel, J.B.; Vesely, W.E. A New Methodology for Obtaining Cut Sets for Fault Trees. *Trans. Am. Nucl. Soc.* **1972**, *15*, 262–263.
14. Berg, U. *RISK SPECTRUM, Theory Manual*; RELCON Teknik AB: Sundbyberg, Sweden, 1994.
15. Rauzy, A. Anatomy of an Efficient Fault Tree Assessment Engine. In Proceedings of the International Joint Conference PSAM'11/ESREL'12, Helsinki, Finland, 25–29 June 2012; Virolainen, R., Ed.; Curran Associates, Inc.: Red Hook, NY, USA, 2012; pp. 3333–3343.
16. Rauzy, A. Mathematical Foundation of Minimal Cutsets. *IEEE Trans. Reliab.* **2001**, *50*, 389–396. [CrossRef]
17. Jung, W.S.; Han, S.H.; Ha, J. A fast BDD algorithm for large coherent fault trees analysis. *Reliab. Eng. Syst. Saf.* **2004**, *83*, 369–374. [CrossRef]
18. Xing, L.; Amari, S.V. *Binary Decision Diagrams and Extensions for System Reliability Analysis*; Scrivener Publishing, Wiley: Beverly, MA, USA, 2015.
19. Minato, S.I. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In Proceedings of the 30th ACM/IEEE Design Automation Conference, DAC'93, Dallas, TX, USA, 14–18 June 1993; pp. 272–277. [CrossRef]
20. International Organization for Standardization. *ISO/TR 12489:2013 Petroleum, Petrochemical and Natural Gas Industries—Reliability Modelling and Calculation of Safety Systems*; International Organization for Standardization: Geneva, Switzerland, 2013.
21. International Electrotechnical Commission. *International IEC Standard IEC61508—Functional Safety of Electrical/Electronic/Programmable Safety-Related Systems (E/E/PE, or E/E/PES)*; International Electrotechnical Commission: Geneva, Switzerland, 2010.
22. International Organization for Standardization. *ISO26262 Functional Safety—Road Vehicle*; International Standardization Organization: Geneva, Switzerland, 2012.
23. Awodey, S. Category Theory. In *Oxford Logic Guides*; Oxford University Press: Oxford, UK, 2010; Volume 52.
24. Davey, B.A.; Priestley, H.A. *Introduction to Lattices and Order*; Cambridge University Press: Cambridge, UK, 2002.
25. Morreale, E. Recursive Operators for Prime Implicant and Irredundant Normal Form Determination. *IEEE Trans. Comput.* **1970**, *C-19*, 504–509. [CrossRef]
26. 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, Orlando, FL, USA, 24–28 June 1990; pp. 40–45. [CrossRef]
27. Wegener, I. *Branching Programs and Binary Decision Diagrams—Theory and Applications*; SIAM Monographs on Discrete Mathematics and Applications, SIAM: Philadelphia, PA, USA, 2000.
28. Rauzy, A. Some Disturbing Facts about Depth First Left Most Variable Ordering Heuristics for Binary Decision Diagrams. *J. Risk Reliab.* **2008**, *222*, 573–582. [CrossRef]
29. Rudell, R.L. Dynamic Variable Ordering for Ordered Binary Decision Diagrams. In Proceedings of the IEEE International Conference on Computer Aided Design, Santa Clara, CA, USA, 7–11 November 1993; Lightner, M., Jess, J.A.G., Eds.; pp. 42–47.
30. Rauzy, A. BDD for Reliability Studies. In *Handbook of Performability Engineering*; Misra, K.B., Ed.; Elsevier: Amsterdam, The Netherlands, 2008; pp. 381–396.
31. Nusbaumer, O.; Rauzy, A. Fault Tree Linking versus Event Tree Linking Approaches: A Reasoned Comparison. *J. Risk Reliab.* **2013**, *227*, 315–326. [CrossRef]
32. Epstein, S.; Rauzy, A.; Wakefield, D. Can We Trust PRA: Take 3. In Proceedings of the 8th International Conference on Probabilistic Safety Assessment and Management, New Orleans, LO, USA, 14–18 May 2006.
33. Valiant, L.G. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* **1979**, *8*, 410–421. [CrossRef]

34. Viennot, X.G. Trees Everywhere. In Proceedings of the 15th Colloquium on Trees in Algebra and Programming, CAAP'90, Copenhagen, Denmark, 15–18 May 1990; Number 431 in LNCS; Springer: Berlin/Heidelberg, Germany, 1990; pp. 18–41.
35. Batteux, M.; Prosvirnova, T.; Rauzy, A. From Models of Structures to Structures of Models. In Proceedings of the IEEE International Symposium on Systems Engineering (ISSE 2018), Roma, Italy, 1–3 October 2018. [CrossRef]
36. Papadopoulos, Y.; Walker, M.; Parker, D.; Rüde, E.; Hamann, R.; Uhlig, A.; Grätz, U.; Liend, R. An approach to optimization of fault tolerant architectures using HiP-HOPS. *J. Eng. Fail. Anal.* **2011**, *18*, 590–608. [CrossRef]