

Uncertainty handling in quantitative Binary Decision Diagram (BDD)-based fault-tree analysis by interval computation

C. Jacob¹ J. Cardoso¹ D. Dubois²

¹DMIA department, Campus Supaéro
Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)

²ADRIA department
Institut de Recherche en Informatique de Toulouse (IRIT)

May 6th, 2011 / CS Seminars

Outline

- 1 Introduction
- 2 Project Goals
 - Operationability Vs. Safety
 - Safety Boolean methods
- 3 BDD and imprecise probabilities
 - Binary Decision Diagrams
 - Interval Analysis
- 4 Algorithm implementation
 - Algorithm
 - Example of application
- 5 Conclusion

Outline

- 1 Introduction
- 2 Project Goals
 - Operationability Vs. Safety
 - Safety Boolean methods
- 3 BDD and imprecise probabilities
 - Binary Decision Diagrams
 - Interval Analysis
- 4 Algorithm implementation
 - Algorithm
 - Example of application
- 5 Conclusion

Outline

- 1 Introduction
- 2 Project Goals
 - Operationability Vs. Safety
 - Safety Boolean methods
- 3 BDD and imprecise probabilities
 - Binary Decision Diagrams
 - Interval Analysis
- 4 Algorithm implementation
 - Algorithm
 - Example of application
- 5 Conclusion

Outline

- 1 Introduction
- 2 Project Goals
 - Operationability Vs. Safety
 - Safety Boolean methods
- 3 BDD and imprecise probabilities
 - Binary Decision Diagrams
 - Interval Analysis
- 4 Algorithm implementation
 - Algorithm
 - Example of application
- 5 Conclusion

Outline

- 1 Introduction
- 2 Project Goals
 - Operationability Vs. Safety
 - Safety Boolean methods
- 3 BDD and imprecise probabilities
 - Binary Decision Diagrams
 - Interval Analysis
- 4 Algorithm implementation
 - Algorithm
 - Example of application
- 5 Conclusion

Introduction

This PhD is a part of an Airbus project: @MOST. Our research group is constituted of researchers and PhD students from four French laboratories, IRIT, LAAS, ONERA and ISAE. The main objectives are

- To ensure the **safety** of the aircraft.
- To ensure the **operationability** of the aircraft.
- To optimize the maintenance schedule.

At present, **safety analysis** are carried out by means of **fault-tree analysis** from the models of the system under study. We are interested in the method of probability evaluation of fault-trees through **Binary Decision Diagrams** (BDDs).

- Extend the safety models to operationability models
- Make the difference between:
 - Variability (probabilities)
 - Lack of information (intervals, sets...)

Operationability Vs. Safety

	Safety model	Operationability model
Goal	Certification	Decision making
Queries	Safety (Probability of a catastrophic event $< 10^{-9}$)	Safety Operationability (Any component C important for mission should be available at a time t) Mission (The aircraft is able to make the next flight)

Safety Boolean methods

One of the objectives of safety analysis is to evaluate the probabilities of dreadful events.

Analytical approach

Dreadful event is described as a **Boolean function F** of atomic events. Its probability is computed from the knowledge of the probabilities of atomic events.

In a Boolean model of a system:

- **Variable** represents the **state** of an elementary component (Ok, failed, ...)
- **Boolean formulas** describe the **failures of the system** as function of those variables.

Fault-tree

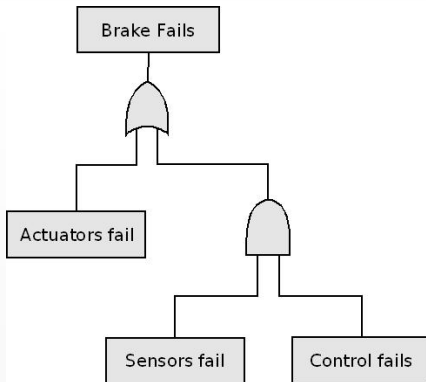


Figure: $F = A \vee (S \wedge C)$

Mathematical problem

A set of propositional variables v_1, \dots, v_n .

Probabilities p_1, \dots, p_n attached to these variables.

Given a formula F expressed in terms of v_1, \dots, v_n , via \vee (or), \wedge (and), and \neg (negation), find (bounds for) $P(F)$ under various assumptions.

- Variables are **independent** and the p_i 's are **known**: then $P(F)$ is precise, and methods exist.
- **Unknown dependencies** of variables and the p_i 's are **known**: then find optimal bounds for $P(F)$. (Fréchet bounds: $\max(0, P(a) + P(b) - 1) \leq P(a, b) \leq \min(P(a), P(b))$)

Mathematical problem

- Variables are **independent** and the p_i 's are only known to live in **intervals** $[a_i, b_i]$: then find optimal bounds for $P(F)$.
- The two previous cases together
- Dependencies between variables are known via a **Bayesian networks** with imprecise probabilities on atoms and ill-known conditional probabilities.
- More general queries than computing $P(F)$ (For example verify that $P(F)$ is under a certain value, ...)

Shannon Decomposition

- Let us consider a Boolean function F on a set of variables X , and $A \in X$. The **Shannon decomposition** of F related to A is given by:

$$F = (A \wedge F_{A=1}) \vee (\neg A \wedge F_{A=0}) \quad (1)$$

- Recursive application of Shannon decomposition for each variable of a function F gives a binary tree: the **Shannon tree**

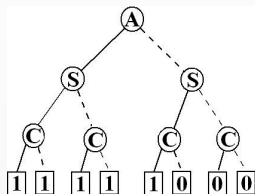


Figure: Shannon tree of $F = A \vee (S \wedge C)$ with order $A > S > C$

Shannon tree

- Each **node** of this tree is an *if-then-else (ite) operator*: contains **one variable** A_i and has **two edges**.
- One **edge** points towards the **positive cofactor** $F_{A=1}$ and the other one towards the **negative cofactor** $F_{A=0}$.
- The **leaves** of the tree are the **truth values** 0 or 1 of the formula.
- Expression obtained by the **conjunction** of variables going down from the root to a leaf 1 is a **minterm**, and the **disjunction** of all those minterms gives the **formula**.

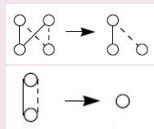
This representation of a formula is exponential in memory space: that is the reason why **Binary decision diagrams** are introduced.

Binary Decision Diagrams

Binary decision diagrams

They reduce the size of the Shannon tree by means of two reduction rules:

- **R1:** Isomorphic subtrees are merged into a single one.
- **R2:** A node, both outward edges of which point to the same node, is useless and should be deleted.



When we apply those two rules as often as necessary, we obtain the BDD associated to the formula.

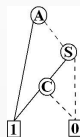


Figure: BDD of $F = A \vee (S \wedge C)$

Binary Decision Diagrams

- A BDD is a **directed graph** without cycles.
- For a given order of variables, it is **unique** up to an isomorphism.
- The size of the BDD is directly linked to the order of the variables: some heuristics can be used to optimize it.
- Each **path** from the root to terminal node 1 can be seen as a **conjunction** of literals, and the **disjunction** of all those paths gives a representation of the **formula**. A variable can be either *present* (in a positive or negative polarity) in a path, or *absent*:
 - A variable is **present** in a positive polarity if the path contains the **then-edge** of a node labeled by this variable.
 - A variable is **present** in a negative polarity if the path contains the **else-edge** of a node labeled by this variable
 - A variable is **absent** if the path doesn't contain a node labeled by this variable.

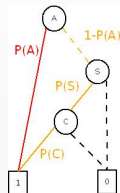
Probability computation from BDD

- Computation from Shannon decomposition:
 $F_{A=0}$ and $F_{A=1}$ are mutually exclusive, so the probability of the formula F is:

$$P(F) = (1 - P(A)) \times P(F_{A=0}) + P(A) \times P(F_{A=1}) \quad (2)$$

- Recursive application** of the formula on each variable gives probability of $P(F)$ from probabilities of its atoms:

$$P(F) = \sum_{p \in \mathcal{P}} \left[\prod_{A_i \in \mathcal{A}_p^+} P(A_i) \prod_{A_j \in \mathcal{A}_p^-} (1 - P(A_j)) \right] \quad (3)$$



Basics about Interval Analysis

- Developed by mathematicians since the 1950s and 1960s as an approach to put **bounds** on rounding or measurement errors in mathematical computation.
- Can also be used to represent some **lack of information**.

Principle

Use an interval instead of a precise value for variables of some function: find the range \bar{f} and \underline{f} of a function f of n variables $\{x_1, \dots, x_n\}$, knowing the intervals containing the variables:
 $x_1 \in [\underline{x}_1, \overline{x}_1], \dots, x_n \in [\underline{x}_n, \overline{x}_n]$

Interval arithmetic

- Basic operations of interval arithmetic generally used are: (for two intervals $[a,b]$ and $[c,d]$ subsets of \mathbb{R})

$$\textit{Addition} : [a, b] + [c, d] = [a + c, b + d] \quad (4)$$

$$\textit{Subtraction} : [a, b] - [c, d] = [a - d, b - c] \quad (5)$$

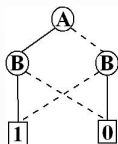
$$\textit{Multiplication} : [a, b] \times [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \quad (6)$$

$$\textit{Division} : \frac{[a, b]}{[c, d]} = [\min(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d}), \max(\frac{a}{c}, \frac{a}{d}, \frac{b}{c}, \frac{b}{d})] \quad (7)$$

- The major limitation in the application of this the **naive interval arithmetic**: the **dependency problem**. It comes from the **repetition of the same variable** in the expression of a function \Rightarrow difficulties for exact range computation.

Illustration of dependency problem

- Illustration of this problem: we consider the formula of the *equivalence* $F = A \Leftrightarrow B = (A \wedge B) \vee (\neg A \wedge \neg B)$



- Compute the range $[f, \bar{f}]$ of $P(F)$.
We know: $P(A) = a \in [\underline{a}, \bar{a}]$ and $P(B) = b \in [\underline{b}, \bar{b}]$, with $[\underline{a}, \bar{a}]$ and $[\underline{b}, \bar{b}]$ includes in $[0, 1]$.

Computation with Interval Arithmetic

- $P(F) = a \times b + (1 - a)(1 - b)$
 By applying **naive interval arithmetic** will get:
 $P(F) \in [\underline{ab}, \overline{ab}] + (1 - [\underline{a}, \overline{a}])(1 - [\underline{b}, \overline{b}])$
 $\Leftrightarrow P(F) \in [\underline{ab}, \overline{ab}] + [1 - \overline{a}, 1 - \underline{a}] \times [1 - \overline{b}, 1 - \underline{b}]$
 $\Leftrightarrow P(F) \in [\underline{ab} + (1 - \overline{a})(1 - \overline{b}), \overline{ab} + (1 - \underline{a})(1 - \underline{b})]$
- Numerical application:** $[\underline{a}, \overline{a}] = [0.3, 0.8]$ and $[\underline{b}, \overline{b}] = [0.4, 0.6]$
 give that $P(F) \in [0.2, 0.9]$.
- This result is wrong because we use two different values of a same variable: \underline{a} and \overline{a} for a , \underline{b} and \overline{b} for b .
 ($P(F) = \underline{ab} + (1 - \overline{a})(1 - \overline{b})$ for example).

Interval analysis

- For a **continuous** function f , with $x_i \in [\underline{x}_i, \bar{x}_i]$, the image of a set of intervals $f([\underline{x}_1, \bar{x}_1], \dots, [\underline{x}_n, \bar{x}_n])$ is an **interval**.
- One way to solve the dependency problem is to **factorize** $f(x_1, \dots, x_n)$ in such a way that variables appear only once in the expression.
- Unfortunately, it is not always possible, hence we must analyze the **monotonicity** of the function.
- A function f is said to be **locally monotonic** if the function obtained by fixing all variables x_i but one is monotonic with respect to the remaining variable $x_j, j \neq i$.
- The extrema of a **locally monotonic** function f are attained at the **boundaries of the domain** of x_j (i.e. 2 possibilities, \underline{x}_j and \bar{x}_j , for each x_j).

Monotonicity of Boolean formula

- If a variable A appears **only once** in a Boolean function F , its monotonicity with respect to A is known:
 - it will increase if A appears in a **positive** way \Rightarrow we use \underline{A} for \underline{F} and \overline{A} for \overline{F} .
 - it will decrease if it appears in a **negative** way ($\neg A$) \Rightarrow we use \overline{A} for \underline{F} and \underline{A} for \overline{F} .
- More generally, if a variable A appears several times, but **only as a positive** (resp. **negative**) literal, then F is **increasing** (resp. **decreasing**) with respect to A .

Monotonicity of the probability of a BDD representation

- Probability of a Boolean formula is **locally monotonic**, it can be easily proved by using the formula of Shannon decomposition. Formula (2) can be written as:

$$P(F) = P(F_{A=0}) + P(A) \times [P(F_{A=1}) - P(F_{A=0})] \quad (8)$$

- Hence, the formula of probability computation from a BDD is **locally monotonic**.

$$P(F) = \sum_{p \in \mathcal{P}} \left[\prod_{A_i \in \mathcal{A}_p^+} P(A_i) \prod_{A_j \in \mathcal{A}_p^-} (1 - P(A_j)) \right] \quad (9)$$

About non-monotonic functions

- Given n intervals $[\underline{x}_i, \bar{x}_i]$, $i = 1..n$, the n -uple of values in the set $\mathcal{X} = \times_i \{\underline{x}_i, \bar{x}_i\}$ is called a **configuration**. The extrema configurations z_j , $j = 1..2^n$, obtained by selecting for each value one interval end, define the set $\mathcal{H} = \times_i \{\underline{x}_i, \bar{x}_i\}$, where z_j has the form $(x_1^{c_1}, \dots, x_n^{c_n})$, $c_n \in \{0, 1\}$ with $x_i^0 = \underline{x}_i$ and $x_i^1 = \bar{x}_i$ and $|\mathcal{H}| = 2^n$.
- The range $[\underline{f}, \bar{f}]$ of a locally monotonic function $f([\underline{x}_1, \bar{x}_1], \dots, [\underline{x}_n, \bar{x}_n])$ can be obtained by testing the **2^n extremal values** of the x_i 's and taking the extremal values: $[\underline{f}, \bar{f}] = [\min\{f(z_j)\}, \max\{f(z_j)\}]$.

Example of computation by exploring all configurations

Let us go back to our previous example of equivalence \Leftrightarrow :

$$P(F) = a \times b + (1 - a)(1 - b)$$

- The function is not monotonic but locally monotonic so we can compute the exact range by exploring the 2^2 configurations:
 - $z_1 : (\underline{a}, \underline{b}), P_{z_1}(F) = \underline{a} \times \underline{b} + (1 - \underline{a})(1 - \underline{b}) = 0.3 \times 0.4 + 0.7 \times 0.6 = 0.54$
 - $z_2 : (\underline{a}, \overline{b}), P_{z_2}(F) = \underline{a} \times \overline{b} + (1 - \underline{a})(1 - \overline{b}) = 0.3 \times 0.6 + 0.7 \times 0.4 = 0.46$
 - $z_3 : (\overline{a}, \underline{b}), P_{z_3}(F) = \overline{a} \times \underline{b} + (1 - \overline{a})(1 - \underline{b}) = 0.8 \times 0.4 + 0.2 \times 0.6 = 0.44$
 - $z_4 : (\overline{a}, \overline{b}), P_{z_4}(F) = \overline{a} \times \overline{b} + (1 - \overline{a})(1 - \overline{b}) = 0.8 \times 0.6 + 0.2 \times 0.4 = 0.56$
- The extrema are obtained for the configurations z_3 and z_4 so result is $P(F) \in [0.44, 0.56]$: much tighter than the previous one.

Algorithm

- This algorithm is able to compute the exact bounds of the probability of a Boolean function, given the interval ranges of its atomic probabilities.

Algorithm attributes

Input: Boolean formula of a dreadful event F

Output: Range of $P(F)$, i.e. $[P_F.lb, P_F.up]$

Each variable v of F is characterized by:

- its probability interval $[v.lb, v.up]$ (the lower/upper bounds)
- its value in a path: $path.value$ (=0 if the variable appears as a positive literal in this path, =1 otherwise)
- its *type* $\in \{Cat1, Cat2, Cat3\}$

Algorithm

Three main steps:

- 1
 - Parsing of the Fault-tree in *Aralia* format to get Boolean formula F .
 - Compute the associated BDD with *CUDD* and *BuDDy* libraries.
- 2
 - Split the variables into three categories:
 - **Cat1**: Variables that only appear positively in the Boolean formula, i.e. the formula increase with respect to them.
 - **Cat2**: Variables that only appear negatively in the Boolean formula, i.e. the formula decrease with respect to them.
 - **Cat3**: Variables that are present in the Boolean formula along with their negation, i.e. we don't know the monotonicity.
- 3
 - Determine the probability $P(F)$ of all configurations z_j .
 - Take the extrema of those probabilities, hence get our final result $[P_F.lb, P_F.up]$.

Algorithm

Three main steps:

- 1
 - Parsing of the Fault-tree in *Aralia* format to get Boolean formula F .
 - Compute the associated BDD with *CUDD* and *BuDDy* libraries.
- 2
 - Split the variables into three categories:
 - **Cat1**: Variables that only appear positively in the Boolean formula, i.e. the formula increase with respect to them.
 - **Cat2**: Variables that only appear negatively in the Boolean formula, i.e. the formula decrease with respect to them.
 - **Cat3**: Variables that are present in the Boolean formula along with their negation, i.e. we don't know the monotonicity.
- 3
 - Determine the probability $P(F)$ of all configurations z_i .
 - Take the extrema of those probabilities, hence get our final result $[P_F.lb, P_F.up]$.

Algorithm

Three main steps:

- 1
 - Parsing of the Fault-tree in *Aralia* format to get Boolean formula F .
 - Compute the associated BDD with *CUDD* and *BuDDy* libraries.
- 2
 - Split the variables into three categories:
 - **Cat1**: Variables that only appear positively in the Boolean formula, i.e. the formula increase with respect to them.
 - **Cat2**: Variables that only appear negatively in the Boolean formula, i.e. the formula decrease with respect to them.
 - **Cat3**: Variables that are present in the Boolean formula along with their negation, i.e. we don't know the monotonicity.
- 3
 - Determine the probability $P(F)$ of all configurations z_i .
 - Take the extrema of those probabilities, hence get our final result $[P_F.lb, P_F.up]$.

Algorithm

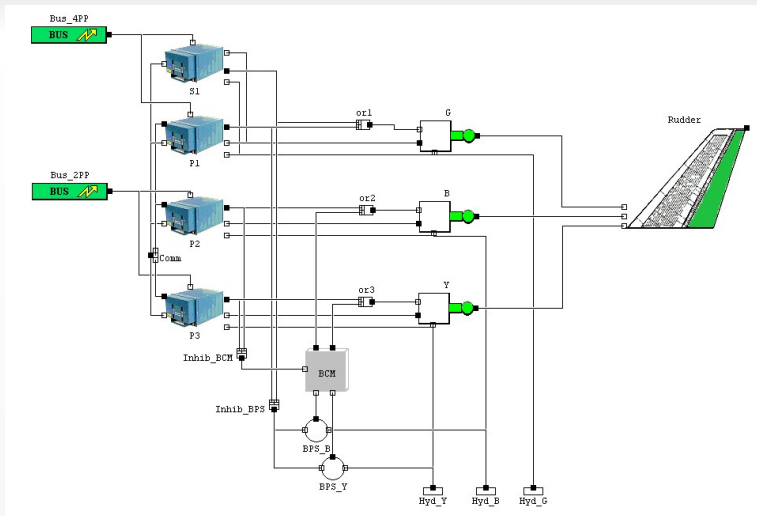
Algorithm 1 Interval Analysis for BDD

- 1: Parse input file with formula F
 - 2: Create a list V of variables of F with attributes lb , ub (lower/upper bound), $type$, $path.value$
 - 3: **Step1:**
 - 4: $Pa = \text{BuDDy}(\text{CUDD}(\text{formula } F))$ {create a BDD from F using the format of eq. ??}
 - 5: $\mathcal{P} = \text{set of paths of } Pa$
 - 6: **Step2: Split the variables in 3 categories**
 - 7: **SplitVariables()**
 - 8: **Step3: Compute the interval [Pmin,Pmax] of $P(Pa)$**
 - 9: **Product1_2InPath()**
 - 10: **for each $i = 1$ to $2^{\text{cat}3}$ do**
 - 11: **Combination1(i)**
 - 12: $P_{\min} = \min(P_{\min}, [z_i].lb)$
 - 13: $P_{\max} = \max(P_{\max}, [z_i].ub)$
 - 14: **end for**
 - 15: Store the interval $[P_F.lb, P_F.ub]$ of $P(Pa)$, $P_F.lb = P_{\min}$ and $P_F.ub = P_{\max}$ in the output file
-

Algorithm

-
- 1: **SplitVariables()** {Split the variables in 3 categories}
 - 2: **for** each variable $v \in V$ **do**
 - 3: set $v.type = 0, 1$ or 2 { v is Cat1, Cat2 or Cat3}
 - 4: $cat3 = cat3 + 1$ {count number of variables Cat3 in the BDD}
 - 5: **end for**
 - 6:
 - 7: **Product1_2InPath()** {Compute for each path the products that are constant (Cat1, Cat2)}
 - 8:
 - 9: **Combinationl(i)** {Compute each element of the vector of different combinations $[z_i]$, $i \in \{1, \dots, 2^{cat3}\}$, and for each combination, compute the value of the probabilities by summing probabilities of all paths. For one combination, A=a same value of bound is used for all paths of the BDD.}
-

Example of application on safety model



Example of application on safety model

- Boolean formula of the Fault-tree in *Aralia* format

```

g.1 := ('B.loss' & 'Bus_4PP.loss' & 'Hyd_Y.loss');
g.2 := ('B.loss' & 'Bus_4PP.loss' & 'Y.loss');
g.3 := ('B.loss' & 'G.loss' & 'Hyd_Y.loss');
g.4 := ('B.loss' & 'G.loss' & 'Y.loss');
g.5 := ('B.loss' & 'Hyd_G.loss' & 'Hyd_Y.loss');
g.6 := ('B.loss' & 'Hyd_Y.loss' & 'P1.loss' & 'S1.active_failure');
g.7 := ('P1.loss' & 'P2.loss' & 'S1.hidden_failure' & 'Y.loss');
g.8 := ('BCM.active_failure' & 'Bus_2PP.loss' & 'P1.loss' & 'S1.hidden_failure');
g.9 := ('BCM.hidden_failure' & 'Bus_2PP.loss' & 'Hyd_G.loss');
g.10 := ('Bus_2PP.loss' & 'Bus_4PP.loss' & 'Y.loss');
g.11 := ('Bus_2PP.loss' & 'G.loss' & 'Y.loss');
g.12 := ('Bus_2PP.loss' & 'Hyd_G.loss' & 'Y.loss');
g.13 := ('Bus_4PP.loss' & 'Hyd_B.loss' & 'Hyd_Y.loss');
g.14 := ('Bus_4PP.loss' & 'Hyd_B.loss' & 'Y.loss');
g.15 := ('Bus_4PP.loss' & 'P2.loss' & 'Y.loss');
    
```

```

'F' := g.1 | g.2 | g.3 | g.4 | g.5 | g.6 | g.7 | g.8 | g.9 | g.10 | g.11 | g.12 | g.13 | g.14 |
g.15
    
```

Example of application on safety model

- BDD paths obtained with *BuDDy* library:

$p_1 = \langle B.\text{loss}:0, \text{Bus_4PP}.\text{loss}:0, \text{Hyd_Y}.\text{loss}:0, Y.\text{loss}:0, P1.\text{loss}:1, S1.\text{active_failure}:1, P2.\text{loss}:0, \text{BCM}.\text{active_failure}:1 \rangle$, $p_2 = \langle B.\text{loss}:1, \text{Bus_4PP}.\text{loss}:0, \text{Hyd_Y}.\text{loss}:1, Y.\text{loss}:1, \text{Hyd_G}.\text{loss}:0, P1.\text{loss}:1 \rangle$, $p_3 = \langle \text{Bus_4PP}.\text{loss}:1, \text{Hyd_Y}.\text{loss}:1, B.\text{loss}:1, Y.\text{loss}:0, Y.\text{loss}:1, P1.\text{loss}:0, P1.\text{loss}:0, \text{BCM}.\text{active_failure}:1 \rangle$

- The probability value of some variables are known:
 - $P(\text{Hyd}_i.\text{loss}) = e^{-4}$
 - $P(\text{Bus}_i\text{PP}) = e^{-3}$
- For some others, only the interval containing the probability values is known:
 - $i.[\text{lb}, \text{ub}] = [0.15, 0.25], i \in \{Y, B, G\}$
 - $P_i[\text{lb}, \text{ub}] = [0, e^{-2}], i = 1..3$
 - $S1.\text{Active_failure}[\text{lb}, \text{ub}] = [0.1, 0.4]$
 - $\text{BCM}.\text{Active_failure}[\text{lb}, \text{up}] = [0.15, 0.345]$

Comparison of results

For the event *Loss of the Rudder* represented by F

- Using our algorithm we obtain the interval $I_1 = [0.00676587, 0.027529]$
- Using naive interval arithmetic and not taking into account logical dependencies we obtain $I_2 = [0.0309801, 0.232487]$

It can be noticed that the interval I_1 is tighter than I_2 ($I_2.lb = 4.58 * I_1.lb$ and $I_2.ub = 8.45 * I_1.ub$).

Conclusion

- Generalization of the Interval analysis to **fuzzy sets**, with α -cuts.
- Use of **reliability laws** with imprecise failure rates, especially exponential laws, to generate imprecise probabilities across time.
- Study for the case of very small probabilities.
- Extend the work by dropping the **independence assumption**.