Geneva University

Advanced Formal Tools

14X007

Probabilistic Model Checking with PRISM
Production Line

Patrick Sardinha

Spring 2021

1 Introduction

Dans le cadre de ce ce projet, nous voulons étudier une chaîne de production en passant par le vérificateur de modèle probabiliste PRISM. La chaîne de production étudiée se base sur le jeu Factorio [1], où le but est de construire et d'entretenir des chaînes de production automatisées. A partir de là, nous avons décidé de construire un modèle basé sur la propagation des erreurs et notamment utilisé par les arbres de défaillances pour étudier des propriétés sur notre chaîne de production tel que typiquement calculer la probabilité qu'une erreur se produisant dans une machine va se propager jusqu'à impliquer une panne globale du système avant un certain temps. La structure du papier va ainsi suivre le plan suivant : nous aborderons dans la section 2, les travaux en lien avec notre sujet. Nous verrons dans la section 3, la méthodologie proposée puis l'implémentation de celle-ci dans la section 4. Dans la section 5, nous verrons les résultats obtenus et évaluerons la méthode proposée. Enfin, dans la section 6, nous évoquerons les conclusions de ce travail et de possibles travaux futurs.

2 Related work

Les arbres de défaillances ou arbres de pannes (en anglais *fault trees* (FT)) représentent graphiquement les combinaisons possibles d'événements, événements de bas niveaux dans l'arbre, menant à un événement indésirable prédéfini, l'événement de niveau supérieur de l'arbre. Schématiquement, un arbre de défaillance est proposé en Figure 1 où les événement bas niveau sont représentés en jaune (LLE) alors que l'événement au niveau de l'arbre est lui indiqué en rouge (TLE). Ces arbres sont construit avec différents types d'événement et de portes logiques. Les principaux événements utilisés dans ces arbres sont les *basic event* représentant des échecs dans des composants du système ainsi que les *intermediate event*, ceux situés à la sortie des portes. Ces dernières permettent de décrire la relation entre les événements d'entrée et de sortie liés à une même porte. Deux types de portes logiques sont ainsi utilisées : *OR* où la sortie se produit si l'une des entrée se produit ainsi que *AND* nécessitant que les deux entrées se produisent pour avoir la sortie.

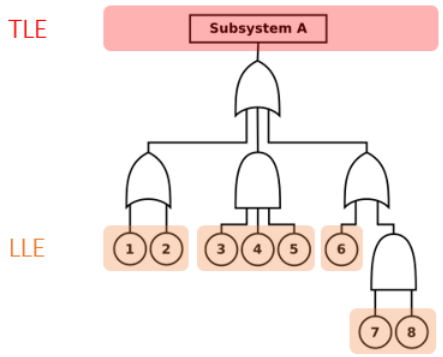

FIGURE 1: Représentation des différents éléments d'une arbre de défaillances

L'analyses des arbres de défaillances (en anglais *Fault Tree Analysis (FTA)*) est une méthode permettant d'évaluer les risques du système relatif au FT en question. Cette méthode permet notamment de comprendre comment un système peut tomber en panne, savoir comment les risques peut être réduit voir supprimés ainsi qu'obtenir les taux de panne du système. Cette analyse est souvent réalisée en transformant l'arbre de défaillances en une fonction booléenne, elle-même utilisée pour la simulation mais une nouvelle méthodologie probabiliste pour l'analyse des arbres de défaillances à été proposé par les auteurs Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed et Yvon Savaria dnas le papier de recherche intitulé "Efficient Probabilistic Fault Tree Analysis

of Safety Critical Systems via Probabilistic Model Checking" [2].

La méthodologie suit les cinq étapes principales. Premièrement, il faut modéliser le système c'est-à-dire la composition de composants et surtout spécifier les paramètres liés aux différents événements. Ensuite, l'idée est de synthétiser l'arbre de défaillance du système puis de modéliser le comportement de chaque porte de l'arbre comme un automate probabiliste (PA). Une fois cette étape effectuée, il faut générer un modèle formel probabiliste de l'arbre de défaillances en combinant les PA et à l'aide d'un vérificateur de modèle probabiliste tel que PRISM. L'étape finale, est d'analyser ce modèle pour évaluer la probabilité maximale qu'un événement atteigne le haut de l'arbre (TLE).

Cette nouvelle méthode à pour avantage de limiter les contraintes au niveau temps/ressources rencontrées avec l'autre méthodes. Il est ainsi possible d'analyser des arbres de défaillances plus rapidement et surtout avec un plus grand nombre d'états.

Comme mentionné auparavant, la nouvelle méthode proposée pour la FTA passe par un vérificateur de modèle probabiliste. Nous allons ainsi utiliser PRISM [3], un outil logiciel de vérification formel pour la modélisation et l'analyse de systèmes qui présentent un comportement probabiliste [4].

3 Model

Comme évoqué auparavant, notre but est d'utiliser le modèle de la propagation des fautes avec les arbres de défaillances sur un système de chaîne de production. Cette chaîne de production est ainsi composée de machines qui extractent les ressources et de machines qui les transforme. De plus, sur chacune des machines constituant la chaîne, il est possible que des événements perturbateurs les affectes selon une certaine probabilité. Pour notre cas pratique, nous avons deux machines d'extraction : la foreuse thermique et la pompe offshore. La première, a besoin d'une source de minerai à extraire (fer ou charbon) pour les fournir à la chaîne de production alors que la pompe nécessite quant à elle une source d'eau. Ces deux machines peuvent être affectée par différents événements telle qu'une panne technique ou un manque de ressources première (nous minimisons volontairement le nombre d'événements pouvant affecter une machine pour éviter une explosion des états du modèle). Nous avons ensuite, quatre machines de transformation : la chaudière prenant du charbon et de l'eau en entrée pour produire de la vapeur, la machine à vapeur transformant la vapeur d'eau en électricité, le four qui produit des minéraux transformés à partir des minéraux brutes et d'un combustible (charbon) et finalement, la machine d'assemblage qui prend en entrée les minéraux transformés et fonctionne grâce à une source électrique pour produire le produit final de la chaîne. Toutes ses chaîne sont aussi susceptibles d'être affectée par des événements perturbateurs dont les principaux sont typiquement, une panne de la machine ou un manque d'approvisionnement des ressources d'entrées.

En assemblant ces différentes machines, nous avons donc construit une chaîne de production typique du jeu Factorio dans le but d'appliquer notre modèle sur celle-ci et d'observer des propriétés voulues. Notre chaîne de production est ainsi composée de huit machines dont quatre foreuses, un four, une chaudière, une machine à vapeur et une machine d'assemblage. Cette chaîne de production est représentée schématiquement en Figure 2.

Le modèle de la propagation des erreurs dans les arbres de défaillances nous permet ainsi d'estimer et de comparer les probabilités que des pannes provenant de différents événements bas niveau de l'arbre provoquent une panne globale du système du à la propagation de la faute jusqu'au TLE et il est ainsi possible, d'identifier le ou les composants les plus critiques de notre chaîne de production. Une possible application proposée dans la littérature à partir des composants critiques d'un système est d'ajouter un principe de redondance à ceux-ci afin de réduire les pannes du systèmes.

L'arbre de défaillances relatif à notre chaîne de production est représenté en Figure 3. Les événements situés en bas de l'arbre (cercles verts) représente les fautes pouvant provoquées une panne du système. Les rectangles jaunes indiquent les événements intermédiaires et sont situés sur les sorties des portes. Finalement l'événement situé en haut de l'arbre "système failure" est le TLE et

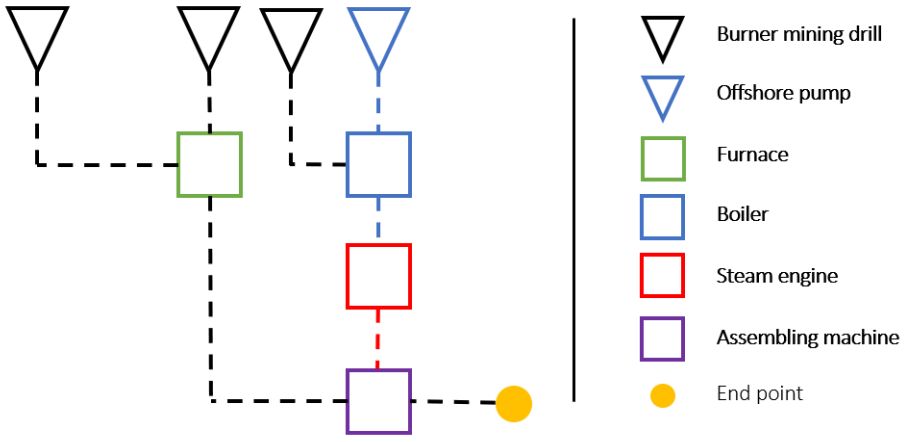

FIGURE 2: Schéma de la chaîne de production

implique une panne globale du système si une faute se propage jusqu'à lui.

Avec l'utilisation des arbres de défaillances, nous ajoutons deux mécanismes supplémentaires. Le *fault propagation* ainsi que le*fault masking*. La propagation d'une erreur est le fait qu'une erreur dans un noeud d'une couche inférieure de l'arbre va se propager avec une certaine probabilité à la couche supérieure alors que le masquage d'erreur va au contraire impliquer une atténuation de l'erreur avec une certaine probabilité et donc stopper la propagation de la faute. Ce mécanisme de masquage peut être perçu comme le fait qu'une machine peut d'elle-même réparer la faute ou alors que la faute n'était finalement pas assez important pour se propager. Le masquage de faute agit donc de la même manière que la redondance pouvant être ajoutée à une machine. Ces deux mécanismes s'appliquent à toutes les portes de l'arbre de défaillance représentant notre chaîne de production et pour simplification, toutes les portes auront les mêmes probabilités pour la propagation et le masquage des erreurs. En Figure 4 et 5, nous schématiquement le principe de ces deux mécanismes.

Le comportement des différentes portes (OR et AND) d'un arbre de défaillances est modélisé sous forme d'automate probabiliste. En Figure 6, nous représentons le comportement d'une porte OR prenant deux entrées. Dans l'état S_0, un des deux événements est déclenché menant à l'état S_1 ou S_2. A partir des ces deux états, il est possible de masquer la faute avec une certaine probabilité menant à l'état S_4 ou alors, avec une probabilité de $1 - P_{masking}$, de propager la faute à travers la porte et atteindre l'état S_3.

Pour la porte logique AND, représenté en Figure 7, le comportement est similaire cependant, pour propager la faute à l'état S_3, il est nécessaire dans les états S_1 et S_2 que deuxième l'événement lié à la porte soit aussi déclenché.

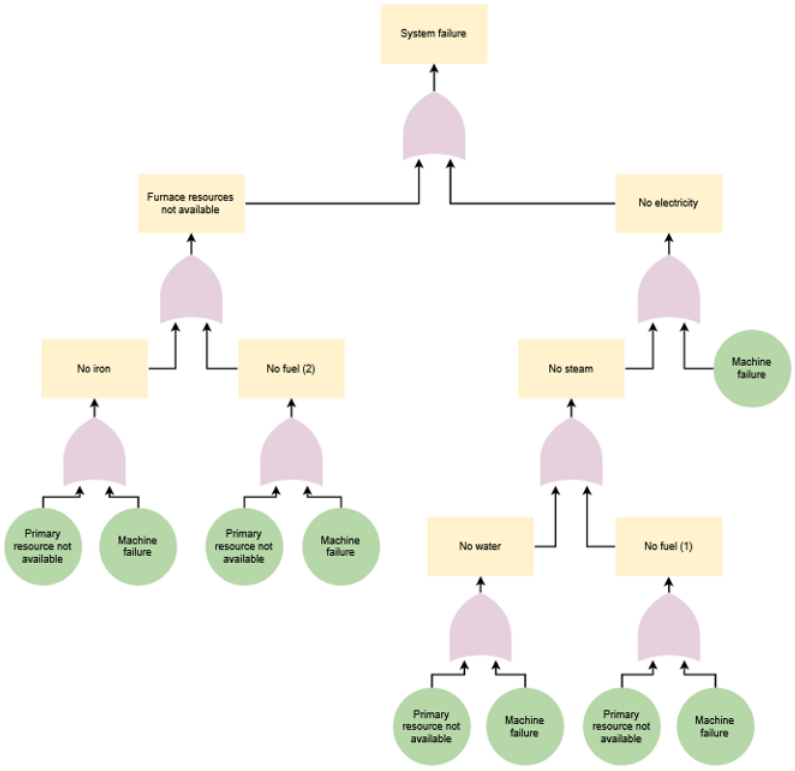

FIGURE 3: Arbre de défaillance de la chaîne de production

Avec le comportement de ces deux types de porte, il est ainsi possible de représenter sous forme d'automate probabiliste, le comportement d'une combinaison de portes et donc d'un arbre de défaillances plus complexe. Notre chaîne de production est ainsi représenté dans la Figure 8.

Nous pouvons maintenant définir des propriétés intéressantes à observer sur notre modèle. L'une d'entre elles, est de pouvoir estimer et comparer la probabilité qu'une faute se produisant dans un événement bas niveau de l'arbre puisse causer une panne globale du système avant un certain temps, par exemple 10 unités de temps. Cette propriété pourra par la suite est traduite sur PRISM (formule PCTL) de la manière suivante : $P_{max} =?[(FX_i = 1)\&(FTLE = 1)\&(FT < X_t)]$ signifiant qu'on cherche la probabilité maximum qu'un événement X_i soit déclenché ($X_i = 1$), que la faute se propage jusqu'en haut de l'arbre ($TLE = 1$) et tout ça avant X_t unités de temps ($T < X_t$).

Pour pouvoir évaluer cette propriété, il est nécessaire de spécifier la façon dont les portes de l'arbre de défaillances doivent être visitées. Premièrement, quelque soit le type de porte (OR ou AND), l'idée est d'évaluer toutes les en-

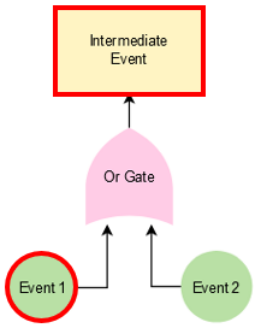

FIGURE 4: Propagation d'une erreur sur une porte OR

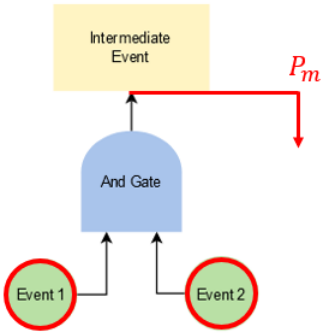

FIGURE 5: Masquage d'une erreur sur une porte AND

trées d'une même porte avant de passer à une autre. Ensuite, les portes de l'arbre de défaillances doivent être évaluées couche par couche et permet ainsi d'assurer pour une porte sur une couche supérieure que toutes ses entrées ont été évalué au préalable. L'ordre dans l'évaluation des portes sur une même couche de l'arbre n'importe pas et peut donc se faire de n'importe quelle manière.

Pour représenter le temps de fonctionnement de notre chaîne de production, nous ajoutons un système de "poids" sur chacune des machines composant la chaîne. Ce poids indique donc le temps nécessaire à une machine pour effectuer son travail. A partir de là, il est impératif de prendre en compte le mécanisme le plus important d'une chaîne de production : le parallélisme entre plusieurs machines. Ce comportement est schématisé en Figure 9 avec un système composé de trois machines.

La première machine effectue son travail en 2 unités de temps (UT), alors que les deux autres mettent 1 unité de temps. Le temps mis par la première machine pour propager une erreur va donc être au minimum de 2, puis va à nouveau être incrémenté de 2 à fois qu'une faute est masquée dans cette machine,

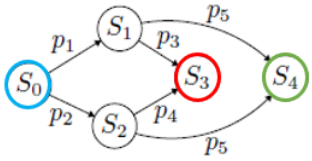

FIGURE 6: AP d'une porte OR

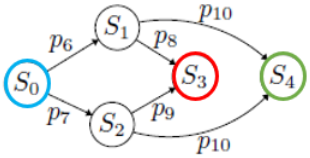

FIGURE 7: AP d'une porte AND

représentée par v_1. En parallèle, la deuxième machine effectue son action. La troisième machine va ainsi prendre la valeur maximale entre T_1 et T_2 puis ajouter son temps d'exécution. Ensuite, à nouveau, deux cas sont possibles, soit la faute est propagée dans la chaîne et on passe à la machine suivante, soit la faute est masquée et alors on recommence l'exécution à la première machine de la branche de l'arbre.

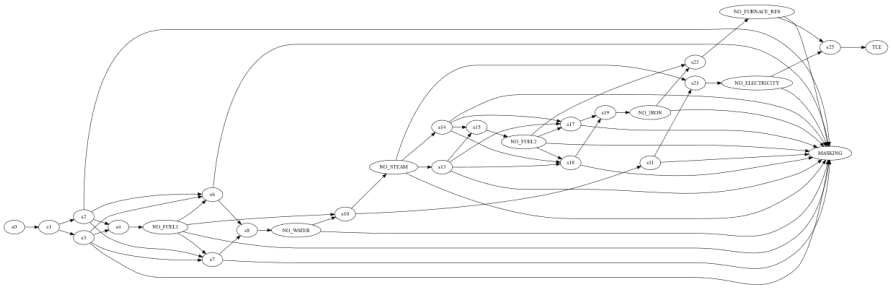

FIGURE 8: AP de notre chaîne de production

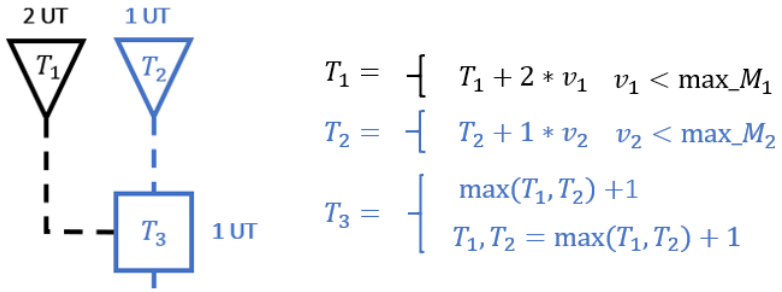

FIGURE 9: Parallélisme sur une chaîne de production

4 Implementation

Pour construire l'arbre de défaillances de notre chaîne de production, nous avons utilisé l'outil VisualParadigm Online [5], un outil de conception visuelle permettant entre autre de construire des arbres de défaillances. Comme expliqué dans les parties précédentes, le vérificateur de modèles probabilistes utilisé est PRISM. Cet outil nous permet d'analyser des systèmes qui présentent un comportement probabiliste tel que la propagation des erreurs dans un arbre de défaillance du au déclenchement d'événements perturbateurs dans une chaîne de production.

Sur PRISM, les portes logiques de notre arbre de défaillances représentant notre chaîne de production, sont chacune créées sous forme de module. Un module définie les états suivants pouvant d'être atteint en fonction de l'état actuel du système. Le module principal nommé "tree" définie l'ordre dans lequel les portes de l'arbre sont à visiter ainsi que la règle définissant si une faute à atteint le TLE et donc provoquée une panne globale du système. Le module "tree" est représenté en Figure 10.

```
module tree
    [] or1=0 -> (or1'=1);
    [] (v1=1) & (v2=0) & (or2=0) -> (or2'=1);
    [] (v1=1) & (v2=1) & (c_or1=1) -> (a_or3'=1) & (v1'=0);
    [] (v1=1) & (v2=1) & (c_or2=1) -> (b_or3'=1) & (v2'=0);

    [] v3=1 & (or4=0) & (v6=0) -> (or4'=1);
    [] v3=1 & (v6=1) & (or7=0) -> (or7'=1);

    [] (v4=1) & (v5=0) & (or5=0) -> (or5'=1);
    [] (v4=1) & (v5=1) & (c_or4=1) -> (a_or6'=1) & (v4'=0);
    [] (v4=1) & (v5=1) & (c_or5=1) -> (b_or6'=1) & (v5'=0);

    [] (v6=1) & (v7=1) & (c_or6=1) -> (a_or8'=1) & (v6'=0);
    [] (v6=1) & (v7=1) & (c_or7=1) -> (b_or8'=1) & (v7'=0);

    [] (c_or8=1) -> (tle'=1);
endmodule
```

FIGURE 10: Module tree

Un module pour une porte logique va de même définir à l'aide de règles, les états atteignables suivants du système. Ces règles déterminent typiquement les déclenchements possibles d'événements perturbateurs (pour des portes situées en bas de l'arbre), le masquage d'une faute ou alors au contraire la propagation d'une faute. Nous avons avec la Figure 11, le module représentant la première porte visitée de l'arbre de défaillance.

```
module or_gate1
    [] (or1=1) & (a_or1=0) & (b_or1=0) & (c_or1=0) & (m1=0) & (p1=0) -> 0.32: (a_or1'=1) & (or1'=2) + 0.05: (b_or1'=1) & (or1'=2) + 0.63: (or1'=1);
    [] (a_or1=1) & (or1=2) & (c_or1=0) & (p1=0) & (m1=0) & (t1<max_t1) -> 0.1: (m1'=1) & (t1'=t1+2) & (a_or1'=0) + 0.9: (p1'=1);
    [] (b_or1=1) & (or1=2) & (c_or1=0) & (p1=0) & (m1=0) & (t1<max_t1) -> 0.1: (m1'=1) & (t1'=t1+2) & (b_or1'=0) + 0.9: (p1'=1);

    [] (m1=1) -> (or1'=1) & (a_or1'=0) & (b_or1'=0) & (c_or1'=0) & (m1'=0) & (p1'=0);

    [] (p1=1) & (a_or1=1) & (c_or1=0) & (t1<max_t1) -> (c_or1'=1) & (v1'=1) & (t1'=t1+2);
    [] (p1=1) & (b_or1=1) & (c_or1=0) & (t1<max_t1) -> (c_or1'=1) & (v1'=1) & (t1'=t1+2);
endmodule
```

FIGURE 11: Module OR Gate 1

Avec PRISM, nous définition aussi les propriétés que nous voulons observer sur notre modèle. Comme expliqué dans la section 3, la propriété observée est définie par la formule suivante : $P_{max} =?[(FX_i = 1)\&(FTLE = 1)\&(FT < X_t)]$ dans le but de calculer la probabilité maximum qu'un événement X_i soit déclenché, que la faute se propage jusqu'en haut de l'arbre et cela avant X_t unités de temps. Nous calculons ainsi cette propriétés pour chaque événements possibles (X_i), nous donnant la possibilité de les comparer.

5 Evaluation

Pour évaluer la propriété voulue sur notre modèle, nous devons dans un premier temps définir les probabilités de déclenchement des événements bas niveau. Nous avons cinq machines qui composent notre chaîne de production avec pour quatre d'entre elles, deux événements perturbateurs pouvant les affectés contre un seul pour la cinquième machine. Les probabilité de déclenchement des événements sont ainsi représentés en Tableau 1.

Bottom Event	Probability trigger
Primary res. (X_1)	0.32
Machine failure (X_2)	0.05
Primary res. (X_3)	0.15
Machine failure (X_4)	0.20
Primary res. (X_5)	0.32
Machine failure (X_6)	0.05
Primary res. (X_7)	0.10
Machine failure (X_8)	0.05
Machine failure (X_9)	0.25

TABLE 1: Tableau des probabilités de déclenchement des événements perturbateurs

Concernant les probabilités du masquage et de la propagation des fautes, nous prenons respectivement 10% et $100-10=90\%$ pour chacune des portes de l'arbre. Finalement, nous avons choisit de calculer la propriété pour un temps $X_t = 10$ et un temps $X_t = 20$, c'est-à-dire : P_{max} =?$[(FX_i = 1)\&(FTLE = 1)\&(FT < 10)]$ et P_{max} =?$[(FX_i = 1)\&(FTLE = 1)\&(FT < 20)]$. Pour le premier cas, nous obtenons les résultats de la Figure 12 et la Figure 13 présente les résultats obtenus pour $X_t = 20$.

En analysant les résultats, nous pouvons observer que la probabilité qu'une faute se propage et atteigne le sommet de l'arbre est beaucoup plus élevée pour $X_t = 20$ que pour $X_t = 10$, ce qui implique que plus le temps de production est long, plus une panne technique est susceptible de se produire.

Nous observons aussi que les mêmes machines avec les mêmes propriétés à un même niveau de l'arbre de défaillances, donnent des résultats similaires et que le nombre d'unités de temps associées à chaque machine influence de manière significative les résultats. En effet, une faute se propageant à travers une machine ayant un temps d'exécution long aura moins de chance de se propager jusqu'au TLE avant un temps donné.

$$P = ?\left[(F\,X_1 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.54066383$$
$$P = ?\left[(F\,X_2 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.08447872$$
$$P = ?\left[(F\,X_3 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.26791824$$
$$P = ?\left[(F\,X_4 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.35722432$$
$$P = ?\left[(F\,X_5 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.54066383$$
$$P = ?\left[(F\,X_6 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.08447872$$
$$P = ?\left[(F\,X_7 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.41676170$$
$$P = ?\left[(F\,X_8 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.20838085$$
$$P = ?\left[(F\,X_9 = 1)\ \&\ (F\,TLE = 1)\ \&\ \left(T_{f1} < 10\right)\right] = 0.12502851$$

FIGURE 12: Résultats pour $P_{max} = ?[(FX_i = 1)\&(FTLE = 1)\&(FT < 10)]$

Finalement, un événement particulier sera plus impacté par les événements qui lui sont plus ou moins directement liés, c'est-à-dire les événements situé sur la même porte logique ou alors partageant la même branche/sous-partie de l'arbre.

6 Conclusion

Ce projet nous a donné la possibilité d'étudier plus en détail les arbres de défaillances et le principe de la propagation des fautes. Nous avons de plus, pu nous baser sur une nouvelle méthodologie pour l'analyse de ces arbres et adapter celle-ci pour notre cas pratique concernant une chaîne de production. A partir de ce modèle de la propagation des fautes, la propriété étudiée dans ce papier nous permet d'évaluer les chances qu'une chaîne de production tombe en panne avant d'atteindre un temps d'exécution donné. Des techniques développées dans la littérature telle que la triple redondance modulaire (TMR) par exemple, peuvent ensuite être appliqué à ce genre de système pour améliorer la fiabilité du système. La propriété ici étudiée est une parmi d'autres et il peut être intéressant pour des travaux futurs d'en vérifier d'autres à partir du modèle proposé.

$$P = ?\left[(F\ X_1 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.83347989$$
$$P = ?\left[(F\ X_2 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.13023123$$
$$P = ?\left[(F\ X_3 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.41301905$$
$$P = ?\left[(F\ X_4 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.55069207$$
$$P = ?\left[(F\ X_5 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.83347989$$
$$P = ?\left[(F\ X_6 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.13023123$$
$$P = ?\left[(F\ X_7 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.64247408$$
$$P = ?\left[(F\ X_8 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.32123704$$
$$P = ?\left[(F\ X_9 = 1)\ \&\ (F\ TLE = 1)\ \&\ \left(T_{f2} < 20\right)\right] = 0.19274222$$

FIGURE 13: Résultats pour $P_{max} = ?[(FX_i = 1)\&(FTLE = 1)\&(FT < 20)]$

Références

1. Factorio. https ://wiki.factorio.com/Main_Page.
2. Marwan Ammar, Ghaith Bany Hamad, Otmane Ait Mohamed, and Yvon Savaria. Efficient probabilistic fault tree analysis of safety critical systems via probabilistic model checking. In *2016 Forum on Specification and Design Languages (FDL)*, pages 1–8, 2016.
3. Documentation prism. https ://www.prismmodelchecker.org/.
4. Description de prism. https ://en.wikipedia.org/wiki/PRISM_model_checker.
5. Outil visualparadigm online. https ://online.visual-paradigm.com/fr/.