Analyse prévisionnelle des fautes des systèmes embarqués discrets par vérification model-based

Matthieu Perin, Jean-Marc Faure

Analyse prévisionnelle des fautes des systèmes embarqués discrets par vérification model-based*

Matthieu Perin, Jean-Marc Faure

Laboratoire Universitaire de Recherche en Production Automatisée
École Normale Supérieure de Cachan, 61 Avenue du Président Wilson, 94235 CACHAN Cedex, France

{matthieu.perin,jean-marc.faure}@lurpa.ens-cachan.fr
http://lurpa.ens-cachan.fr

Résumé— **Les méthodes usuelles d'analyse prévisionnelle des fautes s'intéressent principalement aux fautes aléatoires de composants physiques. Dans le cas des systèmes embarqués, qui comportent de nombreuses boucles de commande, il importe de vérifier également si une commande propage (ou ne propage pas) les fautes aléatoires du processus qu'elle contrôle, ceci afin d'éviter une défaillance du système. Ce papier propose une telle méthode d'analyse, basée sur une technique de vérification formelle model-based. Le principe de cette méthode est de vérifier qu'une propriété traduisant l'absence de défaillance est satisfaite (ou non) par un modèle formel du système bouclé. Si cette propriété est vérifiée, la commande ne propage pas la faute ; sinon, l'analyse d'un contre-exemple permet de modifier la commande afin d'améliorer la sûreté globale du système.**

Mots-clés— **Propagation de fautes, Preuve de propriétés, Analyse de traces, NuSMV.**

I. Introduction

Depuis une cinquantaine d'années, de nombreuses méthodes d'analyse prévisionnelle des fautes (arbre des fautes, diagramme de succès, ...) ont été développées afin d'améliorer la conception des systèmes critiques. Ces méthodes, largement utilisées dans les secteurs industriels concernés, s'intéressent uniquement aux fautes aléatoires des composants physiques du système, les modélisations sous-jacentes étant de type probabiliste.

Dans le cas des systèmes embarqués cependant, l'analyse prévisionnelle des fautes ne doit pas se limiter aux seuls composants physiques mais doit considérer le système bouclé constitué globalement d'un processus physique et d'une commande. En particulier, il importe d'analyser la manière dont la commande propage (ou ne propage pas) les fautes aléatoires des composants du processus physique. Une commande qui propage une faute aléatoire, par exemple en générant des signaux de sortie incompatibles avec l'état réel du processus, peut en efet conduire à une défaillance du système et devra donc être modifiée en conséquence.

Ce papier propose une méthode permettant d'effectuer cette analyse en se limitant aux systèmes embarqués qui peuvent être modélisés par des systèmes à événements discrets. Nous supposerons de plus par la suite que la commande ne comporte pas de fautes systématiques qui conduiraient à une défaillance lorsque le processus est exempt de fautes. Cette méthode d'analyse de la propagation des fautes aléatoires est basée sur une technique de vérification formelle par model-checking qualifiée de model-based [1], c'est-à-dire dans laquelle la vérification porte non pas sur un modèle de commande pris isolément, mais sur un modèle de commande couplé à un modèle de processus.

Il convient de remarquer que cette méthode se positionne clairement dans le cadre de l'analyse de sécurité à base de modèles (model-based safety analysis) proposée par [2]. Par rapport à cette dernière référence, qui utilise une modélisation sous forme de diagrammes Simulink, notre contribution propose des modèles de commande et de processus plus formels, indépendants d'un logiciel de conception, et s'intéresse également de manière détaillée à l'interprétation des résultats de preuve. L'expérience montre en effet qu'une preuve négative peut provenir d'une sensibilité de la commande à la faute du processus mais aussi d'une modélisation trop approximative. La maîtrise de ce problème, qui n'est pas mentionné dans la référence précédente, conditionne bien évidemment la réussite de l'approche.

En dernier lieu, nous tenons à souligner que, même si plusieurs auteurs ont développé des travaux visant à l'utilisation des techniques de model-checking pour l'analyse prévisionnelle des fautes, les objectifs de ces travaux : construction (ou enrichissement) automatique d'un arbre de défaillance relatif à un processus physique [3] [4] [5] ou analyse de la cohérence et de la complétude d'un tel arbre de défaillance préalablement élaboré [6] [7] [8] [9], diffèrent nettement de celui de notre proposition.

Une description de la méthode d'analyse est donnée dans la section II. Les sections III et IV sont consacrées respectivement à la représentation des modèles initiaux de la commande et du processus ainsi qu'à la traduction de ces modèles dans le langage de l'outil de vérification. La section V s'intéresse enfin à l'interprétation des résultats fournis par cet outil.

* Ce travail à été réalisé dans le cadre d'un projet de recherche coopérative avec la Direction de la Recherche et de l'Innovation Automobile (DRIA) de PSA Peugeot-Citroën.

II. Présentation générale de la méthode

Le principe de cette méthode est de prouver, à l'aide d'un model-checker existant, qu'un modèle formel, qui représente le comportement du système bouclé constitué de la commande couplée au processus qu'elle contrôle, satisfait ou non une propriété. Ceci permet de vérifier si la commande ne propage pas, ou propage, une faute aléatoire du processus si la propriété utilisée traduit qu'une défaillance due à cette faute ne doit pas se produire

La figure 1 montre de manière synthétique les quatre principales étapes de cette méthode :

1. Traduction des modèles de la commande issus du processus industriel de développement dans la syntaxe du model-checker.

2. Construction des modèles représentant le comportement du processus commandé et traduction de ces modèles dans cette syntaxe.

3. Écriture de la propriété à vérifier en logique temporelle CTL, logique acceptée par le model-checker.

4. Interprétation des résultats de preuve.

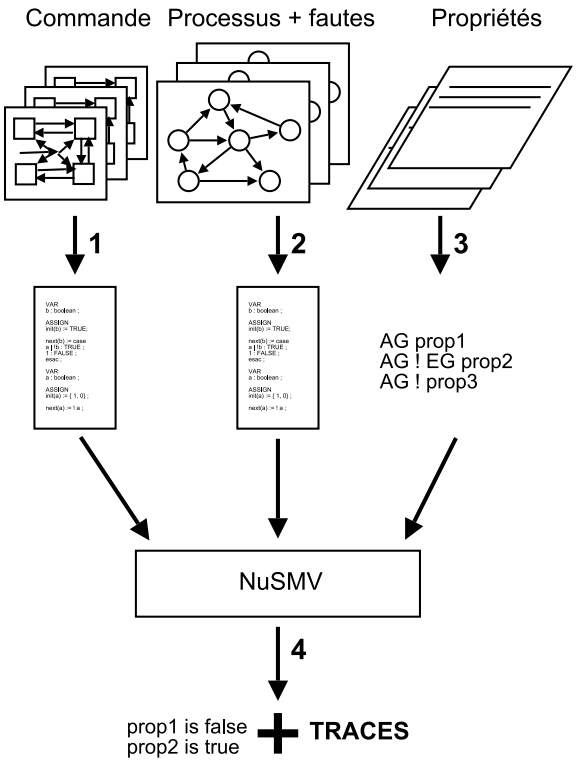

Fig. 1. Principe de la méthode proposée

Dans l'optique d'une utilisation de cette méthode sur des cas de taille et de complexité industrielles, nous avons retenu le model-checker symbolique et non temporisé NuSMV [10] qui permet d'éviter (ou de limiter) les problèmes d'explosion combinatoire inhérents aux méthodes de model-checking. Un modèle NuSMV représente, dans une syntaxe particulière, un système de transitions. Ce type de représentation est bien adapté à la preuve de propriétés, mais l'est nettement moins pour la modélisation d'une commande, d'un processus physique ou de fautes de composants de ce processus. Ceci explique que nous ayons choisi d'exprimer les modèles correspondants à l'aide de modèles à états discrets, qui sont présentés dans la section suivante. La traduction de ces modèles, adaptés à l'automaticien, en systèmes de transitions fait l'objet de la section IV.

III. Modèles initiaux

Cette section est consacrée à la présentation des modèles sur lesquels repose la méthode. Tout au long de ce papier, nous appuierons les explications sur un exemple simple : un système d'essuie-glace de pare-brise automobile (figure 2) qui comprend :

- deux détecteurs produisant les variables booléennes *EAV1* et *EAV2* qui codent la position du commodo de balayage à quatre positions : STOP, AUTO, PV (balayage Petite Vitess), et GV (balayage Grande Vitesse).
- un détecteur de la position basse du balai qui génère la variable booléenne AREFIX.
- un détecteur de pluie qui fournit la variable booléenne INFO_PLUIE.
- un moteur qui met en mouvement le balai et qui est piloté par la variable UMOT.

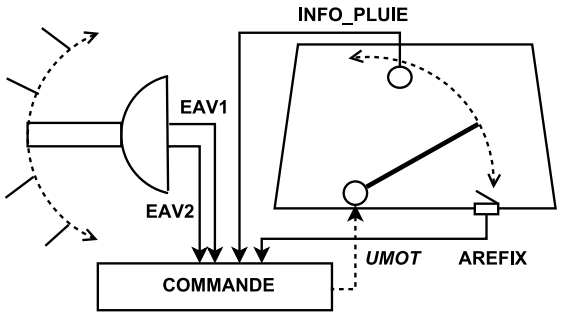

Fig. 2. Schéma du système d'essuyage

A. Modèles de la commande

Le processus de développement industriel des automatismes embarqués exige que ces modèles soient décrits en statecharts.

Le modèle de commande comporte quatre états :

- MODE_STOP : le balai stoppe sa course en position basse.
- MODE_AUTO : le balayage est conditionné par la détection de pluie.
- MODE_PV : balayage continu en petite vitesse.
- MODE_GV : balayage continu en grande vitesse.

Les évolutions entre ces états reflètent les positions successives possibles du commodo (STOP $\rightleftharpoons$ AUTO $\rightleftharpoons$ PV $\rightleftharpoons$ GV) ainsi qu'une initialisation possible vers tous les états conditionnée par un test sur la position du commodo.

La sémantique de ce modèle est donnée dans [11] et on pourra se reporter à [12] et [13] pour une traduction de modèles statecharts ou SFC dans le langage formel de NuSMV.

B. Modèles du processus et des fautes

B.1 Formalisme utilisé

Les modèles du processus et des fautes sont décrits en Machines de Moore Non déterministes (MMN) formellement définies par le 6-uplet $(\Sigma_{in}, \Sigma_{out}, S, S_0, \tau, \Lambda)$ avec :

- Σ_{in} : l'alphabet d'entrée non vide et fini de la MMN.
- Σ_{out} : l'alphabet de sortie non vide et fini de la MMN.
- S : l'ensemble non vide et fini des états de la MMN.
- S_0 : le sous-ensemble non vide de S définissant les états initiaux de la MMN.
- τ : la fonction transition de la MMN, soit $S \times \Sigma_{in} \rightarrow S$, l'état d'arrivée n'étant pas forcément unique (non déterminisme).
- Λ : la fonction d'assignation de la MMN, soit $S \rightarrow \Sigma_{out}$.

Ce formalisme a été retenu pour les deux raisons suivantes :

- il permet de prendre en compte les entrées et sorties des modèles (valeurs des commandes, informations sur l'état réel du processus ou fournies par un détecteur défectueux) ;
- il permet de modéliser des comportements dépendant du temps, en remplaçant le temps physique par un temps logique. Les évolutions du processus physique sont alors représentées par des transitions non déterministes.

Afin de faciliter la lecture de nos modèles, on adoptera les conventions de représentation suivantes :

- Les transitions (éléments de τ) sont représentées par une flèche, les conditions de franchissement (éléments de Σ_{in}) sont inscrites en police normale.
- L'absence de condition sur une transition signifie qu'elle est toujours franchissable.
- Les assignations (éléments de Λ) sont écrites en gras.
- Les états initiaux (S_0) sont représentés par des états ayant une flèche entrante.

B.2 Modèles du processus

Ces modèles sont créés en s'inspirant des travaux présentés dans [14] et [15] et en adoptant les critères de conception suivants :

- Chaque mouvement et chaque état limite du processus doit être représenté par un état du modèle.
- Lorsque l'information fournie par un capteur ne dépend que du fonctionnement d'un actionneur, le modèle de ce capteur peut être intégré au modèle de cet actionneur, sinon il doit faire l'objet d'un modèle séparé.
- L'état initial est le plus souvent laissé libre afin de représenter une initialisation en position quelconque du processus.
- Le temps physique est remplacé par un temps logique et des transitions non déterministes.
- Les modèles sont crées nominaux, c.-à.d. exempts de fautes.

La figure 3 montre le modèle du processus « balai du système d'essuyage » qui comprend quatre états représentant les positions BASse et HAUTe du balai ainsi que les mouvements de MONTée et DESCente. On peut remarquer que les évolutions ne dépendent que de la variable UMOT (commande du moteur) car l'inversion du sens de rotation du balai est réalisée mécaniquement (pas de changement du sens de rotation du moteur). L'information (SRC_AREFIX) fournie par le détecteur de position basse du balai ne dépendant que de la position du balai, son évolution a donc été intégrée au modèle de celui-ci. Afin de conserver la possibilité d'une initialisation dans un état quelconque, tous les états du modèle sont initiaux. Pour modéliser les mouvements du processus physique, des transitions non déterministes sont introduites (le modèle peut rester un nombre indéterminé d'évolutions dans un état avant de passer au suivant).

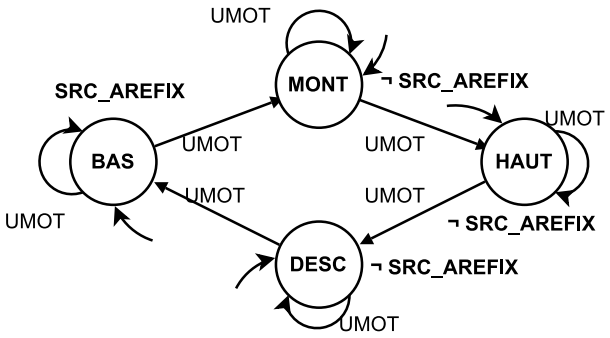

Fig. 3. Machine de Moore non déterministe du balai avec : $S = S_0 = \{\text{BAS, MONT, HAUT, DESC}\}$, $\Sigma_{in} = \{\text{UMOT}\}$ et $\Sigma_{out} = \{\text{SRC\_AREFIX}, \neg \text{SRC\_AREFIX}\}$, $\Lambda(BAS) = \text{SRC\_AREFIX}$, $\Lambda(HAUT, MONT, DESC) = \neg \text{SRC\_AREFIX}$.

B.3 Modèle de faute

Nous ne traiterons dans ce papier que des fautes non réparables de détecteurs logiques. Ces fautes sont modélisées par une transformation des variables issues des modèles nominaux du processus (figure 4). Ce choix de ne pas intégrer les modèles de fautes dans les modèles du processus est motivé par le souci de disposer, dans les contre-exemples fournis en cas de preuve négative, à la fois de l'état réel du processus commandé (modèles nominaux) et de l'état perçu par la commande.

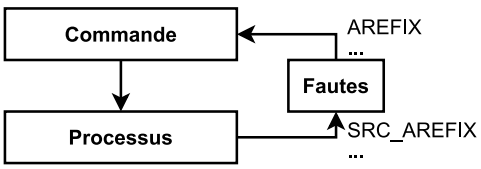

Fig. 4. Implantation des fautes dans la boucle de commande du système

La figure III-B.3 montre le modèle de faute du détecteur de la position basse du balai. Dans l'état OK de ce modèle, qui correspond à l'absence de faute, la variable AREFIX, transmise au modèle de commande, prend la valeur de la variable SRC_AREFIX, issue du modèle nominal du processus. Dans les autres états, AREFIX est forcée soit à l'état vrai, soit à l'état faux, suivant le type de faute.

Étant donné que nous nous limitons au cas de fautes non-réparables, les deux états de la MMN correspondant à une faute sont des états puits.

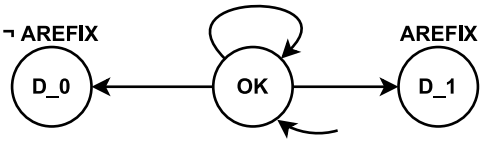

Fig. 5. Machine de Moore non déterministe de faute du capteur booléen AREFIX avec :
S = {OK, D_1, D_0},S_0 = {OK}
Σ_{in}= $\emptyset$ et Σ_{out} = {AREFIX,¬ AREFIX},
$\Lambda(D\_1)$ = AREFIX, $\Lambda(D\_0)$ = ¬ AREFIX .

IV. Obtention des modèles NuSMV

A. Éléments de syntaxe NuSMV

Un modèle NuSMV représente, sous la forme d'un système de transitions, les évolutions d'un ensemble de variables. Les opérateurs utilisés pour décrire ces évolutions sont les opérateurs logiques usuels et les opérateurs de syntaxe de base qui sont notés :

- & pour le ET logique.
- | pour le OU logique.
- ! pour le complément.
- := pour l'assignation d'une valeur à une variable.
- ; pour la fin d'une instruction.

ainsi que les opérateurs *init* et *next* dont l'utilisation est indiquée ci-dessous. L'opérateur *init*( ) permet de définir l'ensemble, éventuellement réduit à un singleton, des valeurs que peut prendre une variable à l'initialisation du modèle ; il est donc possible d'exprimer du non déterminisme à l'initialisation, lorsque cet ensemble comporte plusieurs éléments. L'opérateur *next*( ), quant à lui, permet de désigner la future valeur d'une variable. Dans le cadre d'une assignation, par exemple, il est possible d'écrire :

- $next(a) := true$, a étant une variable logique, pour assigner à la valeur future de cette variable la valeur vrai,
- $next(a) := next(b)$, pour deux variables ayant des évolutions simultanées,
- $next(a) := \{true, false\}$, a étant une variable logique, pour assigner à la valeur future de cette variable une valeur non déterministe, prise dans un ensemble.

B. Modèle NuSMV du balai

Ce modèle (figure 6) est obtenu à partir de la MMN de la figure 3. Dans un souci de modularité, deux systèmes de transitions, qui évoluent simultanément, ont été définis. Le premier d'entre eux représente les états et les transitions entre états, le second, les évolutions de la variable SRC_AREFIX.

C. Expression des propriétés

Chaque propriété à prouver traduit le fait qu'une défaillance du système, événement indésirable (au sens de [16]), ne se produit pas. Nous illustrerons l'approche à l'aide de la défaillance suivante :

```
-- Definition de l'automate : balai

VAR

  balai : desc, bas, haut, mont;

ASSIGN

  init(balai) := {bas, mont, haut, desc};

  next(balai) :=
    case
      (balai = mont) & umot : {mont, haut};
      (balai = bas) & umot  : {bas, mont};
      (balai = desc) & umot : {desc, bas};
      (balai = haut) & umot : {haut, desc};
      1                     : balai;
    esac;
-- Definition de la variable - processus : SRC_AREFIX/boolean

VAR

  SRC_AREFIX : boolean;

ASSIGN

  init( SRC_AREFIX ) := (balai = bas);

  next( SRC_AREFIX ) :=
    case
      balai = bas & next(balai) = mont & (UMOT) : FALSE;
      balai = desc & next(balai) = bas & (UMOT) : TRUE;
      1                                         : SRC_AREFIX;
    esac;
```

Fig. 6. Code NuSMV du balai décomposé en 2 systèmes de transitions :
l'un pour les évolutions de la position du balai,
l'autre pour les évolutions de la variable booléenne SRC_AREFIX.

« Le balayage de l'essuie-glace ne s'arrête pas alors que le commodo est mis en position STOP ».

Cette défaillance pouvant conduire à une détérioration potentielle du balai, et donc à un défaut d'un élément de sécurité du véhicule, doit être évitée. Nous vérifierons si la faute du détecteur de position basse du balai peut se propager au travers de la commande afin de faire apparaître cette défaillance du système.

Dans le cas étudié, pour vérifier qu'il n'y a pas propagation de la faute, il faut prouver sur le modèle du système incluant la faute que la propriété énoncée de façon informelle suivante : « le moteur s'arrête et reste à l'arrêt lorsque le commodo est mis en position STOP » est toujours vraie.

Il est alors possible d'énoncer formellement, en logique temporelle CTL, cette propriété de la manière suivante :

$$AG\,!\,EG\,(\,UMOT\,\&\,!\,EAV1\,\&\,!\,EAV2\,)$$

Où :

- ! correspond à l'opérateur logique NON,
- E est le quantificateur de chemin : il Existe un chemin,
- A est le quantificateur de chemin : pour tous les chemins (All),
- G est l'opérateur de temps : tout le long du chemin (Globally),
- $UMOT\,\&\,!\,EAV1\,\&\,!\,EAV2$ correspond au fait que le moteur est piloté ($UMOT$) et que le commodo est en position stop ($!\,EAV1$ et $!\,EAV2$).

On pourra remarquer qu'une formulation plus simple,

telle que : $AG!(UMOT\&!EAV1\&!EAV2)$ ne correspond pas à la défaillance étudiée. Cette formulation correspond en effet à des évaluations simultanées des trois variables le long de tous les chemins. Or la défaillance étudiée concerne une séquence telle qu'après la mise en position STOP du commodo, la commande continue à maintenir UMOT à vrai, et ce indéfiniment (EG).

V. Interprétation des résultats de preuve

Une preuve positive (la propriété est vérifiée) indique que le système bouclé n'est jamais défaillant, une preuve négative, que la défaillance correspondant à la propriété étudiée peut se produire. Dans ce dernier cas, l'outil de model-checking fournit un contre-exemple sous la forme d'une trace d'évolutions conduisant à un état du système dans lequel la défaillance se produit. Cette section est consacrée à l'analyse des contre-exemples dans plusieurs cas de figure.

A. Test avec processus sans fautes

L'objectif de ce type de preuve n'est bien évidemment pas d'étudier la propagation des fautes du processus par la commande, mais de valider les modèles nominaux. A priori, en effet, toute propriété indiquant qu'une défaillance ne doit pas se produire devrait être vérifiée dans ce cas. Il peut arriver, cependant, que des contre-exemples soient obtenus ; ces contre-exemples proviennent d'une modélisation trop approximative comme il est indiqué ci-après.

A.1 Contre-exemples issus d'une erreur de modélisation de la synchronisation entre commande et processus

Ces contre-exemples contiennent des évolutions simultanées des modèles de la commande et du processus. Or ceci n'est pas vrai dans un système bouclé réel où la commande est plus réactive que le processus. Cette erreur de modélisation est corrigée en définissant une variable GO_TEMPO, qui n'est vraie que lorsque la commande est stable, c.-à.d. que l'état actif courant du modèle de commande est identique à son état actif précédent. Lorsque GO_TEMPO devient vraie, le modèle de processus peut évoluer. Toute évolution de ce modèle force la variable GO_TEMPO à faux et provoque l'(les) évolution(s) du modèle de commande jusqu'à stabilité, et ainsi de suite. Le modèle du processus doit donc être modifié en introduisant cette variable dans le système de transitions qui définit ses évolutions, comme indiqué figure 7 dans le cas du modèle du balai.

A.2 Contre-exemples dus à la modélisation du temps retenue

Une commande réelle fait souvent référence au temps physique, en particulier en utilisant des temporisations qui permettent de modéliser une réaction du processus non observée par un capteur. Le temps physique ayant été remplacé par un temps logique lors de la traduction dans le langage du model-checker, il est possible d'obtenir des contre-exemples qui ne correspondent pas à des évolutions réalistes du processus. Ce problème est éliminé en introduisant des modèles de temporisation logiques, modèles à trois états (inactif, en_cours, écoulé) et à transitions non déterministes, comme indiqué dans [17], et en contraignant les évolutions de la commande par celles de ces modèles.

```
-- Definition de l'automate : balai

VAR

  balai : desc, bas, haut, mont;

ASSIGN

  init(balai) := OUT;

  next(balai) :=
    case
      (balai = mont) & umot & GO_TEMPO  : {mont, haut};
      (balai = bas) & umot & GO_TEMPO   : {bas, mont};
      (balai = desc) & umot & GO_TEMPO  : {desc, bas};
      (balai = haut) & umot & GO_TEMPO  : {haut, desc};
      1                                 : balai;
    esac;
```

Fig. 7. Code NuSMV correspondant au modèle du balai modifié, le modèle de la variable SRC_AREFIX n'a pas été représenté car il ne change pas.

Les deux problèmes exposés ci-dessus étant résolus, il est alors possible de vérifier les propriétés en présence de fautes du processus.

B. Test avec processus sujet à fautes : contre-exemples montrant la réaction de la commande

Lorsque la propriété n'est pas vérifiée, ce qui indique que la commande propage la faute, un contre-exemple tel que celui présenté en figure 8 est fourni. Dans ce contre-exemple, les rectangles représentent les états du système (processus + fautes + commande) et les flèches, les transitions entre ces états. À côté de chaque flèche figurent les seules variables ayant changé de valeur lors de la transition et dont la connaissance est utile à la compréhension de l'exemple. Outre les variables précédemment définies (UMOT, GO_TEMPO, AREFIX et SRC_AREFIX), les variables suivantes sont introduites :

- METEO, qui peut prendre deux valeurs PLUIE ou SOLEIL.
- CMD, qui indique l'état actif du modèle de commande (Cf. section III-A).
- BALAI, qui indique l'état actif du modèle de processus (Cf. section III-B.2).
- ETAT_AREFIX, qui indique l'état actif du modèle de faute (Cf. section III-B.3).

De manière synthétique, ce contre-exemple comporte trois familles d'états :

1. l'état initial, où METEO=PLUIE, le commodo est en position AUTO et ETAT_AREFIX vaut OK. La commande passe donc à l'état MODE_AUTO et le moteur est mis en route lors de la première transition (car on détecte de la pluie).

2. les états 2 et 3, qui représentent l'évolution vers la configuration redoutée, caractéristique de la défaillance. Une faute du détecteur de position basse se produit lors de la première transition : AREFIX est donc forcée à 0. Par la suite, la météo change (transition de l'état 2 à l'état 3) et le conducteur souhaite arrêter le balayage (même transition).

3. une boucle (états 4 à 11) qui représente la défaillance. Malgré le souhait du conducteur, le balai ne s'arrête pas.

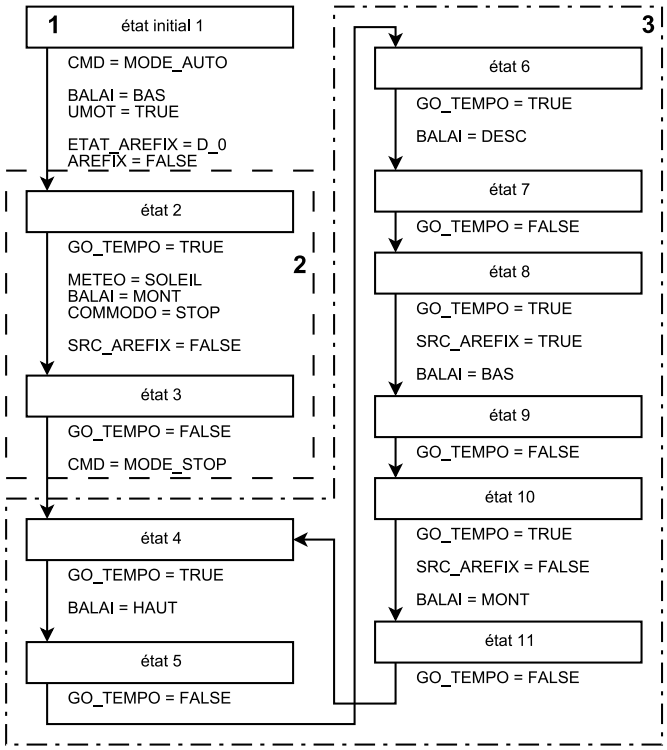

Fig. 8. Contre-exemple de la propriété

C. *Amélioration de la commande à l'aide des contre-exemples fournis*

L'analyse d'un contre-exemple permet de modifier la commande afin d'éviter que la faute du processus soit propagée et conduise à une défaillance. Dans le cas de notre exemple, la modification consiste à introduire une surveillance de la variable AREFIX lorsque le moteur est alimenté (UMOT est vraie). Si cette variable garde une valeur constante durant un temps donné (ce qui ne devrait pas être le cas en présence d'un mouvement du balai), la commande passe en mode dégradé ; dans ce mode, l'arrêt du balayage ne se fait plus sur occurrence de AREFIX mais après un temps fixé. Cette solution a été testée avec la méthode proposée, ce qui a permis de prouver qu'ainsi modifiée, la commande ne propage plus la faute du capteur de position basse du balai.

VI. Conclusion

La méthode présentée dans ce papier permet de vérifier si une commande propage (ou ne propage pas) une faute du processus qu'elle contrôle. Cette méthode est basée sur une technique de model-checking model-based. Il a été montré à la fois comment construire les différents modèles utilisés, en évitant des erreurs de modélisation pouvant être préjudiciables à l'analyse, et comment formuler les propriétés à prouver, en fonction des défaillances à éviter. L'analyse des éventuels contre-exemples obtenus en cas de preuve négative permet de modifier la commande afin d'éviter qu'une faute du processus se traduise par une défaillance du système. Cette méthode est donc un outil supplémentaire d'analyse prévisionnelle des fautes qui vient en renfort des nombreuses méthodes permettant d'accroître la sûreté des systèmes critiques. Nos travaux en cours concernent l'automatisation de cette méthode et la prise en compte du temps physique.

Références

[1] G. Frey et L. Litz. Formal methods in PLC programming. *IEEE Systems Man and Cybernetics (SMC) 2000*, pages 2431–2436, Nashville, October 2000.

[2] A. Joshi, S. P. Miller, M. Whalen, et Mats P.E. Heimdahl. A proposal for model-based safety analysis. *24th Digital Avionics Systems Conference*, October 2005.

[3] M. Bozzano et A. Villafiorita. Integrating fault tree analysis with event ordering information. *ESREL 2003*, pages 247–254, Maastricht, The Netherlands, June 2003.

[4] M. Bozzano et A. Villafiorita. Improving system reliability via model checking : the fsap / nusmv-sa safety analysis platform. *SAFECOMP*, pages 49–62, 2003.

[5] C. Kehren et C. Seguin. Evaluation qualitative de systèmes physiques pour la sureté de fonctionnement. *FAC 03*, Toulouse, France, 2003.

[6] A. Schäfer. Combining real-time model-checking and fault-tree analysis. *LNCS 2003*, pages 522–541, 2003.

[7] G. Schellhorn, A. Thums, et W. Reif. Formal fault tree semantics. *Proceedings of the Sixth World Conference on Integrated Design & Process Technology*, Pasadena, CA, June 2002.

[8] A. Thums et G. Schellhorn. Model checking fta. *World Congress on Formal Methods FME, LNCS 2805*, pages 739–757, 2003.

[9] J. Xiang. *FTA and Formal Methods for requirements engineering*. PhD thesis, Japan Advanced Institute of Science and Technologie, September 2005.

[10] Center for Scientific and Technological Research of the Autonomous Province of Trento ITC-IRST. http ://nusmv.irst.itc.it/. NuSMV website. Italy.

[11] W. Damm, B. Josko, H. Hungar, et A. Pnueli. A compositional real-time semantics of STATEMATE designs. *Lecture Notes in Computer Science*, 1536 :186–238, 1998.

[12] E. Clarke et W. Heinle. Modular translation of statecharts to smv. Technical Report CMU-CS-00-XXX, Carnegie Mellon University School of Computer Science, August 2000.

[13] S. Bornot, R. Huuck, Y. Lakhnech, et B. Lukoschus. Verification of sequential function charts using smv. *PDPTA 2000 special session on Formal Validation*, 2000.

[14] J. Machado. *Influence de la Prise en Compte d'un Modèle de Processus en Vérification Formelle des Systèmes à Evénements Discrets*. PhD thesis, École Normale Supérieure de Cachan, Universidade do Minho, June 2006.

[15] D. Gouyon. *Contrôle par le produit des systèmes d'exécution de la production : apport des techniques de synthèse*. PhD thesis, Université Henri Poincaré, Nancy-I, December 2004.

[16] W. Vesely, M. Stamatelatos, J. Dugan, J. Fragola, J. Minarick III, et J. Railsback. *Fault Tree Handbook with Aerospace Applications*, pages 25–26. NASA Office of Safety and Mission Assurance, 2002.

[17] V. Gourcuff. *Représentations formelles efficaces pour l'aide à la certification de contrôleurs logiques industriels*. PhD thesis, École Normale Supérieure de Cachan, December 2007.