Elaboration de propriétés formelles de contrôleurs logiques à partir d'analyse prévisionnelle par Arbre des Défaillances

Israel Barragan Santiago

THESE DE DOCTORAT
DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Présentée par

Monsieur Israel BARRAGAN SANTIAGO

pour obtenir le grade de

DOCTEUR DE L'ECOLE NORMALE SUPERIEURE DE CACHAN

Domaine :
Electronique Electrotechnique Automatique

Elaboration de propriétés formelles de contrôleurs logiques à partir d'analyse prévisionnelle par Arbre des Défaillances

Thèse présentée et soutenue à Cachan le 6 juillet 2007 devant le jury composé de :

D. NOYES	Professeur à l'ENI de Tarbes	Président
Y. DUTUIT	Professeur à l'Université de Bordeaux	Rapporteur
E. NIEL	Professeur à l'INSA de Lyon	Rapporteur
J.J. LESAGE	Professeur à l'ENS Cachan	Examinateur
J.M. FAURE	Professeur à l'ENS Cachan	Directeur de thèse

Laboratoire Universitaire de Recherche en Production Automatisée
Ecole Normale Supérieure de CACHAN
61, avenue du Président Wilson, 94235 CACHAN CEDEX (France)

À la mémoire
de ma mère,
Magdalena Santiago

Remerciements

Le travail de recherche exposé dans ce mémoire de thèse a été réalisé au sein du Laboratoire Universitaire de Recherche en Production Automatisée (LURPA) de l'Ecole Normale Supérieure de Cachan.

Je tiens à exprimer mes vifs remerciements et toute ma reconnaissance à Monsieur le Professeur Jean-Marc FAURE pour avoir assuré la direction de mes travaux et pour la qualité de son encadrement. Tout au long de ces années de thèse, il a su m'apporter son expérience et son soutien scientifique. Muchas gracias.

Je remercie Messieurs les Professeurs Yves DUTUIT et Eric NIEL de me faire l'honneur d'être les rapporteurs de cette thèse et Monsieur le Professeur Daniel NOYES de m'accorder le privilège d'être membre de mon jury.

Je souhaite exprimer ma profonde reconnaissance à Monsieur le Professeur Jean-Jacques LESAGE de faire partie de mon jury mais aussi de m'avoir accueilli au sein du LURPA et de sa gentillesse vers moi.

J'adresse aussi mes sincères remerciements au Conseil National de Science et de Technologie du Mexique (CONACYT) pour son soutien économique, ce qui m'a permis de mener à bien mon séjour et mon travail en France.

Pour leur contribution à la réalisation d'une partie du travail, je remercie Monsieur le Professeur Yiannis PAPADOPOULOS, et également Matthias ROTH, Vincent GOURCUFF et Yann HIETTER. Pour son assistance dans la résolution de problèmes informatiques, l'installation des logiciels et surtout pour ses qualités humaines et son amitié, j'exprime aussi toute ma gratitude à Marc JACHYM. Mes remerciements vont également à tous mes collègues de l'équipe ISA (permanents et étudiants) pour les conseils techniques et scientifiques et pour les discussions sympathiques. Je tiens aussi à exprimer ma reconnaissance à Cécile et Françoise pour leur aide administrative, et à l'ensemble des membres du LURPA.

J'accorde enfin une attention particulière à tous mes amis : Charyar, David, Steve, Thomas (de vrais hermanos), Annick, Antonio, Eric, Gaëlle.

Il me reste à remercier les personnes qui comptent tant : Isabelle, pour la correction orthographique d'une partie du mémoire de thèse et pour sa tendresse. Miguel et Inga, qui m'ont adopté comme un frère. Alethia, Alejandro, Fabiola, Iván, Muganes et Norma, mes amis au Méxique. Zulema, pour son amitié très chère, sa compagnie durant toutes ces années de thèse et sa solidarité. Ma sœur Mónica, qui a toujours été là et m'a apporté un soutien inconditionnel. Ma grand-mère, mon père, mes frères et toute ma famille.

Table des matières

Introduction

Le développement d'un système technologique, tel qu'un système de transport, de production d'énergie, un appareillage médical, …, exige de s'intéresser non seulement à son comportement nominal, mais aussi à son fonctionnement dans des conditions dégradées, afin notamment d'évaluer ses capacités à fournir un service, même réduit, sans causer de dommages aux biens, aux personnes et à l'environnement. La sûreté de fonctionnement des systèmes technologiques n'est donc pas une préoccupation nouvelle. Depuis les années 50, de nombreuses méthodes pour améliorer la sûreté de fonctionnement (diagramme de succès, analyse des défaillances, ...), basées généralement sur des modèles probabilistes des composants physiques des systèmes, ont en effet été développées. Il existe donc aujourd'hui de nombreuses méthodes pour la prévention, la prévision, l'élimination des fautes et pour la tolérance aux fautes.

Depuis une vingtaine d'années, les systèmes technologiques se sont rapidement transformés et améliorés grâce aux nouvelles technologies d'automatisation. Ainsi, à l'heure actuelle, un système technologique est fortement automatisé et intègre (ou embarque) de nombreux contrôleurs qui assurent différentes fonctions de commande. Pour assurer la sûreté d'un système technologique actuel, il convient par conséquent de prendre en compte non seulement les fautes ou défaillances de ses composants physiques mais aussi celles des contrôleurs qui commandent ces composants. En effet, un contrôleur défaillant peut entraîner très aisément un service incorrect du système.

L'origine de ce travail de thèse se situe dans ce constat, en limitant notre analyse au cas des contrôleurs logiques, dont l'importance est notoire.

Pour concevoir un contrôleur logique sûr de fonctionnement, il est intéressant d'appliquer, durant les phases de conception et d'implantation, des méthodes de vérification formelle. Ces méthodes formelles permettent de vérifier, de manière exhaustive, que le modèle du contrôleur étudié satisfait ou ne satisfait pas des propriétés formelles. Elles constituent un pré-requis à l'élimination des fautes du contrôleur ; l'élimination d'une faute nécessite en effet que cette faute soit détectée, par exemple en montrant qu'une propriété n'est pas satisfaite.

Une technique de vérification formelle très connue et basée sur la théorie des systèmes à événements discrets est le model-checking. Pour vérifier automatiquement un contrôleur logique par model-checking, il est nécessaire d'en construire une représentation formelle sous la forme d'un automate à états, et ensuite d'énoncer formellement les propriétés à vérifier.

Enoncer de façon formelle, à partir des spécifications, les propriétés du contrôleur logique à vérifier n'est pas une tâche simple et représente un des obstacles majeur pour la mise en place de la technique du model-checking dans l'industrie, où la pratique courante est d'exprimer les spécifications de manière informelle, c'est-à-dire avec quelques phrases en langage naturel ou avec des schémas mais jamais avec des expressions formelles.

L'objectif du travail de thèse présenté ici est de **faciliter la tâche d'obtention des propriétés formelles de contrôleurs logiques**. Ainsi, tout au long de ce mémoire, une méthode visant à l'élaboration de propriétés formelles à partir des spécifications sera exposée. La principale caractéristique de cette méthode est qu'elle s'appuie sur une technique connue et utilisée dans l'industrie pour la conception des systèmes critiques : la méthode de l'Arbre des Défaillances (AdD). L'un des intérêts de ce choix est que les résultats de l'analyse sont montrés de façon graphique dans une structure arborescente.

En corollaire, nous pouvons également indiquer que cette méthode permettra d'établir un lien entre les méthodes de prévision des fautes, développées et appliquées pour les systèmes physiques depuis de nombreuses années, et les méthodes formelles de détection des fautes des contrôleurs logiques, apparues plus récemment.

Ce mémoire de thèse est organisé comme indiqué ci-après.

Le premier chapitre permet de définir le contexte du travail, la conception de contrôleurs logiques sûrs de fonctionnement, et la problématique abordée, l'obtention des propriétés formelles d'un contrôleur logique en vue de sa vérification par model-checking.

La méthode de l'Arbre des Défaillances est un élément essentiel dans notre approche. En raison de son importance, le deuxième chapitre y est entièrement consacré. Ce chapitre comprend tout d'abord une présentation des bases de cette méthode ; une étude bibliographique qui recense les principaux travaux scientifiques développés afin d'étendre cette méthode d'analyse prévisionnelle des fautes est ensuite conduite. A la fin du chapitre, en nous basant sur l'étude précédente, nous formulons quatre propositions permettant d'atteindre notre objectif ; deux de ces propositions sont de nature méthodologique, les deux autres étant de nature formelle.

Les deux propositions de nature méthodologique concernent, d'une part l'intégration des **fautes systématiques** du contrôleur logique dans la structure classique de l'arbre des défaillances, et plus particulièrement dans le modèle générique de faute de composant, et

d'autre part la définition d'un vocabulaire de portes particulières permettant d'exprimer ces fautes systématiques. Ces deux propositions sont détaillées au chapitre trois.

Quant aux deux propositions de nature formelle, elles sont exposées aux chapitres quatre et cinq. Le chapitre quatre est consacré entièrement à la définition des représentations formelles du comportement de chacune des portes du vocabulaire retenu dans notre approche. Une fois ces représentations obtenues, nous montrons comment il est possible d'en déduire des propriétés formelles de contrôleur logique acceptables par des model-checkers existants.

Au cinquième chapitre, nous nous intéressons à la représentation formelle de différentes associations de portes. Certaines des associations considérées requièrent une analyse préalable de leur cohérence, ce qui est également développé dans ce chapitre.

Finalement notre méthode est appliquée à un système réel au chapitre six. L'étude de cas développée dans ce dernier chapitre du mémoire de thèse permet de montrer l'intérêt de notre méthode, qui vise à faciliter l'obtention des propriétés formelles en vue de vérification.

Le mémoire se conclut par quelques indications sur des perspectives de recherche qui peuvent être conduites à partir des résultats de ce travail.

Chapitre 1.

Sûreté de fonctionnement des contrôleurs logiques

L'objectif de ce premier chapitre est de présenter la problématique de cette thèse qui s'insère dans le domaine de la conception des contrôleurs logiques sûrs de fonctionnement. Tout d'abord, nous définirons ce qu'est un contrôleur logique. Ensuite, nous présenterons les concepts de base de la sûreté de fonctionnement des systèmes. Le chapitre se poursuit par la présentation des méthodes qui contribuent au développement d'un contrôleur logique sûr de fonctionnement. La discussion est surtout centrée sur la vérification formelle dont l'amélioration de l'utilisation industrielle motive ce travail de thèse.

1 Contexte du travail

Le développement de systèmes automatisés industriels, surtout dans le domaine des industries critiques (secteur chimique, énergie nucléaire, aéronautique, etc.), exige de vérifier que leur fonctionnement soit celui prévu non seulement dans des conditions nominales, mais également dans certaines situations dégradées. Si une défaillance se produit, ses conséquences

doivent être de faible importance sur le service fourni par ces systèmes. Il existe donc un besoin industriel réel de concevoir des systèmes automatisés sûrs de fonctionnement.

Un système automatisé est composé d'un processus contrôlé et d'un contrôleur (figure 1.1). Le processus contrôlé est appelé aussi partie opérative. Il est composé de capteurs, de pré-actionneurs, d'actionneurs et de terminaux de dialogue opérateur. Le contrôleur est destiné à coordonner les différentes actions de la partie opérative. Il envoie des ordres (variables de sortie) vers les pré-actionneurs à partir des données d'entrée (variables d'entrée provenant des capteurs ou de pupitres opérateur), de consignes et d'un logiciel de commande. Les sorties et les entrées du contrôleur sont des signaux logiques, car nous nous limitons à l'étude des contrôleurs logiques.

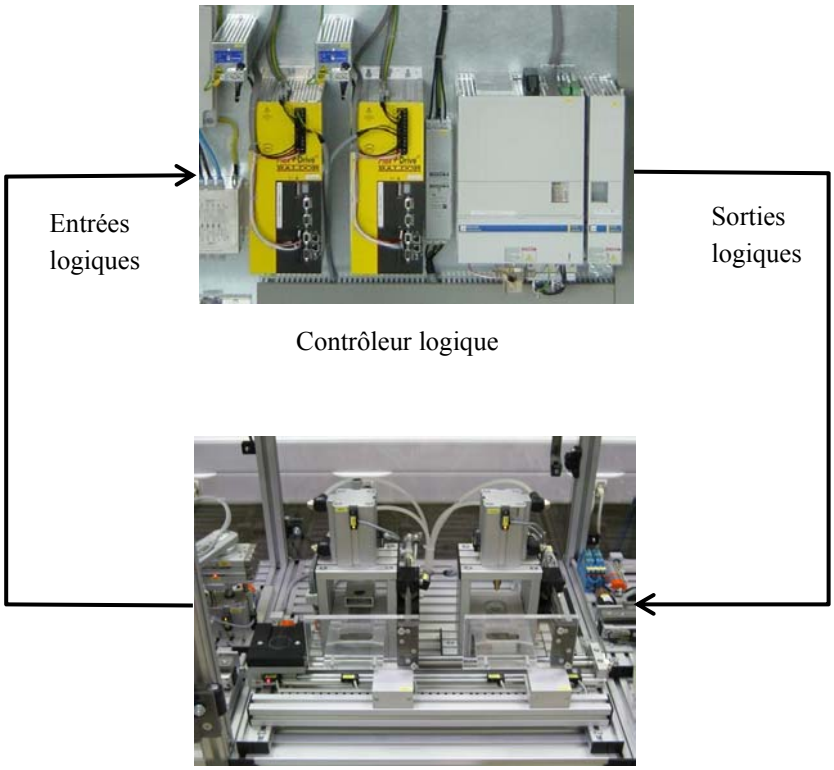
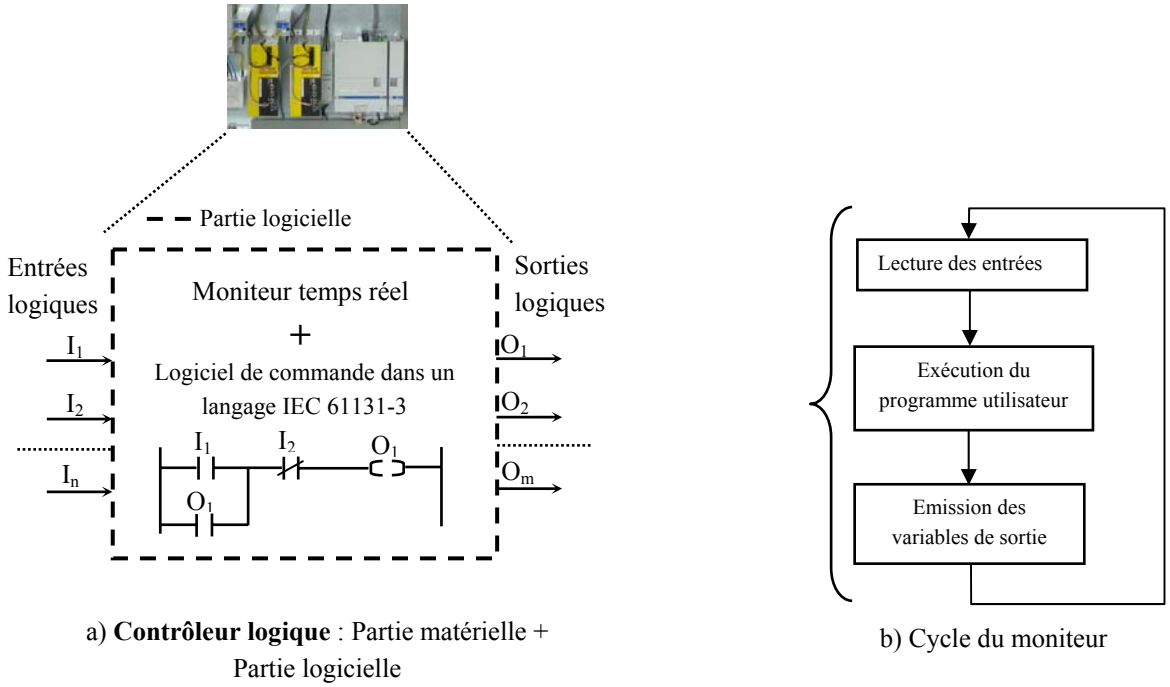


Figure 1.1. Système automatisé : contrôleur logique + partie opérative

Le logiciel de commande implanté dans le contrôleur logique est écrit dans un langage normalisé (IEC 61131-3, 1993), et s'exécute sous le contrôle d'un moniteur temps réel. Dans tout ce qui suit, nous faisons l'hypothèse que ce moniteur est mono-tâche, à scrutation cyclique ou périodique des entrées. Il réalise alors un cycle (non forcément périodique) à trois étapes :

- lecture des variables d'entrée,
- exécution du programme utilisateur,
- émission des variables de sortie.

Un contrôleur logique comporte donc une partie matérielle et une partie logicielle (figure 1.2.a). Cette partie logicielle peut elle-même être décomposée en deux parties : moniteur temps réel (figure 1.2.b) et programme utilisateur. Dans ce qui suit, on supposera que la partie matérielle et le moniteur sont sans fautes ; seules les fautes du programme utilisateur seront étudiées.



a) **Contrôleur logique** : Partie matérielle + Partie logicielle

b) Cycle du moniteur

Figure 1.2. Structure d'un contrôleur logique

Pour concevoir un système automatisé sûr de fonctionnement, il faut prendre en compte non seulement la partie opérative du système, mais aussi le contrôleur logique, car sa fonction est très importante. En effet, un contrôleur logique défaillant peut entraîner un service incorrect du système (Faure et Lesage, 2001).

Avant d'aborder en détail la conception d'un contrôleur logique sûr de fonctionnement, la section suivante introduit les principaux concepts de la sûreté de fonctionnement des systèmes.

2 La sûreté de fonctionnement des systèmes

Les notions définies ci-dessous, sont extraites des ouvrages de J.C. Laprie (Laprie et al., 1989 ; Laprie, 2004). La sûreté de fonctionnement (SdF) est la propriété d'un système caractérisant la confiance que placent ses utilisateurs dans le service qu'il leur délivre. Le service délivré par un système est son comportement tel que perçu par ses utilisateurs. La sûreté de fonctionnement est donc un concept général. Afin de préciser ce concept, l'arbre de la sûreté de fonctionnement, qui est formé de ses attributs, de ses entraves et de ses moyens (figure 1.3), est présenté ci-après.

2.1 Les attributs

Les attributs de la SdF sont la fiabilité, la sécurité, la disponibilité et la maintenabilité.

- **Fiabilité** : elle permet de mesurer la délivrance continue d'un service correct ou, ce qui est équivalent, caractérise le temps jusqu'à défaillance à partir de la mise en service ;
- **Sécurité** : c'est l'absence de conséquences catastrophiques pour l'environnement, les opérateurs, les biens ;
- **Disponibilité** : c'est le fait d'être prêt à l'utilisation ou le pourcentage de temps où le service est assuré ;
- **Maintenabilité** : c'est l'aptitude aux réparations et aux évolutions techniques.

Fiabilité et disponibilité sont des attributs quantifiables, ce qui n'est pas le cas des deux autres attributs.

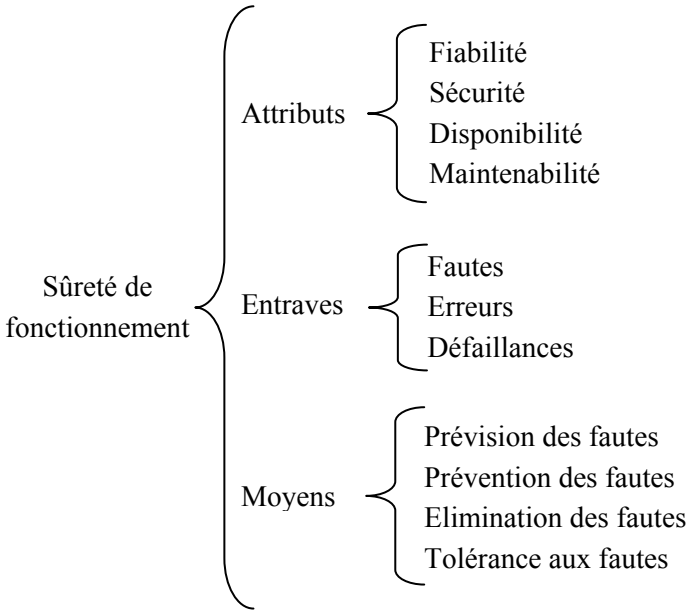


Figure 1.3. L'arbre de la sûreté de fonctionnement

2.2 Les entraves

On a énoncé précédemment que la SdF d'un système est la propriété caractérisant la confiance que placent ses utilisateurs dans le service qu'il leur délivre. Une défaillance (absence de service) est la manifestation d'une **erreur** sur le service délivré par le système, alors qu'une erreur est la manifestation d'une **faute** dans le système. On peut indiquer ici en résumé que les mécanismes de création et de manifestation des fautes, erreurs et défaillances permettent de compléter la chaîne fondamentale donnée par la figure 1.4. Les flèches de la chaîne expriment des relations causales entre fautes, erreurs et défaillances. En voici deux exemples :

- défaillance d'un programmeur →faute dormante dans le module logiciel qu'il a développé (instruction erronée) → erreur à l'exécution (valeurs fausses générées par le logiciel) →défaillance du logiciel.

- court-circuit dans un composant (défaillance du composant) → faute sur la carte →erreur dans les signaux générés → défaillance de la carte.

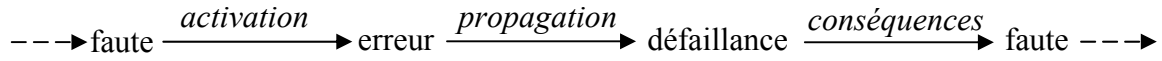


Figure 1.4. La chaîne fondamentale des entraves à la SdF

2.3 Les moyens

Les moyens pour l'amélioration de la sûreté de fonctionnement sont :

- **la prévision des fautes** : quelles sont les fautes qui peuvent se produire et quelles en sont les conséquences ?
- **la prévention des fautes** : comment empêcher, par construction, l'occurrence ou l'introduction de fautes ?
- **l'élimination des fautes** : comment éliminer les fautes préalablement détectées ?
- **la tolérance aux fautes** : comment délivrer, par redondance structurelle ou fonctionnelle, un service conforme à la spécification en dépit des fautes ?

Dans le cadre de cette thèse, nous nous sommes intéressés uniquement à la prévision et à l'élimination des fautes du programme utilisateur d'un contrôleur logique. La première catégorie est détaillée dans la sous-section suivante. La section 4 est consacrée à la présentation des techniques de vérification formelle, qui permettent de détecter des fautes du programme utilisateur qui pourront être éliminées par la suite.

2.4 La prévision des fautes

Les méthodes de prévision des fautes peuvent être classées en deux catégories : les méthodes d'évaluation ordinale et les méthodes d'évaluation probabiliste. L'objectif des méthodes de la première catégorie est d'établir une relation d'ordre partiel entre défaillances, de déterminer les causes potentielles d'une défaillance et les conséquences d'une faute. Les méthodes les plus usuelles sont :

- l'Analyse des Modes de Défaillance et de leurs Effets (AMDE) et l'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité (AMDEC) ;
- l'Analyse des Effets des Erreurs du Logiciel (AEEL) ;
- l'HAZOP (HAZard and OPerability analysis) : variante de l'AMDEC pour les processus continus ;
- l'Arbre des Défaillances (en anglais Fault Tree Analysis).

Quant à l'évaluation probabiliste, l'objectif est de calculer les valeurs numériques d'attributs de surêté (fiabilité, disponibilité). Les méthodes les plus usuelles sont :

- la méthode du diagramme de succès (ou diagramme de fiabilité) ;

- le calcul à partir de l'Arbre des Défaillances ;
- l'analyse de modèles à états stochastiques (processus ou chaînes de Markov, réseaux de Petri stochastiques) qui représentent le système.

Parmi l'ensemble de ces méthodes, nous nous focalisons maintenant sur les méthodes d'analyse ordinale suivantes : l'Arbre des Défaillances, l'AMDEC et l'AEEL. Les deux premières méthodes sont utilisées dans notre travail. L'AEEL n'est pas appliquée dans notre approche, mais elle est une adaptation de l'AMDEC au niveau logiciel et nous trouvons pertinente sa présentation.

2.4.1 Analyse par Arbre des Défaillances

L'analyse par Arbre des Défaillances (AdD) est une méthode de type déductif. Il s'agit, à partir d'un Evénement Indésirable (EI) **défini a priori**, de déterminer la combinaison d'événements pouvant conduire à cet EI. Cette analyse permet de remonter de causes en causes jusqu'aux événements de base susceptibles d'être à l'origine de l'événement indésirable. Les liens entre les différents événements identifiés sont réalisés grâce à des portes logiques (de type « ET » et « OU » par exemple). Cette méthode utilise une symbolique graphique particulière qui permet de présenter les résultats dans une structure arborescente. Etant donnée l'importance de cette technique dans le cadre de cette thèse, elle est largement détaillée dans le deuxième chapitre.

2.4.2 L'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité

La méthode de l'AMDEC est une suite logique de l'Analyse des Modes de Défaillance et de leurs Effets (AMDE). Elle indique de plus la criticité des défaillances. Le but de cette méthode est d'étudier l'impact d'une défaillance d'un composant physique ou fonctionnel sur le service fourni par le système. L'AMDE et l'AMDEC sont définies par la norme internationale (CEI 60812, 2006). Une analyse AMDEC s'effectue en quatre étapes :

1. **Définition du système, de ses fonctions et de ses composants**. L'AMDEC nécessite une définition précise du système à étudier et de sa structure fonctionnelle et/ou physique.
2. **Etablissement des modes de défaillance** des composants. Le mode de défaillance d'un composant est défini comme l'effet par lequel une défaillance de ce composant est observée. Par exemple, la défaillance d'un moteur est observée car il ne démarre pas. Une liste des principaux modes de défaillance est présentée dans le tableau 1.1.
3. **Etude des effets des modes de défaillance**. Pour chaque mode de défaillance, le comportement du système est observé composant par composant, c'est-à-dire qu'un mode de défaillance est fixé sur un certain composant pendant que les autres sont en fonctionnement ou en état de fonctionner.

4. **Conclusions, recommandations**. Des conclusions peuvent être tirées par rapport aux objectifs de l'étude et des recommandations utiles sont émises, par exemple : modification d'architecture (introduction des redondances), mise en place de capteurs, mise en place de procédures de surveillance, d'opérations de maintenance, formation d'opérateurs, etc. Les informations obtenues dans l'analyse, sont présentées dans un tableau récapitulatif (voir par exemple le tableau 1.2).

1. Défaillance structurelle (rupture)	18. Mise en marche erronée
2. Blocage physique au coincement	19. Ne s'arrête pas
3. Vibration	20. Ne démarre pas
4. Ne reste pas en position	21. Ne commute pas
5. Ne s'ouvre pas	22. Fonctionnement prématuré
6. Ne se ferme pas	23. Fonctionnement après le délai prévu (retard)
7. Défaillance en position ouverte	24. Entrée erronée (augmentation)
8. Défaillance en position fermée	25. Entrée erronée (diminution)
9. Fuite externe	26. Sortie erronée (augmentation)
10. Fuite interne	27. Sortie erronée (diminution)
11. Dépasse la limite supérieure tolérée	28. Perte de l'entrée
12. Est en dessous de la limite inférieure tolérée	29. Perte de la sortie
13. Fonctionnement intempestif	30. Court-circuit (électrique)
14. Fonctionnement intermittent	31. Circuit ouvert (électrique)
15. Fonctionnement irrégulier	32. Fuite (électrique)
16. Indication erronée	33. Autres conditions de défaillance …
17. Ecoulement réduit	

Tableau 1.1. Modes de défaillance des composants

Voyons maintenant un exemple qui illustre la méthode de l'AMDEC : il s'agit d'un système hydraulique de distribution à deux pompes en redondance passive (figure 1.5) avec une pompe assurant le fonctionnement normal et l'autre pompe en secours. Les cinq vannes du système sont du type TOR. Cet exemple montre l'analyse AMDEC en se limitant à la vanne V2 à l'aide du tableau 1.2. Ce tableau suppose que trois niveaux de criticité (faible/moyenne/forte) ont été définis.

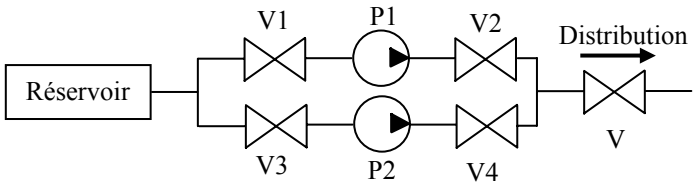


Figure 1.5. Système de distribution à deux pompes en redondance passive

Le tableau contient toute l'information concernant le composant, c'est-à-dire les modes de défaillance (extraits de la liste du tableau 1.1), les effets produits sur le système global, la criticité et des observations.

Identification du composant	Modes de défaillance	Effets sur système	Criticité	Observations (moyens de détection, ...)
Vanne V2	Ne s'ouvre pas	Impossibilité de distribution	Faible si P1 «normal» Forte si P1 «secours»	Introduction de capteur débit ou position vanne
	Ne se ferme pas	Distribution non interruptible	Faible car coupure de distribution possible par la vanne générale	
	Fuites externes	Débit de distribution plus faible	Moyenne	
		Liquide déversé dans la station	Forte	

Tableau 1.2. Exemple d'un tableau d'AMDEC pour la vanne V2

2.4.3 Analyse des Effets des Erreurs du Logiciel

L'AEEL est une adaptation de l'AMDEC au niveau du logiciel. On parle alors d'erreur du logiciel et non pas de mode de défaillance. Elle s'applique dans la phase de conception d'un logiciel. Le but est d'étudier l'impact d'une erreur dans un élément d'un logiciel sur le fonctionnement global de ce dernier. L'élément à analyser est appelé module. Un module est une partie d'un logiciel, qui peut être compilée, testée et archivée séparément. Il interagit avec les autres modules au travers de partages de ressources logicielles (fichiers, base de données, etc) ou d'échanges de paramètres via des canaux de communications. Les erreurs considérées sont classées en six grandes catégories (Metge, 1996) :

- les erreurs de calcul ;
- les erreurs d'algorithmique ;
- les erreurs sur les données traitées ;
- les erreurs de synchronisation entre tâches ;
- les erreurs d'interface entre procédures ;
- les erreurs de transfert de données avec l'environnement.

Le détail de ces erreurs est donné dans le tableau 1.3.

Types d'erreurs logicielles	Classes d'erreurs utilisées	Exemples de défauts originels
Erreur de calcul	Evaluation d'une équation incorrecte	Choix d'opérande erroné
		Valeur d'opérande interdite par un opérateur
		Choix d'opérateur incorrect
		Fonction d'un opérateur non assurée
		Omission dans les calculs

		Opérateur à supprimer
	Résultat erroné d'une opération	Dépassement de capacité
		Troncature d'une valeur
		Précision insuffisante
Erreur d'algorithmique	Erreur de séquencement	Ordre erroné des instructions
		Instruction omise
		Instruction à supprimer
		Séquence inaccessible
	Branchement inconditionnel erroné	Implantation du branchement erronée
		Destination du branchement erronée
Erreur sur les données traitées	Erreur de définition	Déclaration de structure erronée
		Déclaration manquante
		Déclaration multiple
		Mauvaise implantation de la déclaration
Erreur de synchronisation entre tâches	Nature de la primitive de synchronisation	Défaut de modélisation
	Paramètre de synchronisation erroné	Défaut de modélisation
	Synchronisation non prévue	Défaut de modélisation
Erreur d'interface entre procédures	Erreurs d'appel de la procédure	Implantation de l'instruction d'appel de procédure erronée
	Erreur de sortie d'une procédure	Implantation de l'instruction de sortie de procédure erronée
Erreur de transfert de données avec l'environnement	Erreurs de définition de données	Déclaration de structuration erronée
		Déclaration de type erronée
	Erreurs de transmission de données	Quantité de données transférées erronée

Tableau 1.3. Critères de l'AEEL

Pour appliquer l'AEEL, il faut alors connaître en détail le logiciel à analyser. C'est donc une méthode de type « boîte blanche ». Notre objectif est d'améliorer la sûreté des programmes utilisateurs des contrôleurs logiques sans avoir à en connaître le détail des lignes de code. Nous souhaitons donc adopter une approche « boîte noire » lors de l'analyse de sûreté. Ceci explique pourquoi nous ne retenons pas l'AEEL pour nos travaux.

Pour conclure cette section, nous soulignons que nous venons d'évoquer rapidement les principaux concepts de la sûreté de fonctionnement des systèmes. De façon particulière, ce qui nous intéresse est le développement de contrôleurs logiques sûrs de fonctionnement. Ce point est abordé dans ce qui suit.

3 Le développement d'un contrôleur logique sûr de fonctionnement

Il est possible aussi de classifier les méthodes ou moyens pour l'amélioration de la sûreté de fonctionnement en prenant en compte le critère de cycle de vie des systèmes (Faure et Lesage,

2001). Avec ce point de vue, les méthodes qui contribuent à la sûreté peuvent être classifiées en deux catégories : méthodes pour la sûreté **hors ligne** et méthodes pour la sûreté **en ligne**. Les méthodes de la première catégorie ont pour but de minimiser le risque de fautes durant le développement du système, c'est-à-dire avant que le système soit employé. A l'opposé, les méthodes de la deuxième catégorie permettent d'assurer la sûreté d'un système existant lors de son exploitation. Le diagnostic et le pronostic de dysfonctionnement sont des exemples de telles méthodes.

L'analyse de sûreté d'un contrôleur logique peut donc se faire en considérant le critère du cycle de vie. Ce cycle comprend globalement deux phases (figure 1.6) :

- le développement, qui comprend les étapes de spécification, de conception, d'implantation, de test et de validation fonctionnelle globale (VFG) ;
- l'exploitation, au cours de laquelle le contrôleur logique est utilisé pour la commande d'un processus physique.

Le cycle de vie introduit également deux concepts importants : la vérification et la validation des modèles de conception et d'implantation.

- La vérification a pour objectif le contrôle des propriétés intrinsèques (stabilité, vivacité, absence de blocage, …) du modèle développé lors d'une étape. Elle peut être définie par la question « réalisons-nous bien le produit ? ». La vérification constitue un préalable indispensable à la validation.
- La validation se fait à la fin d'une étape, après avoir vérifié le résultat de cette étape. Elle assure le contrôle de la cohérence entre le résultat de cette étape et celui de l'étape antérieure. La validation détermine si le modèle est conforme aux besoins et peut être définie par la question « réalisons-nous le bon produit ? ».

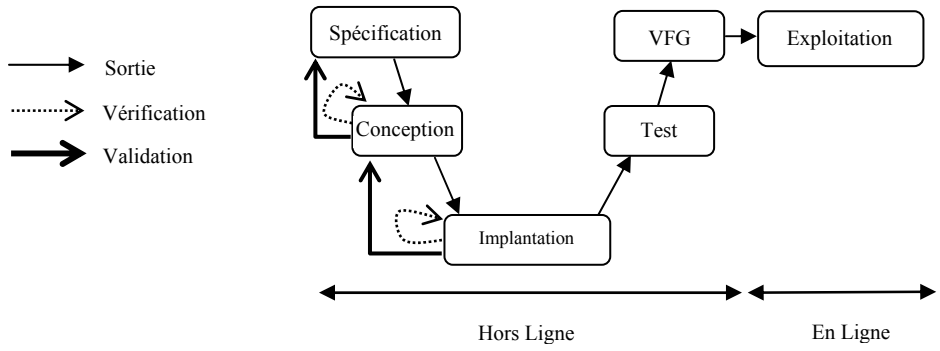


Figure 1.6. Vérification et validation de la conception et de l'implantation

Afin de vérifier et de valider des modèles de conception et d'implantation, il est très intéressant d'utiliser des techniques de vérification formelle (Lampérière-Couffin et al. 1999). La vérification formelle permet de vérifier, **de manière exhaustive**, si un modèle donné satisfait, ou ne satisfait pas, des propriétés. Elle est un pré-requis à l'élimination des fautes car

elle permet de détecter des fautes qui pourront être éliminées par la suite. Les méthodes de vérification formelle peuvent être rangées dans l'un ou l'autre des classes suivantes : le theorem proving et le model-checking.

Le theorem proving

Dans ces méthodes, le système et les propriétés à vérifier sont exprimés par des formules dans une logique mathématique (modèles algébriques). Un système formel est construit par la définition d'un ensemble d'axiomes et de règles d'inférence. La preuve consiste à trouver les axiomes des propriétés à partir des axiomes du système, en se basant sur les règles d'inférence, (Rushby, 2000). Ces méthodes présentent l'avantage d'éviter l'explosion combinatoire ; cependant, seules certaines classes de propriétés sont vérifiables, et la mise en œuvre de ce type de méthode requiert l'intervention de spécialistes.

Le model-checking

Ces méthodes sont basées sur la théorie des systèmes à événements discrets et ont pour principe de vérifier si un modèle à états du système satisfait ou non une propriété formelle. L'analyse consiste alors en une exploration exhaustive de l'espace d'états. Il existe des outils informatiques, qualifiés de model-checkers, tels que SMV, NuSMV, UPPAAL, qui, une fois construits les modèles formels du système et des propriétés, permettent d'automatiser la vérification. D'après l'expérience d'utilisation au LURPA, nous choisissons le model-checking dans le cadre de notre travail. La section suivante est alors consacrée à une présentation des bases de cette méthode.

4 Vérification formelle de contrôleurs logiques par model-checking

La vérification par model-checking (Schnoebelen, et al. 1999) est une technique qui permet de détecter des fautes dans un système modélisé sous la forme d'un modèle à états discrets. La détection des fautes permet au concepteur du programme utilisateur de corriger (éliminer) ces fautes. La vérification par model-checking s'applique hors ligne et peut être considérée comme un pré-requis à l'élimination des fautes. Pour vérifier automatiquement un contrôleur logique par cette méthode, illustrée dans la figure 1.7, il est nécessaire de construire une représentation formelle du contrôleur, par exemple sous la forme d'un automate à états (à gauche dans la figure).

Il faut ensuite énoncer formellement les propriétés à vérifier, à partir des spécifications exprimées de façon informelle, c'est-à-dire avec des phrases en langage courant ou avec des schémas (partie droite de la figure). On utilise à cette fin un langage de spécification de propriétés, par exemple une logique temporelle. Il est également possible d'exprimer les propriétés à l'aide d'automates à états qui ont le rôle d'observateurs et de formules en logique temporelle portant sur un ou plusieurs états de ces automates observateurs.

Enfin, il faut disposer d'un algorithme capable de dire si le système vérifie ou non les propriétés énoncées. Cet algorithme est implanté dans un model-checker : un outil informatique pour le model-checking. La plupart des model-ckeckers sont capables de donner un diagnostic d'erreur lorsqu'une propriété n'est pas vérifiée (Schnoebelen, et al. 1999) sous la forme d'un exemple d'exécution qui ne satisfait pas la propriété.

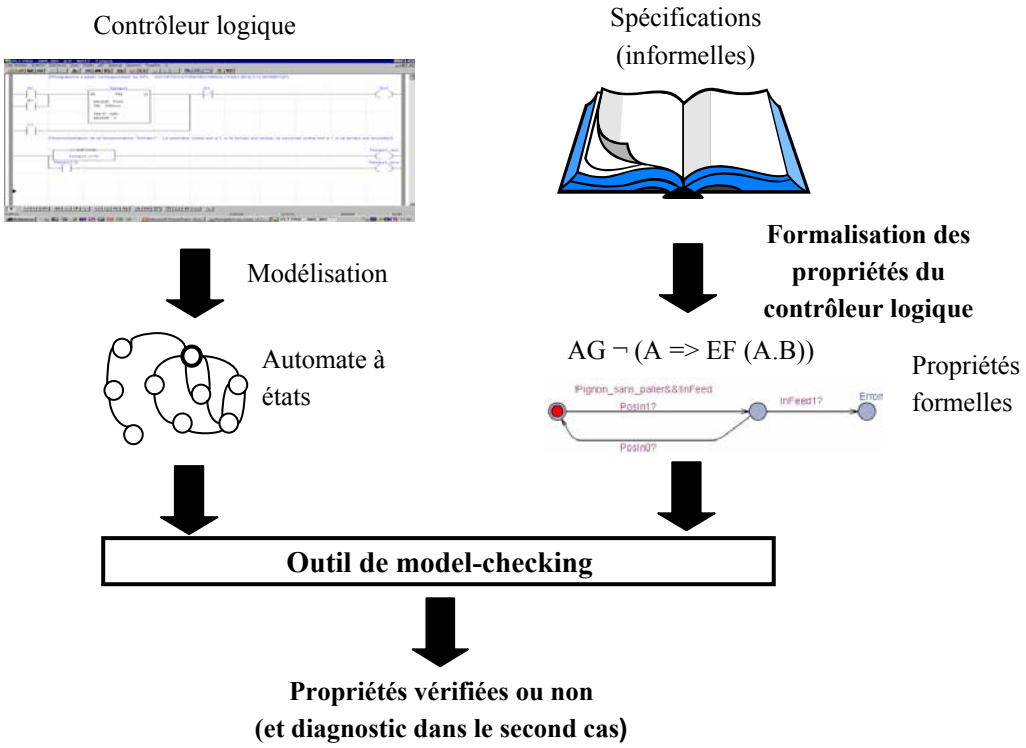


Figure 1.7. Le principe du model-checking de contrôleur logique

Le model-checking peut être non temporisé ou temporisé. Le model-checking non temporisé permet la vérification de propriétés faisant intervenir le **temps logique** (relation d'ordre total ou partiel entre événements). Le model-checking temporisé permet la vérification de propriétés temps réel, c'est à dire faisant intervenir de façon explicite le **temps physique**.

4.1 Le modèle formel du système à vérifier

Il a été mentionné précédemment que pour vérifier automatiquement un contrôleur logique par model-checking, il est nécessaire d'en construire un modèle formel. La modélisation peut concerner seulement le contrôleur logique, ce qui est connu sous le nom d'approche « non model-based », ou bien considérer dans l'analyse, en plus du modèle du contrôleur, un modèle de la partie opérative, ce que l'on appelle une approche « model-based ». Frey et Litz (Frey et Litz, 2000), recense de nombreux travaux scientifiques classifiés dans ces deux catégories. Développés au sein du LURPA, plusieurs travaux (Lampérière-Couffin et al. 1999 ; De Smet et Rossi, 2002 ; Rossi, 2003) abordant la vérification formelle de programmes d'Automates Programmables Industriels (API's), sont des exemples de la première catégorie. Un résultat

important de ces travaux scientifiques est une définition de la sémantique formelle des langages de la norme IEC 61131-3 : SFC, LD, IL, ST. Le travail de Machado (Machado et al. 2003), est un exemple de la deuxième catégorie.

4.2 Les propriétés à vérifier

4.2.1 Classification des propriétés

Les propriétés peuvent être classifiées selon les objectifs de vérification (Schnoebelen et al. 1999), en propriétés d'atteignabilité, propriétés de sûreté, propriétés de vivacité et propriétés d'équité.

Une ***propriété d'atteignabilité*** énonce qu'un certain état de l'automate modélisant le contrôleur logique *peut être atteint* à partir d'un état initial.

Une ***propriété de sûreté*** énonce que, sous certaines conditions, un état indésirable *ne se produit jamais*. Il convient de signaler que, dans ce contexte du model-checking , sûreté a une signification différente de celle utilisée en sûreté de fonctionnement. Les propriétés de sûreté (au sens du model-checking) correspondent souvent à des analyses de sécurité (au sens de la sûreté de fonctionnement).

Une ***propriété de vivacité*** énonce que, sous certaines conditions, un état *finira par avoir lieu*.

Une ***propriété d'équité*** énonce que, sous certaines conditions, un état aura lieu (ou n'aura pas lieu) *un nombre infini de fois*.

4.2.2 Expression des propriétés en logique temporelle CTL

La logique temporelle est une forme de logique spécialisée dans les énoncés et raisonnements faisant intervenir la notion d'ordre dans le temps logique. Dans le cadre du travail de cette thèse, et d'après l'expérience des cas traités au LURPA (De Smet et Rossi, 2002 ; Rossi, 2003, Gourcuff et al. 2006), on utilisera la logique CTL (pour Computational Tree Logic). La logique CTL, comme les autres logiques temporelles utilisées dans les outils de model-checking, sert à énoncer formellement des propriétés portant sur les exécutions d'un système, c'est-à-dire qu'elle s'intéresse à des chemins composés d'états représentatifs du système. Toutes les définitions de cette sous-section sont extraites de l'ouvrage de Schnoebelen (Schnoebelen, et al. 1999). Dans CTL, sont utilisés :

- des propositions logiques qualifiant les états ;
- les constantes *true* et *false* ;
- les opérateurs booléens : la négation $\neg$, la conjonction $\wedge$, la disjonction $\vee$, l'implication logique $\Rightarrow$, et la double implication $\Leftrightarrow$;
- des opérateurs temporels qui permettent de parler de l'enchaînement des états le long d'une exécution (suite d'états), et non plus d'états considérés individuellement. Les opérateurs les plus simples sont :
  - **G** (globally), permet d'énoncer qu'une proposition est vraie dans tous les états d'un chemin**,**

- **F** (futur), permet d'énoncer qu'une proposition est vraie dans un état futur d'un chemin,
- **X** (next), énonce qu'une proposition sera vraie dans l'état suivant,
- **U** (until), énonce qu'une proposition est vraie jusqu'à ce qu'une autre le soit. Par exemple, *P* **U** *Q* énonce que *P* est vérifiée jusqu'à ce que *Q* le soit.
- **W** (weak until), énonce qu'une proposition sera vérifiée tant qu'une autre ne l'est pas. Dans *P* **W** *Q*, on peut lire *P* sera vérifiée tant que *Q* ne l'est pas. On peut dire qu'avec **W** on exprime encore la notion de **U**, mais la différence réside dans le fait qu'il n'est pas exigé que *Q* finisse par avoir lieu (et si *Q* n'a jamais lieu, alors *P* reste vraie jusqu'à la fin).

— des quantificateurs de chemins qui permettent de quantifier sur l'ensemble des exécutions. On exprime ainsi le coté arborescent du comportement (plusieurs futurs sont possibles à partir d'une situation donnée).

- **A** (always) indique que la proposition est vérifiée par tous les chemins débutant à l'état courant.
- **E** (existe) indique que la proposition est vérifiée dans au moins un chemin débutant à l'état courant.

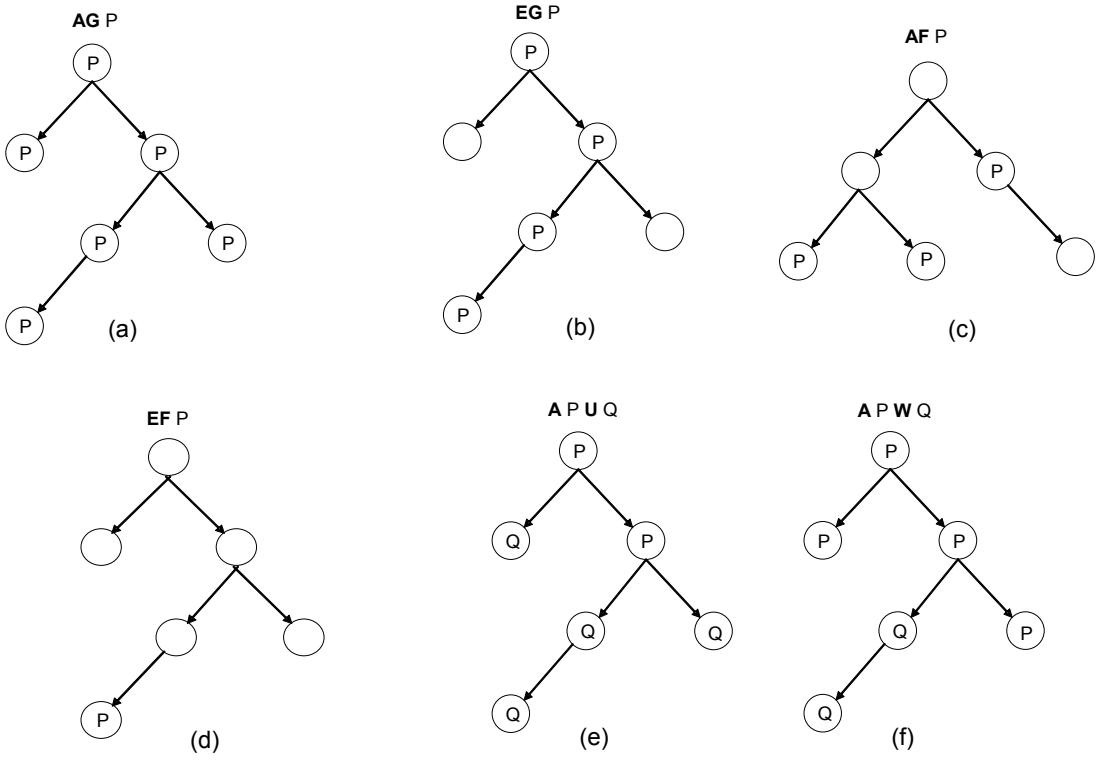


Figure 1.8. Représentation d'expressions courantes en logique temporelle CTL

Les opérateurs et les quantificateurs s'utilisent souvent par paire. Par exemple, **AG** *P* signifie que *P* est toujours vraie (figure 1.8.a) tandis que **EG** *P* dit qu'il existe une exécution au cours de laquelle *P* reste toujours vraie (figure 1.8.b).

AF *P* signifie que l'on aura forcément, quelle que soit l'exécution retenue, au moins un état où *P* sera vraie (figure 1.8.c). **EF** *P* signifie qu'il est possible (en suivant une des exécutions) d'avoir un état où *P* sera vraie (figure 1.8.d). **A** *P* **U** *Q* dit que sur tous les chemins, *P* est vraie jusqu'à ce que Q soit vraie (figure 1.8.e). **A** *P* **W** *Q* énonce que sur tous les chemins, *P* sera vraie tant que *Q* ne l'est pas (figure 1.8.f). Dans CTL, chaque utilisation d'un opérateur temporel (G, X, F, U, W) est immédiatement sous la portée d'un quantificateur **A** ou **E**.

Pour illustrer ces concepts, reprenons maintenant le cas du système hydraulique de distribution à deux pompes en redondance passive, qui a été utilisé dans la sous section 2.4.2. On peut exprimer en CTL les propriétés à vérifier pour ce système. Pour cet exemple, on reprend la configuration complète du système, originalement introduite par Roussel et Denis (Roussel et Denis, 2002). On montre le contrôleur logique qui commande le système, avec la liste de ses entrées et de ses sorties (figure 1.9). A titre d'exemple, trois propriétés simples du contrôleur logique seront formalisées :

Propriété 1 : les deux pompes ne doivent jamais fonctionner en même temps.

$$AG \neg (L1\_pompe \wedge L2\_pompe)$$

C'est-à-dire, il est toujours vrai (AG), que l'on n'aura jamais (¬) les commandes de deux pompes vraies simultanément.

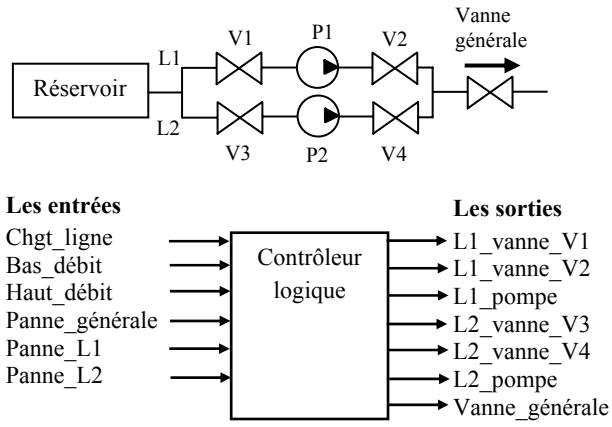


Figure 1.9. Système hydraulique et son contrôleur logique

Propriété 2 : une panne générale provoque l'arrêt des pompes et la fermeture des vannes.

$$AG\ (Panne\_générale \rightarrow \neg L1\_pompe \wedge \neg L1\_vanne\_V1 \wedge \neg L1\_vanne\_V2 \wedge \neg L2\_pompe \wedge \neg L2\_vanne\_V3 \wedge \neg L2\_vanne\_V4 \wedge \neg Vanne\_générale)$$

Il est toujours vrai (AG) que si Panne_générale est vraie, alors on n'aura jamais (¬) les commandes des pompes et des vannes vraies.

Propriété 3 : la mise en route d'une pompe ne peut se faire qu'après avoir ouvert la vanne en amont.

A¬ L1_pompe W (L1_vanne_V1)
A¬ L2_pompe W (L2_vanne_V3)

C'est-à-dire, quelque soit l'exécution (A), une pompe ne se met en route jamais (¬) sauf si on ouvre la vanne en amont.

4.2.3 Expression des propriétés à l'aide d'automates observateurs

Un autre moyen pour exprimer les propriétés formelles du contrôleur consiste à introduire des automates à états finis qui ont le rôle d'observateurs. Un automate observateur est un automate chargé de détecter l'occurrence d'événements sans effectuer aucun changement au système observé. C'est un automate qui évolue en parallèle avec le modèle du système observé. Introduire des automates observateurs permet d'exprimer les propriétés formelles d'une façon plus simple, surtout quand les expressions en logique temporelle correspondant aux propriétés à vérifier sont longues et compliquées.

5 Problématique : difficulté de mise en œuvre du model-checking

5.1 Obstacles à son utilisation industrielle

La mise en place de la technique du model-checking dans l'industrie présente certaines difficultés qui, bien que non suffisantes pour mettre en doute sa validité et son utilité, représentent un obstacle pour une utilisation plus large. Pourtant, il existe une réelle nécessité de vérifier les contrôleurs logiques des systèmes automatisés. Cependant, les ingénieurs automaticiens préfèrent employer les techniques de simulation traditionnelles, même si elles sont souvent fastidieuses et non exhaustives, pour vérifier que les programmes développés respectent les conditions d'application. Plusieurs raisons peuvent expliquer cette situation (Gourcuff, et al. 2006) :

- l'expression formelle des propriétés en logique temporelle ou sous forme d'automates observateurs est une tâche difficile pour la plupart des ingénieurs ;
- en cas de preuve négative, les model-checkers fournissent des contre-exemples qui sont souvent difficiles à interpréter ;
- les fabricants de contrôleurs logiques industriels ne proposent pas de logiciel commercial capable de traduire automatiquement les programmes utilisateur dans des modèles formels ;
- l'explosion de l'espace d'états (explosion combinatoire), qui se produit en traitant des programmes de contrôle de grande taille, rend l'analyse impossible. Ceci découle du fait que le model-checking nécessite une analyse **exhaustive** de l'espace d'états de l'automate à vérifier.

L'objectif de cette thèse est de contribuer à la résolution du premier problème, c'est-à-dire la difficulté que pose l'expression formelle de propriétés à partir des spécifications. En effet, la

pratique courante dans l'industrie est d'exprimer les spécifications dans un langage informel, i.e. quelques phrases en langage naturel ou des schémas mais jamais avec des formules mathématiques.

5.2 Exemple de travaux visant à faciliter l'élaboration des propriétés formelles

La formalisation des propriétés à des fins de vérification formelle a été abordée par peu de travaux scientifiques. Dans cette sous-section nous allons présenter les deux principaux : les approches de Filkorn (Filkorn, 1999) et de Klein (Klein, 2001), qui est en fait une adaptation de la première.

Méthode des incompatibilités entre les variables d'entrée/sortie

Cette première approche propose la *Méthode des incompatibilités entre les variables d'entrée/sortie*. La méthode cherche les incompatibilités entre les entrées et les sorties du contrôleur à l'aide de tableaux, comme celui montré ci-dessous. Les sorties (O_1,..,O_i,..,O_n) sont écrites en colonnes. Les entrées (I_1,..I_i..,I_m) sont rangées en lignes.

	O_1= 0	O_i= 0	-----	O_n= 0
I_1	1		1	1
I_i		1	0	1
⋮				
I_m	1			

Tableau 1.4. Tableau générique des incompatibilités entrées/sorties

Le principe est le suivant :

- construire la liste de toutes les entrées et de toutes les sorties ;
- pour chaque variable de sortie, déterminer la combinaison (produit logique) de variables d'entrée nécessaire pour qu'elle soit à la valeur 0 (FAUX) ; si n combinaisons d'entrées contraignent une sortie, on introduit n colonnes pour cette sortie.

On obtient alors, à partir du tableau, des relations logiques qui expriment la mise à 0 des sorties. Par exemple, à partir du tableau 1.4 nous savons que la sortie O_1 doit valoir 0 pour la combinaison logique de $I_1 \wedge I_m$. A partir de ces relations, il est possible de générer les propriétés formelles, exprimées en logique temporelle. Nous prenons maintenant un exemple pour illustrer la méthode.

Le système considéré (figure 1.10), est un poste de perçage (Timking Electronics, 2000) composé de :

- un convoyeur linéaire, dont le mouvement est commandé par un moteur électrique et qui transporte la pièce à percer ;
- la broche (avec son moteur électrique pour la faire tourner), qui est montée dans une tête qui se déplace verticalement pour la monter ou la descendre. Le mouvement vertical est produit par un moteur électrique à deux sens de rotation. Deux détecteurs de fin de course servent à indiquer le position haute ou basse de la broche ;
- deux détecteurs de position, un pour indiquer la présence de la pièce à percer en début de ligne et l'autre pour indiquer sa présence au poste de perçage.

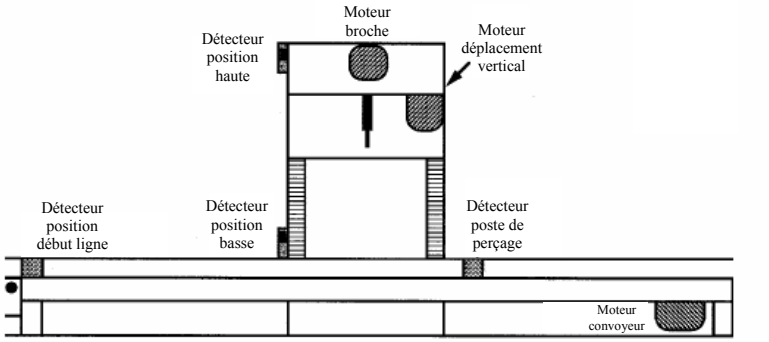


Figure 1.10. Poste de perçage

Le fonctionnement est le suivant : suite à l'arrivée d'une pièce en début de ligne (avec la broche en position haute) le convoyeur se met en marche pour l'avancer au poste de perçage. Une fois la pièce dans cette position, le moteur de la broche commence à tourner et la broche descend. Une fois en position basse, la rotation de la broche s'arrête et la broche commence à monter. Quand la broche est revenue en position haute, la pièce est évacuée et le cycle est fini. Il peut recommencer avec l'arrivée d'une autre pièce. Un contrôleur logique commande le fonctionnement automatique du système. Ses entrées-sorties sont listées dans la figure 1.11.

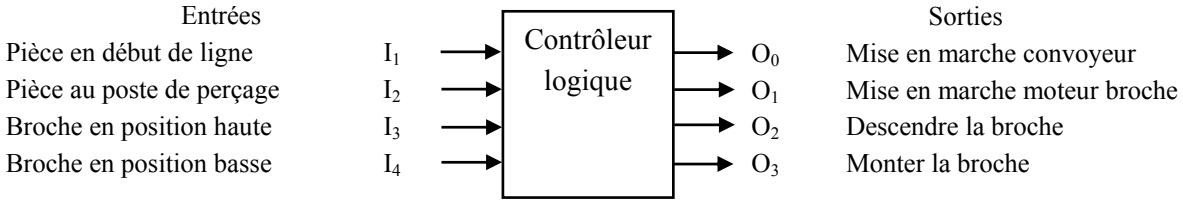


Figure 1.11. Entrées et sorties du contrôleur logique du poste de perçage

On construit alors le tableau 1.5 et, pour chaque variable de sortie, on détermine la combinaison de variables d'entrée nécessaire pour qu'elle soit à la valeur 0.

	$O_0 = 0$	$O_1 = 0$	$O_2 = 0$	$O_3 = 0$
I_1				
I_2		0	0	0
I_3			1	
I_4	1			1

Tableau 1.5. Tableau des incompatibilités entrées/sorties pour l'exemple

Par exemple, pour O_0 qui indique le mise en marche du convoyeur, cette sortie doit être fausse (indiquant que le convoyeur est arrêté) lorsque l'entrée I_4 est à 1 (indiquant que la broche est en position basse). Ceci est exprimé par :

$(I_4 = 1) \Rightarrow (O_0 = 0)$, le convoyeur doit être arrêté si la broche est en position basse.

On procède de la même façon avec les autres sorties et on obtient les conditions logiques suivantes :
$(I_2 = 0) \Rightarrow (O_1 = 0)$, ce qui exprime le fait que le moteur de broche ne tourne pas s'il n'y a pas de pièce en position sur le poste de perçage.
$(I_2 = 0 \wedge I_3 = 1) \Rightarrow (O_2 = 0)$, la broche ne descend pas si elle est en position haute et s'il n'y a pas de pièce en position sur le poste de perçage.
$(I_2 = 0 \wedge I_4 = 1) \Rightarrow (O_3 = 0)$, la broche ne monte pas si elle est en position basse et s'il n'y a pas de pièce en position sur le poste de perçage.

On peut maintenant déterminer les propriétés formelles à partir de ces implications logiques. Prenons le cas de la première relation. Elle peut être ainsi exprimée : on ne doit jamais avoir simultanément le convoyeur en marche et la broche qui est en position basse. C'est une propriété de sûreté qu'il est possible de formuler avec la logique temporelle CTL : dans toutes les exécutions (AG), il est vrai que l'on n'aura jamais ($\neg$) une mise en marche du convoyeur ($O_0 = 1$) si la broche est en position basse ($I_4 = 1$) : $AG \neg (O_0 \wedge I_4)$. Un raisonnement similaire peut être appliqué pour les autres relations. Par souci de concision, elles ne sont pas traitées ici.

Méthode des incompatibilités entre les sorties

Une adaptation de cette méthode est proposée par Klein (Klein, 2001). L'idée de base dans ce cas-là est de trouver les signaux de sortie incompatibles. A partir de la liste des signaux de sortie, un tableau est construit et les incompatibilités mutuelles sont recensées (ce qui est représenté par les signes « X » dans le tableau). Les sorties $(O_1,..,O_i,..,O_n)$ sont écrites en lignes et en colonnes. Le tableau est donc carré (tableau 1.6).

	O_1	O_i	-----	O_n
O_1	X			X
O_i		X		X
⋮				
O_n	X			

Tableau 1.6. Tableau générique des incompatibilités entre les sorties

Toujours dans notre exemple du poste de perçage, voici le tableau ainsi construit avec les signaux de sortie à 1.

	$O_0 = 1$	$O_1 = 1$	$O_2 = 1$	$O_3 = 1$
$O_0 = 1$		X	X	X
$O_1 = 1$				
$O_2 = 1$	X			X
$O_3 = 1$	X		X	

Tableau 1.7. Tableau d'incompatibilités entres les sorties du poste de perçage

Le tableau 1.7 exprime en particulier que $O_0 = 1$ est incompatible (ce qui est montré par les signes « X ») avec $O_2 = 1$ OU $O_3 = 1$. On procède de la même façon avec les autres sorties.

Cette relation peut être ainsi exprimée : on ne doit jamais descendre (O_2) ou monter (O_3) la broche quand le convoyeur (O_0) fonctionne. La relation peut alors être formulée, en termes de CTL, comme une propriété de sûreté. On écrit alors que dans toutes les exécutions (AG), il est vrai que l'on n'aura jamais ($\neg$) une montée ($O_3 = 1$) ou une descente ($O_2 = 1$) de la broche simultanément à un déplacement de la pièce ($O_0 = 1$). Ce qui se traduit en logique temporelle par :

$$AG \neg(O_0 \wedge O_2)$$
$$AG \neg(O_0 \wedge O_3)$$

Des propriétés similaires peuvent être déduites des autres colonnes du tableau.

Pour conclure cette sous-section, nous pouvons dire que les travaux que l'on vient d'exposer, quel que soit leur intérêt, présentent les inconvénients suivants :

- ces méthodes sont fastidieuses et consommatrices de temps pour des systèmes réels ;
- elles sont limitées à l'obtention de propriétés invariantes ;
- elles ne considèrent ni le temps logique, ni le temps physique ;
- elles ne sont pas liées à des méthodes industrielles existantes pour la sûreté de fonctionnement.

6 Objectif du travail de thèse

L'objectif de cette thèse est de faciliter l'obtention des propriétés formelles des contrôleurs logiques. Pour atteindre cet objectif, nous proposons de mettre au point une méthode qui s'appuie sur une autre technique connue et utilisée dans l'industrie pour la conception des systèmes critiques : la méthode de l'Arbre des Défaillances (AdD). L'un des intérêts d'utiliser l'Arbre des Défaillances est que les résultats de l'analyse sont montrés de façon graphique dans une structure arborescente.

La figure 1.12 schématise notre approche : nous proposons dans une première étape, de réaliser une analyse par AdD des possibles fautes du système. Ce qui nous intéresse dans cette analyse est de repérer et d'analyser les fautes qui sont dues au contrôleur logique. Le résultat de cette étape sera alors un AdD qui intégrera les fautes du contrôleur logique.

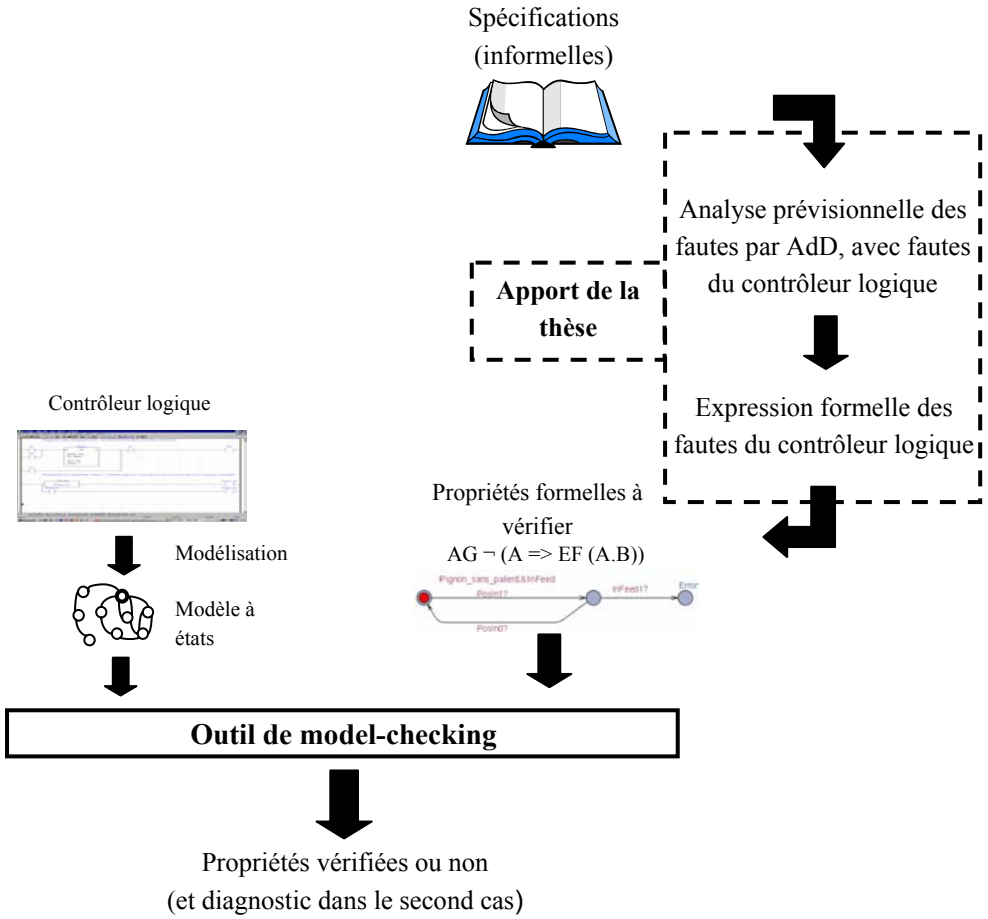


Figure 1.12. Objectif de la thèse

La deuxième étape consiste à exprimer de façon formelle, à partir de l'arbre résultant, les fautes du contrôleur analysées, ce qui permettra de décrire le comportement défaillant que l'on doit éviter pour avoir un contrôleur logique sûr de fonctionnement. Les propriétés

formelles devront alors exprimer le fait que les fautes du contrôleur ne doivent pas se produire.

Notre approche permettra donc d'établir **un lien entre les méthodes de prévision de fautes et les méthodes de détection de fautes des contrôleurs logiques**. Une limitation évidente de cette approche est que les propriétés formelles obtenues, déduites d'une analyse de défaillances, seront relatives au comportement dysfonctionnel du contrôleur ; les propriétés fonctionnelles du contrôleur devront être obtenues par un autre moyen. Etant donné l'importance des propriétés liées aux dysfonctionnements du contrôleur lors des études de sûreté, nous considérons que cette limitation est raisonnable.

Il convient enfin de souligner les fautes dans le programme utilisateur du contrôleur logique sont des **fautes systématiques**, reproductibles, et non des fautes aléatoires, comme il est usuel en AdD de systèmes physiques. Aborder dans notre travail les fautes systématiques nécessite de relever les deux challenges scientifiques suivants :

- **Les fautes systématiques,** qui donneront lieu à des expressions formelles en logique temporelle ou sous forme d'automates observateurs, **peuvent être fonction de l'ordre d'apparition d'événements ou du temps physique** et ne peuvent donc pas être représentées seulement par des expressions booléennes.
- **Elles doivent pouvoir être exprimées formellement** afin de pouvoir en déduire des propriétés formelles.

7 Conclusion

En conclusion, ce premier chapitre a permis de présenter en premier lieu le contexte de ce travail de thèse, c'est-à-dire la conception de contrôleurs logiques sûrs de fonctionnement, et en deuxième lieu la problématique abordée : l'obtention de propriétés formelles en vue de la vérification par model-checking.

L'objectif de l'approche que nous proposons est de faciliter cette tâche d'obtention des propriétés formelles. Nous avons vu dans ce chapitre que ce sujet avait été abordé par d'autres travaux de recherche et quelles étaient les limitations de ces travaux. L'approche que nous préconisons diffère car elle ne souhaite pas se limiter à des invariants, et elle est liée à une technique industrielle très utilisée pour les systèmes critiques : l'analyse prévisionnelle des fautes par Arbre des Défaillances. Etant donnée son importance dans notre approche, l'objectif du chapitre suivant est la présentation détaillée de cette méthode de prévision des fautes.

Chapitre 2.

Analyse prévisionnelle des fautes par Arbre des Défaillances

La méthode de l'Arbre des Défaillances (AdD), connue aussi comme Arbre de Pannes (CEI 61025, 1990) ou Arbres des Causes (INERIS, 2003) est, avec d'autres méthodes de prévision de fautes, largement connue et utilisée dans les études de sûreté de fonctionnement destinées à l'évaluation de la fiabilité des systèmes industriels. Depuis sa création en 1962, au sein de la société Bell (Limnios, 2005), pour évaluer et améliorer la fiabilité du système de lancement du missile Minuteman, cette technique a connu diverses évolutions et a été et continue d'être le sujet d'un grand volume des travaux techniques et scientifiques, qui ont contribué à la rendre plus efficace et qui ont visé à améliorer sa capacité d'expression, d'évaluation et même sa formalisation.

Ce deuxième chapitre comporte trois parties :

- la présentation des concepts de base de la méthode de l'AdD : ses éléments, la synthèse et l'analyse d'un arbre ;
- un état de l'art des principaux travaux scientifiques portant sur les évolutions et améliorations à la méthode ;
- une présentation succincte de nos propositions.

1 La méthode de l'Arbre des Défaillances

Cette première partie du chapitre est basée sur plusieurs documents et ouvrages qui présentent des concepts similaires, notamment : (US NR Commission, 1981 ; Andrews, 2002 ; Dutuit et al. 2002 ; INERIS, 2003 ; Limnios, 2005 ; et CEI 61025, 1990).

1.1 Concepts de base

L'analyse par AdD est une méthode de type déductif qui est utilisée pour l'analyse prévisionnelle des défaillances de systèmes physiques. Elle utilise une symbolique graphique particulière qui permet de présenter les résultats dans une structure arborescente. Elle est connue en anglais comme Fault Tree Analysis. Même si le terme *Fault* ne correspond pas à *Défaillance*, au vu de la relation entre faute et défaillance expliquée dans la section 2.2 du premier chapitre, nous adoptons dans notre travail le terme classique **Arbre des Défaillances**.

Cette méthode a pour objectif, à partir d'un Evénement Indésirable défini a priori (appelé aussi Evénement-sommet, car il est placé au plus haut niveau de l'AdD, racine de l'arbre), de déterminer les combinaisons d'évènements, c'est-à-dire les défaillances de composants, pouvant finalement conduire à cet Evénement Indésirable (EI). Cette analyse permet de remonter de causes en causes jusqu'aux évènements de base (les feuilles de l'arbre) susceptibles d'être à l'origine de l'événement indésirable. Les évènements de base sont indépendants, et ne seront pas décomposés en éléments plus simples faute de renseignements, d'intérêt ou bien parce que cela est impossible.

Les liens entre les différents évènements identifiés sont réalisés grâce à des portes logiques standards telles que ET, OU, SI (porte conditionnelle). Outre les portes, de nombreux symboles sont utilisés dans la méthode. Par exemple, les événements sont représentés par un rectangle, un cercle ou un losange. Les rectangles représentent des événements (événement-sommet ou événements intermédiaires) résultant de la combinaison d'autres événements par l'intermédiaire des portes. Les cercles représentent des événements de base élémentaires ne nécessitant pas de futur développement. Les losanges représentent des événements de base qui ne peuvent pas être considérés comme élémentaires, mais dont les causes ne sont pas et ne seront pas développées.

La figure 2.1 montre le symbole de quelques éléments de l'AdD que l'on vient d'évoquer et que nous utiliserons par la suite. Une liste complète des conventions de présentation de l'AdD couramment utilisées, est donnée en Annexe A. Cette liste est basée sur la norme (CEI 61025, 1990 : Analyse par Arbre de Pannes) et sur le Fault Tree Handbook (US NR Commission, 1981).

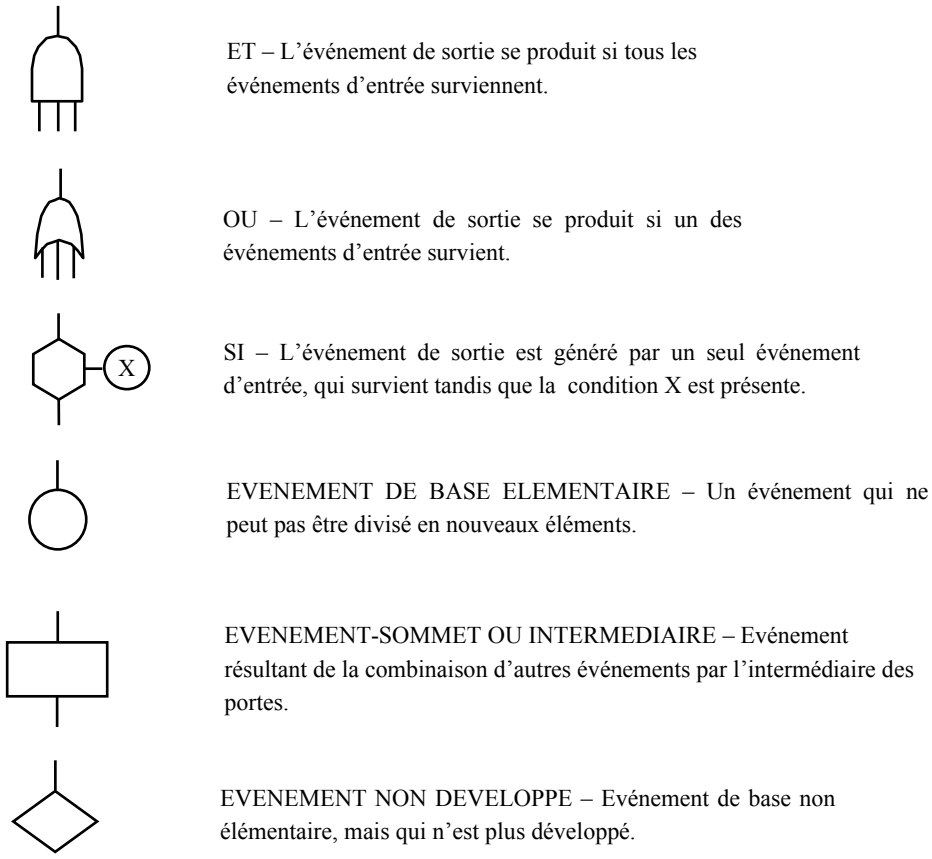


Figure 2.1. Symboles de certains éléments de l'AdD utilisés dans ce chapitre

A titre illustratif, la figure 2.2 montre un exemple d'AdD. C'est un AdD simple (ou une partie d'un AdD plus large).

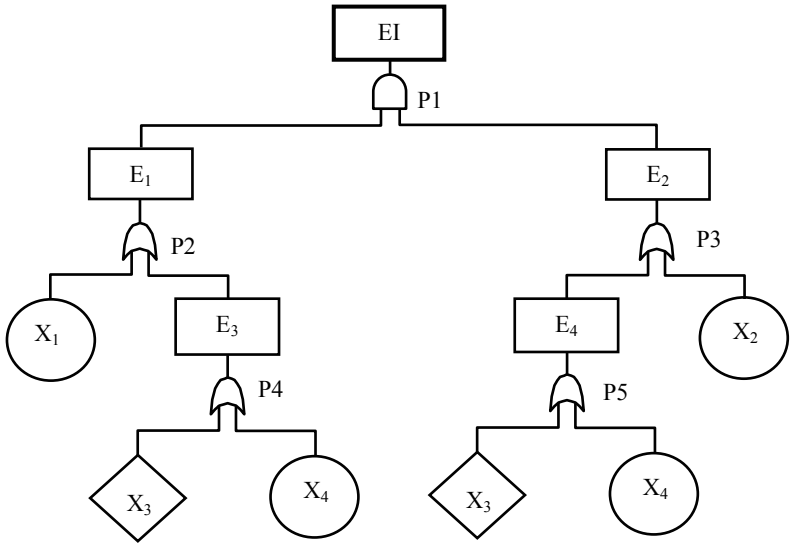


Figure 2.2. Exemple d'un AdD

La description des éléments dans l'AdD de la figure ci-dessus est la suivante :

- l'événement-sommet : EI (le rectangle étant utilisé pour l'événement-sommet et les événements intermédiaires, nous faisons une différence en dessinant le premier avec un trait plus épais) ;
- les événements intermédiaires : E_1, E_2, E_3, E_4 ;
- les événements de base élémentaires : X_1, X_2, X_4 ;
- l'événement non développé : X_3 ;
- P1 : Porte ET ;
- P2, P3, P4, P5 : Portes OU.

Un AdD est sous sa forme restreinte, s'il est décrit uniquement par les portes ET et OU. On dit qu'un AdD contient des variables biformes, lorsqu'il est sous sa forme restreinte et contient deux événements complémentaires. Dans le cas contraire, on dit qu'il contient des variables monoformes. Si un AdD est sous sa forme restreinte et contient seulement de variables monoformes, c'est un **AdD cohérent.** D'autre part, si dans sa forme restreinte, il contient des variables biformes, on dit alors qu'il est un **AdD non cohérent** (Limnios, 2005). La figure 2.3 montre un exemple de chaque catégorie.

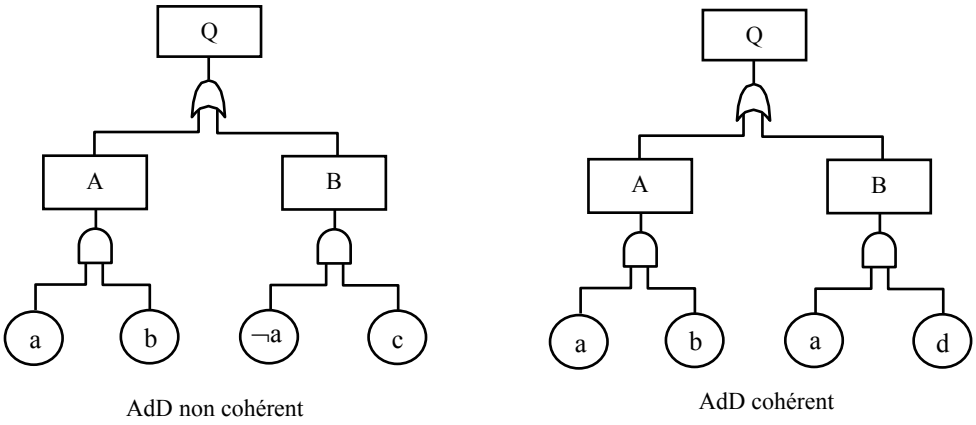


Figure 2.3. AdD cohérent et non cohérent

Les étapes constitutives de toute étude par arbre des défaillances sont les suivantes:

- Synthèse de l'arbre,
- Analyse de l'arbre, elle même divisée en :
  - — Analyse qualitative,
  - — Analyse quantitative.

Elles sont expliquées dans ce qui suit.

1.2 Synthèse de l'Arbre des Défaillances

La définition de l'événement indésirable, qui fera l'objet de l'analyse, est une étape cruciale pour la construction de l'arbre. Plus cet événement est défini de manière précise, plus simple sera l'élaboration de l'arbre des défaillances. L'utilisation préalable de méthodes inductives (AMDEC, HAZOP) permet d'identifier les évènements qui méritent d'être retenus pour une analyse par arbre des défaillances. Une manière systématique de construire l'AdD (synthèse de l'arbre) et qui est généralement suivie par les analystes, est fournie par le Fault Tree Handbook (US NR Commission, 1981). L'ouvrage propose de classifier les événements dans l'AdD comme fautes de composants et fautes de système. Si une faute est due à la défaillance d'un seul composant, alors on parle de faute de composant ; par contre, si la défaillance d'un seul composant n'est pas la seule à produire la faute, alors on parle d'une faute de système.

1.2.1 Faute de système

Si la faute est classifiée comme étant une faute de système, elle se comporte comme un événement intermédiaire, obtenu par combinaison logique de fautes des composants. Une faute de système peut être définie à partir d'une porte OU, ou d'une porte ET. Une porte OU décrit la défaillance d'un système série (figure 2.4.a). Une porte ET décrit la défaillance d'un système parallèle à redondance active (figure 2.4.b).

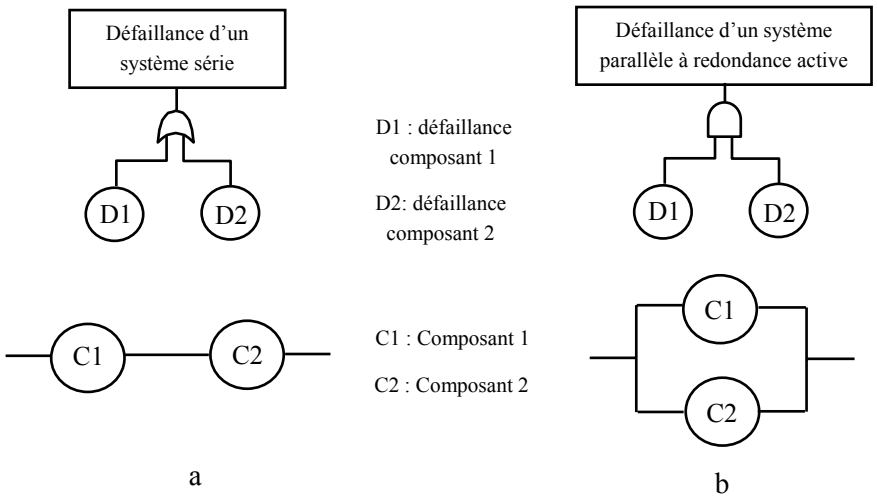


Figure 2.4. Représentation de la défaillance de systèmes série et parallèle à redondance active

1.2.2 Faute de composant

Les fautes de chaque composant sont développées en utilisant la structure arborescente donnée aussi par le Fault Tree Handbook et qui est illustrée dans la figure 2.5. Dans cette structure, les fautes de composant sont classifiées ***par rapport aux causes qui les produisent***.

Une ***faute primaire*** représente la défaillance d'un composant due à ses défauts internes. Cela se produit dans un environnement pour lequel le composant est qualifié, c'est à dire que les

paramètres de l'environnement (température, humidité, pression, ..) sont à l'intérieur des limites des spécifications opérationnelles du composant. Les fautes primaires représentent des événements de base élémentaires de l'arbre.

Une ***faute secondaire*** est une faute d'un composant provoquée par un environnement excessif. En d'autres termes, une faute secondaire représente une situation dans laquelle le composant est défaillant dans des conditions qui excèdent celles pour lesquelles il a été conçu. Ces fautes secondaires sont généralement des événements de base non élémentaires, non développés.

Une ***faute de commande*** décrit une situation dans laquelle le composant n'a pas de défaillance physique et se trouve dans un environnement qualifié mais son opération correcte se produit à un temps erroné ou de façon erronée, à cause des informations erronées fournies par des composants en amont. Les fautes de commande représentent des événements intermédiaires dans l'arbre qui doivent être toujours développés afin d'établir comment d'autres composants en amont dans le système peuvent générer des informations incorrectes pour le composant considéré.

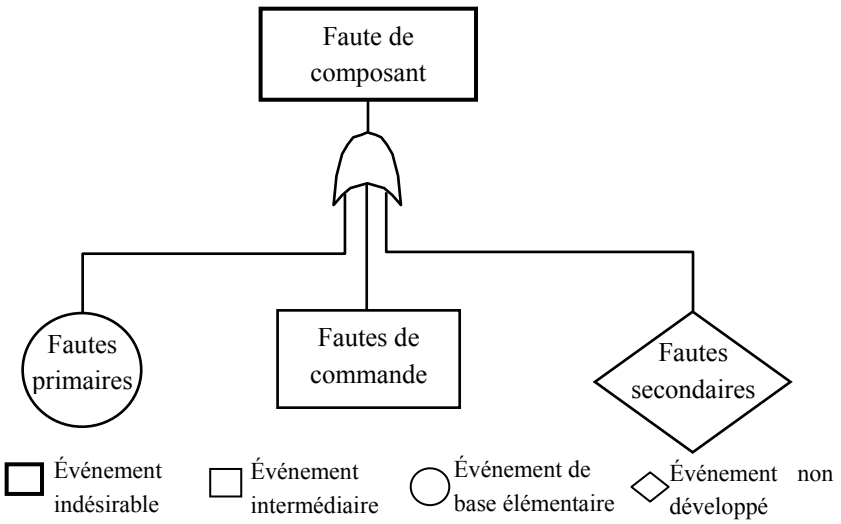


Figure 2.5. Structure de l'AdD classique pour la faute de composant

Une autre classification, basée ***sur la façon dont les fautes de composant se manifestent***, est fournie par Papadopoulos (Papadopoulos et al. 2001). Dans cette approche, le composant est vu comme étant une boîte noire recevant des entrées et générant des sorties. Les fautes sont alors des déviations des sorties du composant, du type :

- Absence ou présence intempestive ;
- Valeur incorrecte (trop faible / trop élevée) ;
- Délai incorrect (trop tôt / trop tard).

Les causes des déviations sont les mêmes que celles indiquées en figure 2.5, c'est-à-dire :

- une déviation de l'une des entrées (faute de commande, le composant transmet la déviation) ;
- une défaillance du composant (faute primaire) ;
- un problème d'environnement (faute secondaire).

Voyons maintenant un exemple pour illustrer cette classification selon le mode de manifestation des fautes. Il s'agit d'analyser une simple vanne TOR (figure 2.6), normalement fermée, et qui possède en entrée le débit *a* et en sortie le débit *b*. Les déviations de la sortie débit *b* sont listées dans la colonne de gauche. Les causes de la déviation sont montrées dans la colonne centrale.

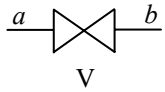
Déviations de la sortie débit *b*	Causes de la déviation	
Faute de composant 1 : Absence intempestive de débit en *b*	Absence de débit en *a* (secondaire), ou vanne bloquée fermée (primaire), ou absence intempestive signal commande (commande).	 V : vanne TOR normalement fermée
Faute de composant 2 : Présence intempestive de débit en *b*	Vanne bloquée ouverte (primaire), ou présence intempestive signal commande (commande).	
Faute de composant 3 : Valeur du débit en *b* trop faible	Valeur de débit en *a* trop faible (secondaire), ou vanne bloquée en position intermédiaire (primaire).	
Faute de composant 4 : Valeur du débit en *b* trop forte	Valeur du débit en *a* trop forte (secondaire).	
Faute de composant 5 : Présence prématurée du débit en *b*	Présence prématurée du débit en *a* (secondaire), ou signal de commande prématuré (commande).	
Faute de composant 6 : Présence trop tardive du débit en *b*	Présence trop tardive du débit en *a* (secondaire), ou signal de commande trop tardif (commande).	

Figure 2.6. Fautes d'une vanne TOR, selon le mode de manifestation des fautes

1.3 Analyse de l'AdD

L'analyse de l'AdD est surtout basée sur le concept de ***coupe minimale***. Une coupe minimale représente la plus petite combinaison (intersection du point de vue logique) d'évènements de base pouvant conduire à l'événement indésirable. On appelle ordre d'une coupe le nombre d'événements qui figurent dans la coupe. Les coupes minimales sont aussi appelées implicants premiers pour les arbres non cohérents, (Rauzy et Dutuit, 1997). L'analyse de l'arbre comprend une analyse qualitative puis éventuellement une analyse quantitative. Ces concepts sont expliqués plus en détail dans ce qui suit.

1.3.1 L'analyse qualitative

Le traitement qualitatif de l'arbre est double. Tout d'abord, il vise à déterminer les coupes minimales puis à examiner dans quelle proportion une défaillance correspondant à un événement de base peut se propager dans l'enchaînement des causes jusqu'à l'évènement indésirable. La recherche des coupes minimales se fait traditionnellement à partir de l'AdD en appliquant les règles classiques de simplification des expressions booléennes à la fonction logique représentée par l'arbre.

L'expression des coupes minimales pour l'EI peut être écrite dans sa forme générale :

$$EI = C_1 \vee C_2 \vee \ldots . \vee C_i$$

Où EI est l'événement indésirable et C_1, C_2,..,C_i sont les coupes minimales. Chaque coupe minimale est une fonction des événements élémentaires de la forme :

$$C_i = \prod_j X_j$$

où X_j, avec $_j \in [1, n]$, est un événement élémentaire.

Un arbre des défaillances est constitué d'un nombre fini de coupes minimales qui sont uniques pour son EI. Une coupe minimale composée d'un seul élément représente une défaillance de composant qui toute seule produit l'EI. Pour une coupe minimale de n-composants, les n-composants doivent tous être défaillants pour l'occurrence de l'EI.

Pour déterminer les coupes minimales, l'arbre est d'abord traduit en une équation booléenne équivalente. Les lois de l'algèbre de Boole s'appliquent ensuite pour éliminer les termes redondants. Nous allons expliquer ceci à l'aide de l'arbre de la figure 2.2, repris en figure 2.7.

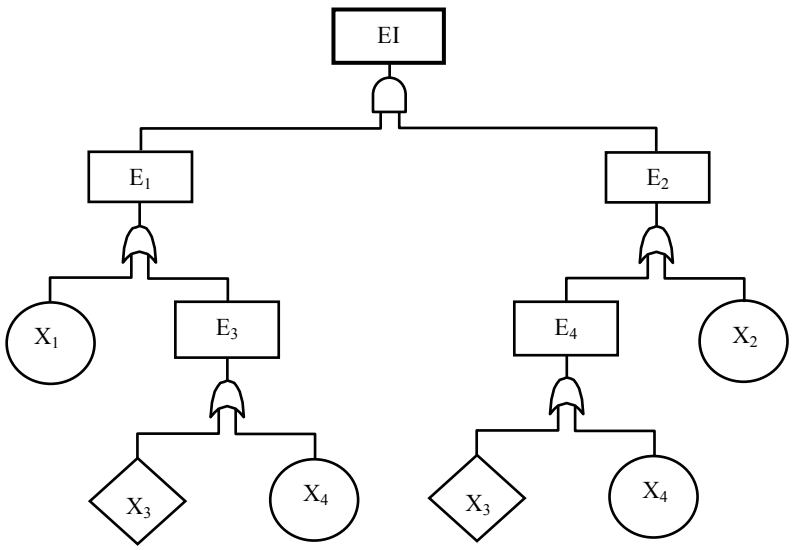


Figure 2.7. Exemple élémentaire pour le calcul des coupes minimales

Il vient :

$EI = E_1 \wedge E_2$

$E_1 = X_1 \vee E_3$

$E_3 = X_3 \vee X_4$

$E_2 = E_4 \vee X_2$

$E_4 = X_3 \vee X_4$

$EI = (X_1 \vee E_3) \wedge (X_2 \vee E_4) = (X_1 \vee X_3 \vee X_4) \wedge (X_2 \vee X_3 \vee X_4)$

$EI = X_1X_2 \vee X_1X_3 \vee X_1X_4 \vee X_3X_2 \vee X_3X_3 \vee X_3X_4 \vee X_4X_2 \vee X_4X_3 \vee X_4X_4$

$EI = X_1X_2 \vee (X_1X_3 \vee X_3X_2 \vee X_3 \vee X_3X_4) \vee (X_1X_4 \vee X_4X_2 \vee X_4X_3 \vee X_4)$

La relation d'absorption nous conduit à :

$$EI = X_1X_2 \vee X_3 \vee X_4$$

L'arbre de la figure 2.7, comporte donc trois coupes, $\{X_1, X_2\}$, $\{X_3\}$ et $\{X_4\}$, (une coupe d'ordre 2 et deux d'ordre 1). L'arbre réduit est montré en figure 2.8.

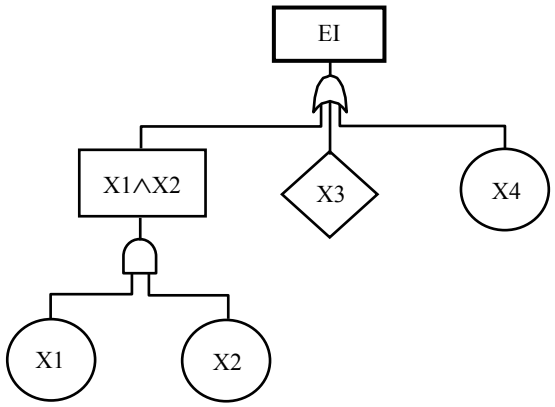


Figure 2.8. Arbre réduit de l'exemple élémentaire de la figure 2.7

Une procédure qui vise à l'obtention des coupes minimales et qui est basée sur les Diagrammes de Décision Binaire est proposée par Rauzy (Rauzy, 1993). L'ouvrage de Limnios (Limnios, 2005), donne une explication plus détaillée d'autres algorithmes développés pour l'obtention des coupes minimales.

Quant au deuxième aspect de l'analyse qualitative de l'AdD, tous les évènements de base sont supposés équiprobables et on étudie le cheminement à travers les portes logiques d'un événement élémentaire ou d'une combinaison d'évènements élémentaires jusqu'à l'événement indésirable. De manière intuitive, une défaillance se propageant à travers l'arbre ne rencontrant que des portes OU est susceptible de conduire très rapidement à l'EI. A l'inverse, un cheminement s'opérant exclusivement à travers des portes ET indique que l'occurrence de l'évènement indésirable à partir de l'événement ou de la combinaison d'évènements de base considérée est moins probable et démontre ainsi une meilleure prévention de l'événement indésirable suite à cette défaillance.

La définition des coupes minimales permet d'accéder directement aux évènements et combinaisons d'évènements les plus critiques pour le système considéré. Ainsi, plus l'ordre d'une coupe minimale est petit, plus forte est a priori la probabilité que les éléments de cette coupe provoquent l'événement indésirable. Un moyen de prévenir les évènements indésirables est de modifier l'arbre des défaillances en vue d'obtenir des coupes minimales d'ordre le plus élevé possible, par l'introduction de portes AND par exemple, ce qui revient a introduire des redondances dans le système étudié.

1.3.2 L'analyse quantitative

L'analyse quantitative de l'arbre des défaillances vise à évaluer, à partir des probabilités d'occurrence des évènements de base, la probabilité d'occurrence de l'événement indésirable. En pratique, il est souvent difficile d'obtenir des valeurs précises de probabilités des évènements de base. En vue de les estimer, il est possible de faire appel à :

- des bases de données,
- des jugements d'experts,
- des essais lorsque cela est possible,
- au retour d'expérience sur l'installation ou sur des installations analogues.

En général, on peut distinguer parmi les méthodes pour calculer la probabilité d'occurrence de l'événement indésirable, celles qui nécessitent la détermination des coupes minimales de l'AdD, et celles qui ne la nécessitent pas. La méthode la plus élémentaire de cette dernière catégorie est connue comme méthode directe. Pour l'appliquer, il faut commencer par le calcul des probabilités des portes de plus bas niveau dans l'arbre et puis remonter l'AdD en calculant, au fur et à mesure, les probabilités des événements intermédiaires, qui sont le résultat de portes. On continue jusqu'à ce que l'on arrive à l'événement sommet. Un recensement des nombreuses méthodes existantes pour calculer la probabilité d'occurrence de l'événement indésirable et les formules pour calculer la probabilité des portes ET, OU, OU-exclusif, k-sur-n, ET prioritaire et SI, peuvent être aussi consultés dans l'ouvrage de Limnios, cité ci-dessus.

En ce qui concerne les méthodes qui nécessitent la détermination des coupes minimales, la plus utilisée est celle basée sur le développement de Sylvester-Poincaré. Elle est implantée dans la majorité des logiciels de traitement des AdD actuellement commercialisés (Dutuit et al. 2002). La formule est la suivante :

$$p(EI) = \sum_{i=1}^{n} p(C_i) - \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p(C_i \bullet C_j) + \sum_{i=1}^{n-2} \sum_{j=i+1}^{n-1} \sum_{k=j+1}^{n} p(C_i \bullet C_j \bullet C_k)$$

avec

C_i coupes minimales ; *p(EI)* probabilité de l'EI, *p(Ci)* probabilités des coupes minimales (la probabilité associée à chaque coupe minimale est obtenue en multipliant les probabilités de défaillance de ses composants).

L'AdD de la figure 2.8 nous permet d'illustrer l'application de cette formule. On note C_1= {X3}, C_2 = {X4}, C_3 = {X1, X2}, $p(X_i = 1) = q_i$ la probabilité des événements de base (supposés indépendants). On a alors :

$$\begin{aligned} P(EI) &= p\{C_1\}+p\{C_2\}+p\{C_3\} - (p\{C_1C_2\}+p\{C_1C_3\}+p\{C_2C_3\}) + p\{C_1C_2C_3\} \\ &= p\{X_3=1\}+p\{X_4=1\} + p\{X_1=1, X_2=1\} - (p\{X_3=1, X_4=1\} + p\{X_3=1, X_1=1, X_2=1\} \\ &\quad + p\{X_4=1, X_1=1, X_2=1\}) + p\{X_3=1, X_4=1, X_1=1, X_2=1\} \\ &= q_3 + q_4 + q_1q_2 - q_3q_4 - q_3q_1q_2 - q_4q_1q_2 + q_3q_4q_1q_2 \end{aligned}$$

Il existe des logiciel industriels d'assistance qui permettent l'édition d'arbres, la recherche de coupes minimales, les calculs de probabilité, par exemple : CABTREE (Cabarbaye, 2001), ARALIA (Arboost Technologies, 2006), FaultTree+ (Isograph Ltd, 1986), Relex Fault Tree Analysis (Relex Software Corporation, 1986).

1.4 La synthèse automatique de l'AdD

La synthèse de l'AdD est une tâche très importante car elle conditionne son analyse qualitative et quantitative. La synthèse automatique de l'AdD est une question déjà abordée par plusieurs travaux scientifiques. La base commune de ces approches est une description préalable du système pour tracer les dépendances et la propagation de défaillances entre les composants. Pour en citer certains, les travaux de Point et Rauzy (Point et Rauzy, 1999) proposent une méthode de génération automatique de l'AdD à partir d'une description du système écrite en langage AltaRica et à l'aide de composants génériques stockés dans une bibliothèque. L'outil KB3-IFAST développé par EDF (Gaudré et al. 2001), permet la synthèse automatique des arbres de défaillances ou des modèles de comportement dynamiques (markoviens ou non) à partir d'une description graphique fonctionnelle du système.

Une autre approche, rapportée par Papadopoulos (Papadopoulos et Maruhn, 2001 ; Papadopoulos et al. 2001), propose un algorithme pour la synthèse de l'AdD. Cet algorithme est intégré dans une méthode appelée HiP-HOPS (pour Hierchically Performed Hazard Origin and Propagation Studies). La figure 2.9 illustre les étapes de HiP-HOPS. Dans cette méthode, le système à étudier est modélisé sous la forme d'un schéma à blocs fonctionnels Simulink. Le système est décomposé en sous-systèmes qui peuvent être eux-mêmes décomposés en composants plus basiques, ce qui donne comme résultat un modèle hiérarchique.

Les défaillances des composants (ou éléments du schéma) sont alors analysées, et présentées sous forme de tableau (comme celui de la figure 2.6 pour l'exemple de la vanne) où sont montrées les déviations des sorties des composants et leurs causes. L'algorithme de synthèse est appliqué pour générer l'AdD à partir de la structure du modèle hiérarchique et de l'information sur le comportement défaillant des composants.

L'algorithme de synthèse est mécanique et peut être automatisé. Il est interfacé à l'éditeur Fault Tree+ qui permet de visualiser les arbres résultants. L'algorithme est aussi interfacé à un éditeur de AMDE.

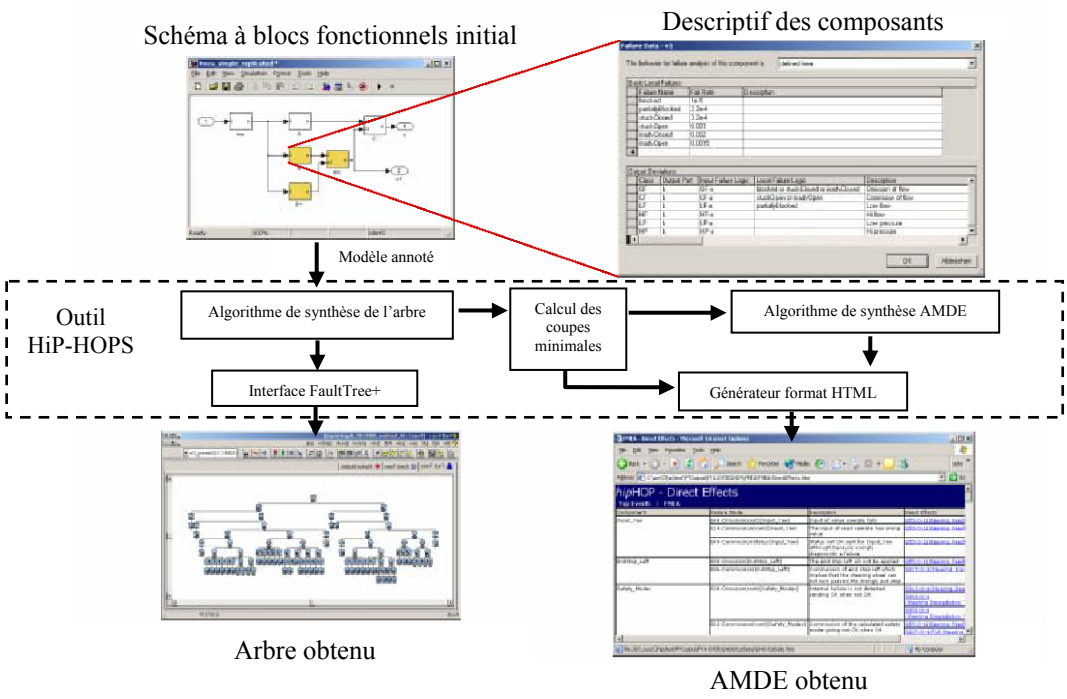


Figure 2.9. La méthode HiP-HOPS pour la construction automatique de l'AdD

2 Etat de l'art des travaux ayant visé à l'extension de la méthode

Le principal avantage de l'analyse par arbre des défaillances est qu'elle permet de considérer des combinaisons d'évènements pouvant conduire *in fine* à un événement indésirable. Cette possibilité permet une bonne adéquation avec l'analyse d'accidents passés qui montre que les accidents majeurs observés résultent le plus souvent de la conjonction de plusieurs évènements qui seuls n'auraient pu entraîner de tels sinistres. Par ailleurs, en visant à l'estimation des probabilités d'occurrence des évènements conduisant à l'événement indésirable, elle permet de disposer de critères pour déterminer les priorités pour la prévention d'accidents potentiels.

Cependant, le caractère statique (c'est à dire ne faisant pas intervenir le temps) de cette représentation limite son utilisation et lui fait souvent préférer d'autres types de modélisation (chaînes de Markov, réseaux de Petri, etc.) pour tenir compte de la dynamique des systèmes. En effet, les arbres des défaillances n'ont pas de dimension temporelle leur permettant de prendre en compte les dépendances fonctionnelles ainsi que l'ordre d'arrivée des événements. Le terme *dynamique* fait appel ici à l'idée d'un ordre d'événements, c'est à dire qu'une défaillance peut se produire lorsque les événements arrivent dans un certain ordre et non pas

dans un autre et que certaines défaillances ne dépendent pas seulement de l'état présent du système mais également de ses états passés.

Il convient de **remarquer** à présent **que le terme «événement» peut avoir deux sens**, selon le contexte :

- état défaillant d'un composant (par exemple, blocage d'une vanne en position ouverte, débit trop important) dans le contexte de l'AdD classique ;
- **changement de la valeur d'une variable** dans le contexte de la théorie des systèmes à événements discrets.

Lorsque nous parlerons de séquence ou d'ordre d'événements (prise en compte du temps logique) ou de temps écoulé entre événements (temps physique), c'est bien entendu **la deuxième définition qui sera utilisée**.

De nombreux travaux de recherche menés par différents laboratoires, en France et à l'étranger, proposent des extensions de la méthode de l'AdD et la rendent plus adaptée au traitement de l'aspect dynamique des systèmes. Dans ce qui suit, nous allons présenter un état de l'art de ces travaux.

Cet état de l'art s'intéresse en premier lieu aux travaux ayant pour objectif d'inclure dans l'AdD la prise en compte de l'ordre des événements (prise en compte du temps logique) ainsi que des valeurs des intervalles de temps entre les événements (prise en compte du temps physique). Ayant constaté, lors de cette analyse bibliographique, qu'un certain nombre d'auteurs proposaient un couplage entre l'AdD et des techniques de vérification formelle pour satisfaire ce besoin de modélisation de la dynamique des systèmes, nous avons intégré dans notre analyse ces recherches, ***même si leur objectif est très différent de celui que nous poursuivons***, comme il sera souligné par la suite.

Globalement, toutes ces méthodes peuvent se ranger dans trois catégories :

- proposition de nouvelles portes modélisant un comportement dynamique ;
- méthodes visant à compléter une analyse par AdD statique en déterminant l'ordre des événements conduisant à l'EI ;
- méthodes de vérification de l'exactitude et de la complétude d'un AdD préalablement construit.

C'est cette classification qui est utilisée dans la suite de cette section.

2.1 Méthodes proposant la création de nouvelles portes

Nous présentons dans cette partie plusieurs propositions visant à intégrer le temps logique et le temps physique dans l'AdD au moyen de portes particulières.

2.1.1 Les portes Priority AND (PAND) et Priority OR (POR)

Le Fault Tree Handbook et le Fault Tree Handbook with Aerospace Applications (US NR Commission, 1981 ; Vesely et al. 2002), définissent deux portes qui permettent de prendre en compte l'ordre d'apparition des événements : la porte PAND et la porte OU Exclusif avec condition de priorité sur une des entrées (que nous appelons Priority OR, ce qui sera expliqué par la suite). Les deux portes seront présentées ci-après. Nous tenons à souligner que, dans ce qui suit, il sera fait l'hypothèse qu'à l'état initial toutes les entrées des portes sont à l'état FAUX et qu'**il n'y a pas d'occurrences simultanées d'événements**.

Priority AND (PAND)

Le symbole et un chronogramme, décrivant de façon graphique informelle le comportement de la porte Priority AND, sont montrés dans la figure 2.10. Cette figure montre le modèle générique d'une porte à *n* entrées différentes et une sortie. La sortie de la PAND, c'est-à-dire la défaillance, est VRAIE seulement si ses entrées se produisent dans un ordre spécifique, de gauche à droite normalement. Cette porte permet donc de représenter une **défaillance résultant d'une séquence d'événements prédéfinie**.

Cette porte n'a jamais été définie avec rigueur, et quand elle est utilisée dans l'analyse qualitative, elle est surtout traitée comme une porte ET (Walker et Papadopoulos, 2006).

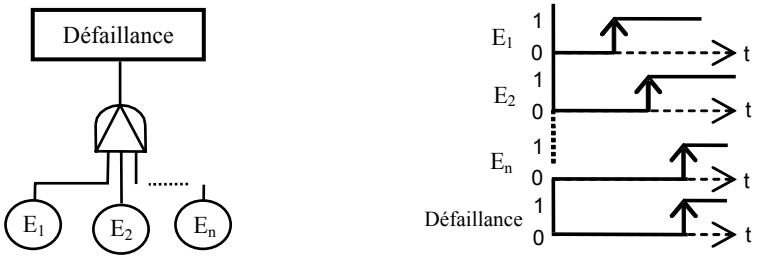


Figure 2.10. La porte PAND : modèle générique

Porte OU Exclusif avec condition de priorité sur une des entrées

Le symbole et un chronogramme, décrivant de façon graphique informelle le comportement de cette porte, sont montrés dans la figure 2.11. Cette figure montre le modèle générique d'une porte à *n* entrées différentes et une sortie, ce qui est une configuration valable (Vesely et al. 2002). La sortie de la porte, c'est-à-dire la défaillance, est VRAIE si et seulement si une seule des entrées, dite **entrée conditionnée**, est VRAIE. Elle permet donc de représenter une **défaillance résultant d'un seul événement** (front montant de l'entrée conditionnée) **se produisant dans une condition prédéfinie** (toutes les entrées non conditionnées à l'état FAUX).

Il existe donc un ordre partiel entre les événements d'entrée de la porte. La défaillance est VRAIE si et seulement si l'événement associé à l'entrée qui porte la condition de priorité se produit avant les autres événements d'entrée, qui peuvent à leur tour se produire après ou non.

Par contre, si c'est une autre entrée qui devient VRAIE en premier, alors il n'y a pas de défaillance. Dans notre travail, nous n'appelons plus OU Exclusif cette porte, mais **Priority OR (POR) avec condition de priorité sur une des entrées**, ou simplement POR, à cause de l'ordre logique qu'elle établit.

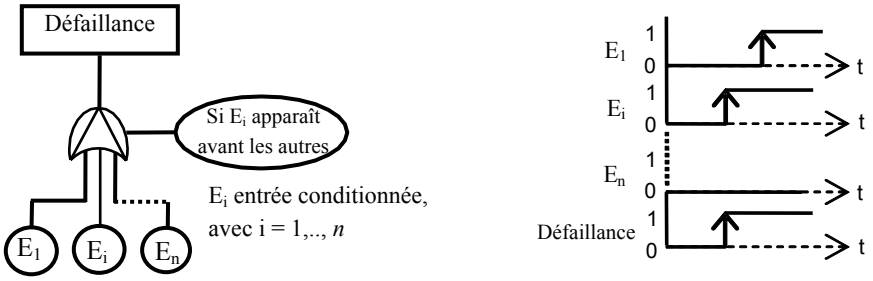


Figure 2.11. La porte POR : modèle générique

En conclusion, les deux portes qui l'on vient de présenter sont une solution intéressante au problème du caractère statique de l'AdD et **nous les adopterons dans notre travail**. Pourtant, toutes les deux manquent de définition formelle. Pour l'instant, nous nous contentons de cette présentation intuitive, mais nous donnerons au chapitre quatre une définition rigoureuse de ces portes.

2.1.2 Portes dynamiques proposées par l'équipe de J.B. Dugan

L'approche développée dans les travaux cités ici (Dugan, Venkataraman et Gulati, 1997 ; Manian et al., 1998 ; Manian et al., 1999 ; Sullivan, Dugan et Coppit, 1999), propose d'ajouter aux portes standard un ensemble spécial de portes dites dynamiques qui prennent en compte l'ordre logique des événements. Ces portes spéciales sont : Functional Dependency (FDEP), SPARE et Sequence Enforcing (SEQ).

La porte **Functional Dependency** est utilisée pour modéliser des situations où le fonctionnement correct d'un composant dépend de celui d'un autre composant. La sortie de cette porte n'est pas une sortie logique (elle est représentée par un trait pointillé), car cette porte traduit la propagation d'une défaillance, comme indiqué dans la figure 2.12.a, où la rupture de la ligne d'alimentation (l'entrée de déclenchement) d'un ordinateur produit la défaillance du CPU et de l'écran, tous les deux événements d'entrée dépendants de l'alimentation. L'événement d'entrée de déclenchement peut être de base, ou bien la sortie d'une porte ET, OU, FDEP, PAND, SPARE, ou SEQ. Par contre, les événements d'entrée dépendants sont des événements de base.

Pour modéliser des systèmes qui contiennent des composants de base réparables, avec des pièces de rechange (pdr) ou des redondances, on peut utiliser la porte **SPARE**. Cette porte peut être du type Cold (les pièces de rechange sont supposées non défaillantes avant leur

utilisation), Hot (les pièces de rechange ont le même taux de défaillance avant et pendant leur utilisation) et Warm Spare (les pièces de rechange ont un taux de défaillance réduit avant leur utilisation). La sortie de chaque porte est vraie quand toutes les unités (primaires et de rechange) ont failli ou ne sont pas disponibles. La figure 2.12.b montre une application de la porte Cold SPARE, où le composant primaire est une unité de mémoire qui a deux unités de rechange. La défaillance se produit si l'unité primaire tombe en panne ainsi que toutes les unités de rechange.

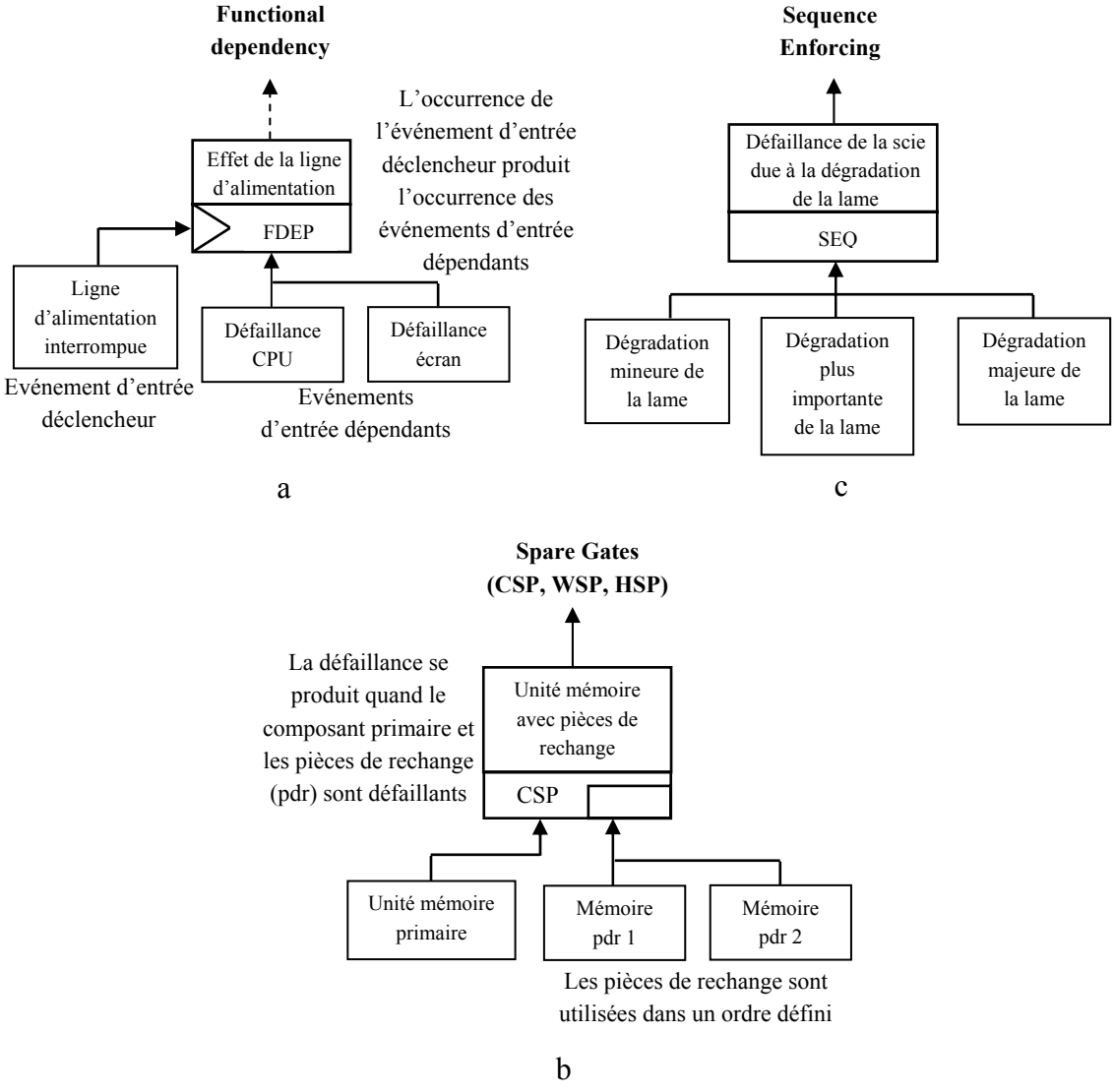


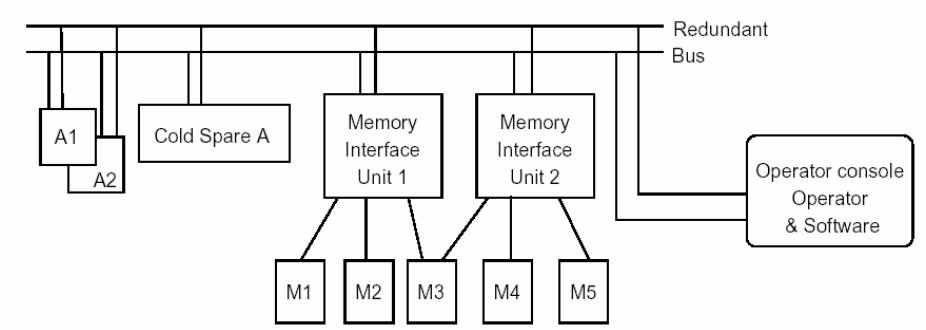
Figure 2.12. Portes spéciales définies par (Dugan et al., 1997)

La porte **SEQ** établit une séquence pré-définie dans l'apparition de ses événements d'entrée. Cette porte force tous ses événements d'entrée à se produire, impérativement, de gauche à droite (ordre séquentiel établi) pour produire la défaillance de sortie. Cette porte peut être comparée avec la porte PAND, définie dans la section précédente. Ces deux portes sont cependant différentes. La porte PAND détecte en effet si les événements se produisent dans un ordre particulier, mais les événements peuvent se produire dans autre ordre. Au contraire,

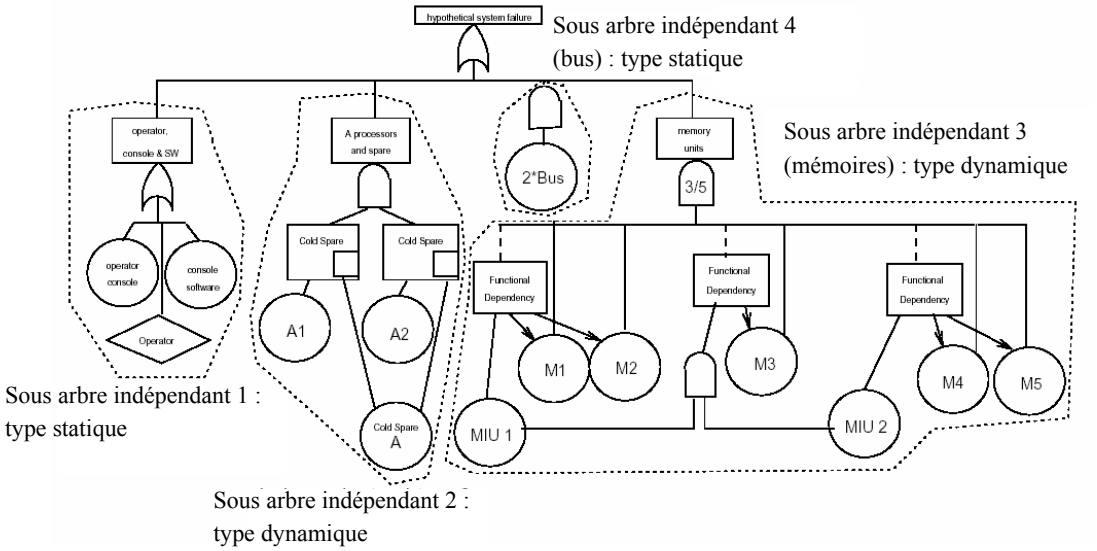
la porte SEQ ne permet que la séquence d'événements qu'elle indique. Le premier événement d'entrée (à gauche de la porte), peut être un événement de base, ou bien le résultat d'une porte ET, OU, ou bien FDEP, PAND, SPARE ou SEQ. Par contre, les autres entrées doivent être des événements de base. Dans l'exemple de la figure 2.12.c, la porte SEQ montre la dégradation graduelle d'une scie où le critère de défaillance de la scie est basé sur la qualité de la lame. Au début, la dégradation de la lame est minimale, donc le système peut fonctionner correctement. Puis, la dégradation augmente mais la lame coupe encore de façon acceptable. Finalement, la dégradation arrive à un point où la lame produit des coupes inacceptables, donc la scie est défaillante.

Cet ensemble de portes est préconisé dans le Fault Tree Handbook with Aerospace Applications (Vesely et al. 2002) et est également implanté dans le logiciel Relex Fault Tree Analysis.

L'exemple de la figure 2.13 (Dugan, Venkataraman et Gulati, 1997), illustre l'utilisation des portes spéciales dans l'analyse par AdD d'un système.



a



b

Figure 2.13. Exemple de système matériel et son AdD dynamique d'aprés (Dugan et al. 1997)

Le système étudié est un ordinateur composé de deux processeurs A1 et A2 (avec un élément de rechange A, considéré comme COLD, c'est à dire, supposé non défaillant avant son utilisation) et cinq mémoires (il suffit de trois pour assurer un bon fonctionnement), qui sont branchées à deux bus de données via deux interfaces mémoire. Si un des interfaces mémoire défaille, les mémoires qui y sont connectées défaillent aussi. Il y a aussi un opérateur qui agit sur le système avec une console et un logiciel d'application (figure 2.13a). Une défaillance du système peut se produire si au moins une des causes suivantes arrive, (AdD illustré dans la figure 2.13.b) :

- l'un des éléments opérateur, console ou logiciel est défaillant ;
- tous les processeurs sont défaillants ;
- trois des cinq mémoires sont défaillantes ;
- les deux bus sont défaillants.

Dans l'approche développée par l'équipe de Dugan, l'arbre des défaillances ainsi construit est décomposé en une partie statique et une partie dynamique qui inclut les portes spéciales. Dans l'exemple, l'arbre de la figure 2.13.b est divisé en 4 sous-arbres dont 2 statiques (sous-arbres 1 et 4) et deux dynamiques portant une ou plusieurs portes spéciales (sous-arbres 2 et 3). Dans la méthode, l'arbre est résolu de façon analytique en traitant la partie statique de manière classique à l'aide de diagrammes de décision binaires, la partie dynamique étant quant à elle traitée à l'aide des chaînes de Markov, par conversion automatique des portes spéciales à une chaîne équivalente. Les détails de cette conversion peuvent être consultés dans le travail de Manian (Manian et al., 1998 et 1999) et aussi dans le Fault Tree Handbook with Aerospace Applications, déjà cité. On peut ajouter qu'une modélisation de chacune des portes, basée sur les réseaux de Petri standard et colorés, est aussi possible (Bobbio et Codetta, 2004). Nous n'allons pas détailler ces modélisations car ceci est en dehors de notre objectif.

On peut conclure cette sous-section en soulignant que, basée sur trois portes spéciales, l'approche qui vient d'être exposée permet d'exprimer la dynamique de systèmes comportant des composants matériels réparables ou redondants. **Nous ne retiendrons pas** cependant **ces propositions de portes** car elles ont été développées spécifiquement pour la modélisation des défaillances aléatoires de composants physiques, ce qui explique leur intérêt pour la synthèse de modèles markoviens, alors que notre travail concerne les défaillances systématiques de composants logiciels : les programmes utilisateurs des contrôleurs logiques.

2.1.3 Portes prenant en compte le temps physique

Le travail développé par Palshikar (Palshikar, 2003), propose aussi la création de nouvelles portes dont la principale caractéristique est leur capacité à exprimer le temps physique. La sémantique de cet ensemble de portes est définie à l'aide d'une Logique Temporelle Linéaire Propositionnelle orientée au Passé (PLTLP pour « past-oriented linear propositional temporal logic »). Cette logique inclut les connecteurs logiques habituels : ¬ (non), ∨(ou), ∧(et), →

(implication), ↔ (si et seulement si) ainsi que plusieurs opérateurs faisant intervenir le temps physique. Ils sont listés dans le tableau 2.1 (avec *n* entier positif) :

Opérateur	Nom	Description
O^-	(PREV) A	A est vraie à l'instant antérieur
O^-_n	(PREV *n*) A	A est vraie à l'instant t_{k-n}, t_k étant l'instant présent, avec $k \geq n$
$\Box^-$	(ALLPAST) A	A est vraie maintenant et dans tous les instants antérieurs dans le passé
$\Box^-_n$	(FORPAST *n*) A	A est vraie maintenant et pendant les *n* instants précédents
$\Diamond^-_n$	(WITHIN *n*) A	A est vraie soit maintenant, soit à un instant quelconque parmi les *n* instants précédents
$\Diamond^-$	(SOMETIME-PAST) A	A est vraie soit maintenant, soit à un instant quelconque dans le passé

Tableau 2.1. Opérateurs faisant intervenir le temps physique

Une porte à une seule entrée est proposée pour chacun de ces opérateurs (figure 2.14).

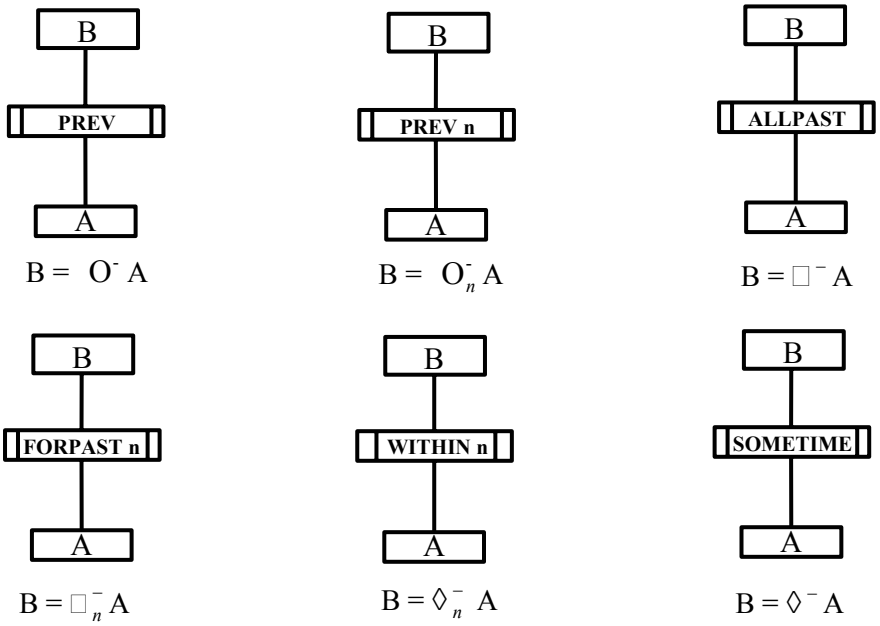


Figure 2.14. Portes définies par (Palshikar, 2003)

Pour mieux comprendre la signification de ces portes, prenons le cas des opérateurs FORPAST *n* et WITHIN *n* (avec $n = 3$). Nous montrons ci-dessous la séquence d'états, dérivée de leur définition, sur l'axe du temps (figure 2.15).

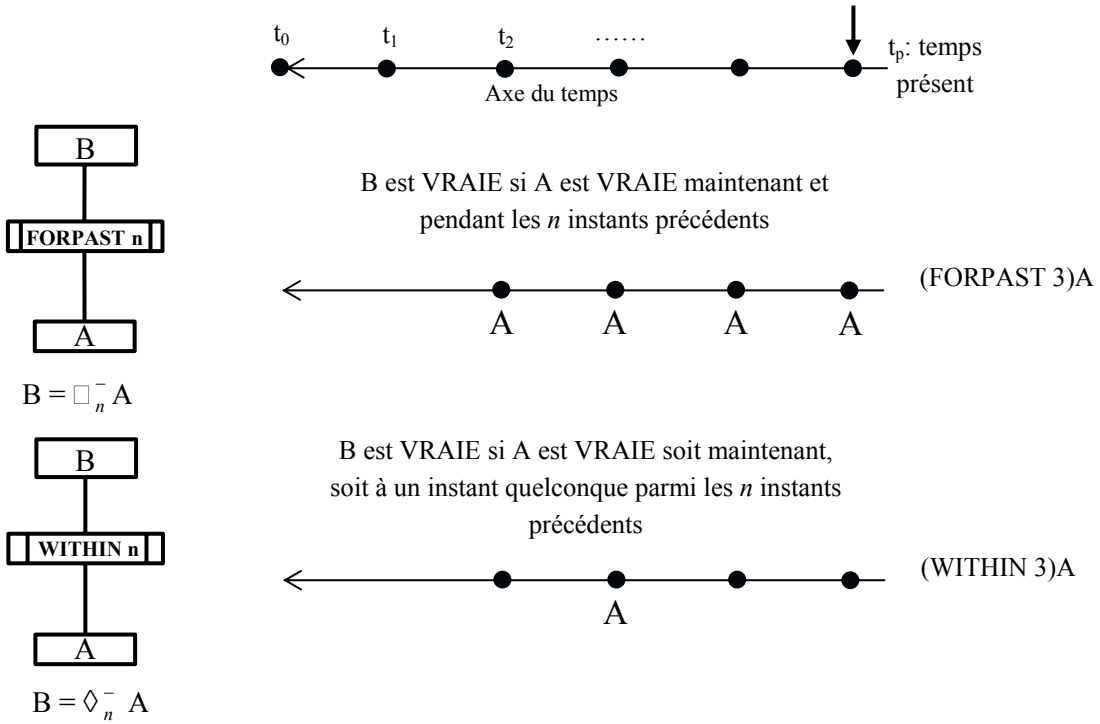


Figure 2.15. Séquences d'états sur l'axe du temps des portes FORPAST n et WITHIN n

Dans la figure ci-dessus, l'axe du temps est orienté vers le passé. Cette orientation inhabituelle de l'axe du temps s'explique car la logique PLTLP suppose de raisonner dans le passé. Une représentation similaire pour les autres portes peut être consultée dans la publication d'origine.

Prise isolément, chacune de ces portes n'exprime pas une défaillance mais elle permet surtout d'exprimer facilement le comportement temporisé (c'est-à-dire, fonction du temps physique) d'une variable logique. Par contre, la combinaison de ces portes avec des portes statiques permet d'exprimer des défaillances. Nous allons illustrer cela à l'aide d'un exemple pris de la publication de Palshikar. Il s'agit d'analyser la défaillance dans la fermeture d'une barrière de passage à niveau. L'AdD correspondant est montré dans la figure 2.16 avec une représentation des événements. L'apparition d'un événement étant représentée par un front montant (flèche vers le haut).

L'analyse débute en considérant la défaillance comme étant VRAIE au présent et la recherche des causes, par conséquence, se fait dans le passé. La défaillance se produit (front montant au temps présent) si le train est arrivé il y a exactement cinq secondes au niveau d'un capteur de présence, et que la commande de fermeture de la barrière n'a pas été générée depuis cet instant. Dans le modèle temporel, la commande n'a pas de front montant ce qui symbolise son absence.

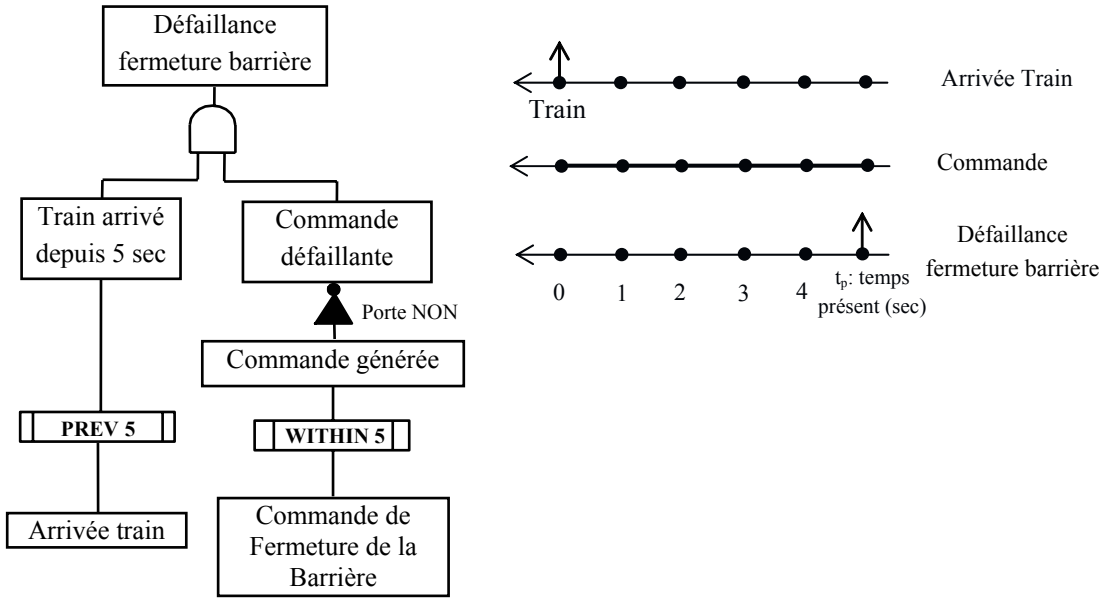


Figure 2.16. AdD et représentation temporelle pour la défaillance dans la fermeture d'une barrière

Cette approche présente aussi des connecteurs appelés « chop connecteurs », qui sont : ${}_m\mathbf{C}_n$ (*m* CHOP *n*), ${}_m\mathbf{SC}_n$ (*m* STRONG-CHOP *n*), ${}_m\mathbf{CF}_n$ (*m* CHOP-FAIL *n*), ${}_m\mathbf{FC}_n$ (*m* FAIL-CHOP *n*) avec *m* et *n* entiers non négatifs. A titre illustratif, nous présentons (figure 2.17), quelques exemples de portes dérivées de ces connecteurs, avec leur définition et la séquence d'états sur l'axe du temps orienté vers le passé (pour tous les cas $m = 2$, $n = 3$).

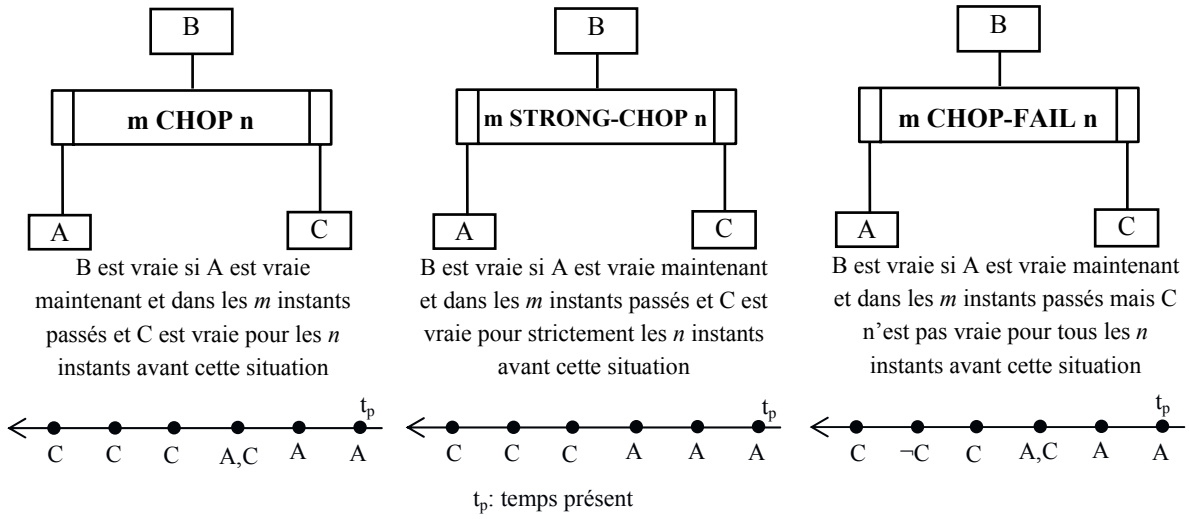


Figure 2.17. Exemples de portes correspondant aux « chop connecteurs »

La figue 2.18 donne un exemple simple d'application de la porte ${}_{30}\mathrm{CF}_{20}$ (avec une représentation des événements sur l'axe du temps). Il s'agit de décrire la défaillance d'un train qui n'arrive pas à freiner au bon moment. Dans ce cas, le train ne s'arrête pas car la condition

Haute vitesse (Hv) est vraie au présent et dans les 30 secondes passés, mais la condition *Frein* (Fr) ne s'est pas produite pour les 20 secondes avant cette situation.

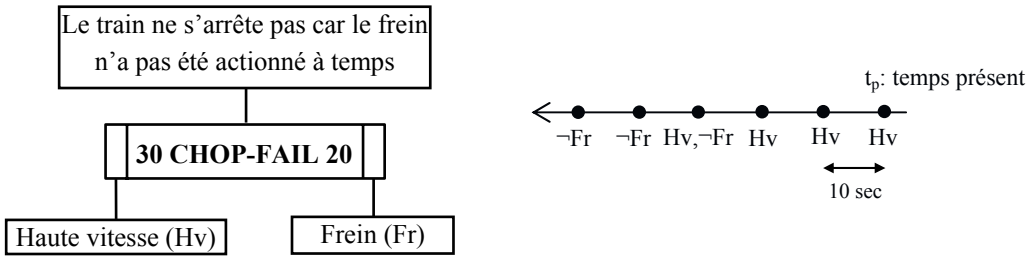


Figure 2.18. Exemple d'utilisation du chop connecteur $_{30}$ CHOP-FAIL $_{20}$

On voit bien que le mode d'analyse des défaillances, qui consiste à considérer l'EI comme vrai au temps présent et à chercher ses causes dans le passé, implique l'usage de cette logique temporelle orientée au passé. Cette logique temporelle contraste avec la logique CTL (section 4.2.2 du chapitre 1), dans le sens où CTL ne permet pas la représentation du temps physique et que les propositions sont exprimées dans le présent et le futur.

Cet ensemble de portes présente l'avantage d'intégrer le temps physique dans l'analyse des défaillances. **Nous considérons que certaines de ces propositions, et en particulier les portes FORPAST n et WITHIN n, peuvent être utiles pour notre travail, mais qu'elles devront être modifiées de manière à exprimer une logique du futur, plus adaptée à notre objectif**.

2.2 Méthodes visant à compléter une analyse par AdD statique en déterminant l'ordre des événements conduisant à l'EI

La création de nouvelles portes pour mieux représenter le caractère dynamique de l'AdD est un des apports scientifiques qui visent à faire évoluer et améliorer cette technique de prévision de fautes. Augmenter la notation de l'AdD n'est pas le seul moyen pour faire apparaître l'ordre logique des événements dans l'analyse. Certains travaux scientifiques proposent des méthodologies pour compléter une analyse statique en déterminant a posteriori l'ordre des événements conduisant à un EI sommet d'un AdD statique. Dans cette sous-section nous allons présenter succinctement deux travaux relevants de cette catégorie.

2.2.1 Première approche

La méthode développée par Kheren et Seguin (Kheren et Seguin, 2003) comporte trois étapes :

1. la construction d'un modèle comportemental formel du système physique étudié. Cette étape est très importante pour l'approche. Le comportement du système physique est

modélisé formellement à l'aide du langage Altarica (Point et Rauzy, 1999), qui est basé sur le concept d'automates à contraintes.

2. la construction d'un AdD statique.
3. l'utilisation de techniques de preuves pour déterminer les séquences d'événements qui conduisent à l'EI sommet de cet AdD.

Cette approche est illustrée à l'aide d'un exemple proposé par les auteurs. Il s'agit d'un système hydraulique qui fournit de la puissance hydraulique à quatre consommateurs. Sa configuration est montrée dans la figure 2.19. Le système comprend trois circuits d'alimentation C_1, C_2 et C_3 et deux unités de transfert de puissance (UTP). Le circuit C_1, qui alimente le consommateur 1, peut être redondé par le circuit C_3. Le circuit C_2, qui alimente les consommateurs 2 et 3, est constitué de deux pompes P_b et P_c, en redondance passive (P_c pompe de secours). Le circuit C_3, pour le consommateur 4, possède également deux pompes P_d et P_e en redondance passive (P_e pompe de secours), et peut être redondé globalement par le circuit C_1 à l'aide de l'UTP. Trois vannes de priorité (VP) permettent d'isoler les consommateurs non prioritaires et d'alimenter uniquement les éléments critiques.

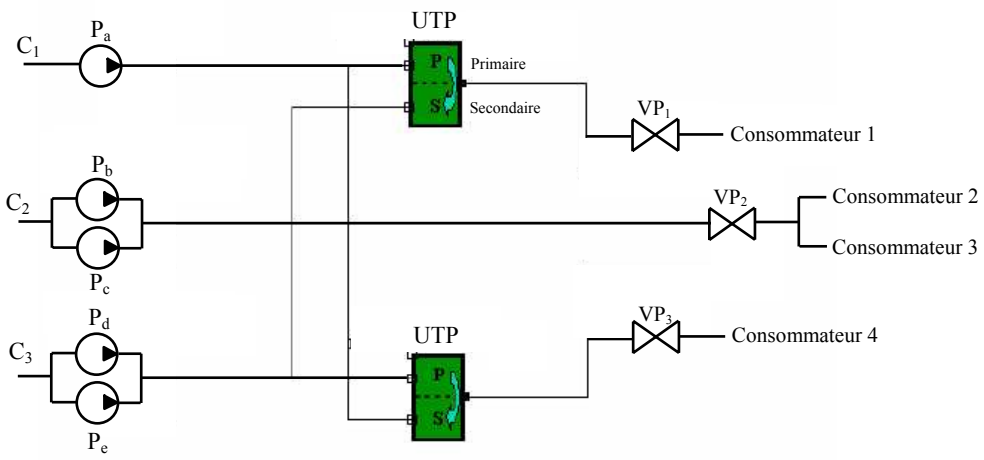


Figure 2.19. Système hydraulique analysé

A l'étape 1 de la méthode préconisée, le modèle formel du système physique étudié (comportement des composants incluant leurs modes des défaillances, relations entre composants) est réalisé en langage Altarica.

L'étape 2 de la méthode consiste à construire manuellement un AdD statique pour un EI donné ; dans ce qui suit on se limite à l'événement indésirable «perte_des_consommateurs_2_et_3». L'arbre déjà réduit (figure 2.20) comporte cinq coupes, une d'ordre 2 (concernant la défaillance des deux pompes P_b et P_c) et quatre défaillances simples (Fuite_Conso_2 ; Fuite_tuyaux ; Fuite_Conso_3 ; VP_2_fermée).

Dans le cas de l'événement intermédiaire «Perte_pompes», si P_b et P_c sont en redondance passive et P_b est la pompe principale, alors la défaillance est vraie si P_b défaille puis P_c défaille également. Cependant, l'AdD ainsi élaboré, qui est statique, ne reflète pas cet ordre logique.

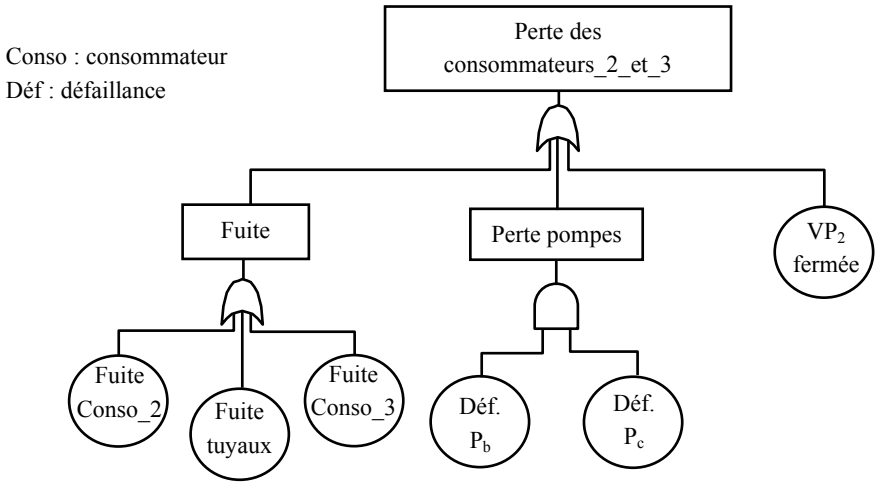


Figure 2.20. AdD pour l'événement indésirable «perte_des_consommateurs_2_et_3»

L'étape trois de la méthode consiste à prouver que le modèle comportemental formel du système physique satisfait (ou ne satisfait pas) des propriétés relatives à l'événement indésirable. Il est par exemple possible de chercher à vérifier que :

«Il n'existe pas de séquence de deux défaillances
qui conduit à la perte des deux consommateurs 2 et 3».

La preuve de cette propriété nécessite d'introduire dans le modèle formel un compteur de défaillances et d'exprimer une propriété formelle fonction de la valeur de ce compteur, ce que nous ne développerons pas.

Cette propriété n'est bien entendu pas prouvée et l'outil de preuves fournit un contre-exemple :

(event = P_b_fail ; event = P_c_fail)}

qui signifie que les défaillances Défaillance_de_Pb et Défaillance_de_Pc doivent se produire dans cet ordre pour que l'EI analysé se produise.

2.2.2 Deuxième approche

Ce même type de méthodologie est aussi proposé par Bozzano et Villafiorita (Bozzano et Villafiorita, 2003). Tout comme l'approche précédente, la méthode est aussi basée sur une modélisation formelle du comportement du système physique étudié, et sur l'application de

techniques de preuves. Le système physique est modélisé formellement en NuSMV (Cimatti, et al. 2000), qui est l'outil choisi par les auteurs.

2.2.3 Conclusion à la sous-section

En conclusion, les travaux présentés dans cette sous-section visent à intégrer le temps logique dans l'analyse prévisionnelle des fautes. Même s'ils combinent AdD statique et techniques de vérification formelle, ils différent notablement de notre approche pour les trois raisons suivantes :

- Leur objectif est l'enrichissement a posteriori d'une analyse basée sur un AdD statique, alors que celui que nous poursuivons est l'extraction de propriétés formelles d'un AdD dynamique. Ces recherches utilisent donc des techniques de preuves pour déterminer les séquences qui conduisent à un EI tandis que nous souhaitons élaborer des propriétés à prouver.

- Ils s'intéressent aux défaillances **aléatoires** de composants matériels, alors que nous nous focalisons sur les défaillances **systématiques** des contrôleurs logiques.

- Ils ne proposent pas de portes permettant au concepteur d'un AdD de représenter un comportement dynamique, ce que nous estimons préjudiciable à la compréhension et à la réutilisation d'une analyse de défaillances.

2.3 Analyse de l'AdD basée sur une formalisation de la sémantique des portes

Un ensemble de recherches scientifiques rapportées dans (Schellhorn et al., 2002 ; Thums et Schellhorn, 2003 ; Schäfer, 2003) proposent une méthodologie pour formaliser les portes de l'AdD et prouver ensuite la justesse de sa construction en vérifiant deux conditions : que l'AdD soit **exact** (les causes dans les coupes minimales produisent effectivement l'événement indésirable) et **complet** (les causes dans les coupes minimales sont les seules à produire l'EI, i.e. il n'y a pas d'autres causes oubliées durant la synthèse «manuelle» de l'arbre). Un AdD global est exact et complet s'il vérifie ces deux conditions.

Ces conditions peuvent être exprimées de façon formelle pour chaque porte. Ainsi, une porte qui associe une cause φ à une conséquence ψ est **exacte** s'il est vrai que $\varphi \rightarrow \psi$ (si la cause se produit, la conséquence doit aussi se produire). Il s'agit dans ce cas d'une implication de la cause à la conséquence. Dans l'autre cas, une porte est **complète** s'il est vrai que $\psi \rightarrow \varphi$ (si la conséquence s'est produite, la cause s'est produite avant). C'est alors une implication de la conséquence à la cause.

L'objectif de cet ensemble de travaux est donc la vérification de l'exactitude et de la complétude d'un AdD préalablement construit sans méthode particulière («à la main»). Pour ce faire, la méthode proposée se déroule en quatre étapes :

1. construction d'un modèle formel, sous la forme d'automates à état finis, du système physique étudié ;
2. synthèse manuelle de l'AdD correspondant à un certain événement indésirable ;
3. formalisation de chacun des événements dans l'arbre ;
4. vérification des conditions d'exactitude et de complétude, pour chaque porte.

2.3.1 Typologie des portes utilisées

La vérification (étape 4 de la liste ci-dessus) est réalisée avec la technique du model-checking qui sert à prouver que chaque porte dans l'arbre est exacte et complète, c'est à dire que les formules qui expriment ces conditions soient vérifiées. Les auteurs introduisent alors une distinction entre les portes, qui peuvent être d'une part, portes de décomposition (*D-portes*), qui sont les portes standard avec une sémantique booléenne, et d'autre part, portes du type cause-conséquence (*C-portes*). Dans les *C-portes*, **le temps physique intervient** entre la cause et la conséquence, c'est à dire que les causes produisent la conséquence mais que celle-ci se produit au bout d'un certain laps de temps[1].

[1] Ces C-portes sont donc des portes dynamiques que nous aurions pu présenter dans la sous-section 2.1. Nous avons préféré les inclure dans la présentation de cette méthode d'analyse pour en faciliter la compréhension.

Pour les *C-portes,* l'exactitude est exprimée, avec la logique temporelle CTL choisie par les auteurs, comme (EF φ) → (EF ψ), i.e. si éventuellement la cause se produit, la conséquence doit se produire aussi. La condition de complétude est A(φ P ψ), i.e. la cause *précède* la conséquence, ou autrement dit, si la conséquence se produit, la cause doit se produire avant ou en même temps. P est un opérateur défini par les auteurs pour faciliter l'expression de la complétude. A(φ P ψ) est équivalent en CTL à ¬E(¬φ U (ψ ∧ ¬φ)), elle peut donc être spécifiée dans le model-checker (Thums et Schellhorn, 2003).

Dans leur travail, les auteurs donnent une expression formelle en CTL de l'exactitude et de la complétude pour les portes ET, OU et SI (chacune de ces portes pouvant être de type *D* ou *C*). En outre, dans (Schellhorn et al., 2002), ces expressions sont définies avec la logique ITL (Interval Temporal Logic).

2.3.2 Exemple d'application

La figure 2.21 montre, à titre d'exemple, une partie d'un système de remplissage d'un réservoir (Thums et Schellhorn, 2003). Le moteur de la pompe assurant le remplissage du réservoir est alimenté par un circuit électrique qui est à son tour contrôlé par un relais K. Par simplicité, le circuit d'alimentation de la bobine de K (CaK) n'est pas détaillé. Le temps supposé de remplissage du réservoir est de 60 secondes.

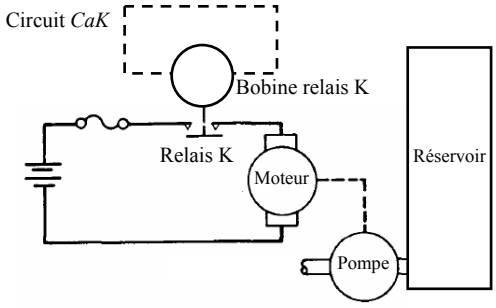


Figure 2.21. Système de remplissage d'un réservoir

Tout le système physique est modélisé à l'aide d'automates à états finis, comme celui modélisant le relais K (figure 2.22) et que nous incluons ici à titre d'illustration. Les automates modélisant les autres composants peuvent être consultés dans la publication d'origine.

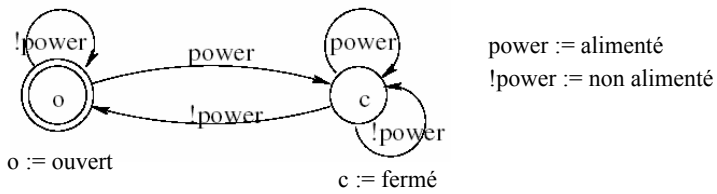


Figure 2.22. Modèle du relais K

Cet automate a deux états et évolue de l'état *ouvert* à l'état *fermé* si K est alimenté. De façon non déterministe, si le relais K n'est plus alimenté (!power), l'automate revient à l'état *ouvert*, ou bien reste dans l'état *fermé* si le relais K ne peut pas s'ouvrir à cause d'une défaillance physique. Ainsi, l'automate prend en compte le mode de défaillance «ne s'ouvre pas» du relais K.

D'autre part, un AdD correspondant à l'EI *Rupture du réservoir*, est construit. La figure 2.23.a, montre un extrait de l'AdD final. Le reste de l'arbre et la description complète du système qui sert à remplir le réservoir, peuvent être aussi consultés dans la publication d'origine. Ce qui nous intéresse ici surtout est de montrer comment sont vérifiées l'exactitude et la complétude de l'arbre.

Dans l'AdD, *Rupture du réservoir* se produit si le réservoir est rempli de façon continue pendant une durée de plus de 60 sec. Cela se produit si le relais K est fermé pendant plus de 60 sec, ce qui peut être dû à la présence de courant dans la bobine du relais K durant plus de 60 sec ou bien à cause d'une défaillance physique de K qui ne s'ouvre pas. Une porte OU relie les deux causes. Dans l'exemple, tous les événements dans l'AdD sont exprimés avec des formules en Clocked CTL (figure 2.23.b), qui permet d'exprimer des intervalles de temps physique et qui est compatible avec le model-checker RAVEN utilisé dans le travail décrit ici. Dans (Schäfer, 2003) et (Cha et al. 2003) sont utilisés les model-checkers Moby/DC et UPPAAL, respectivement.

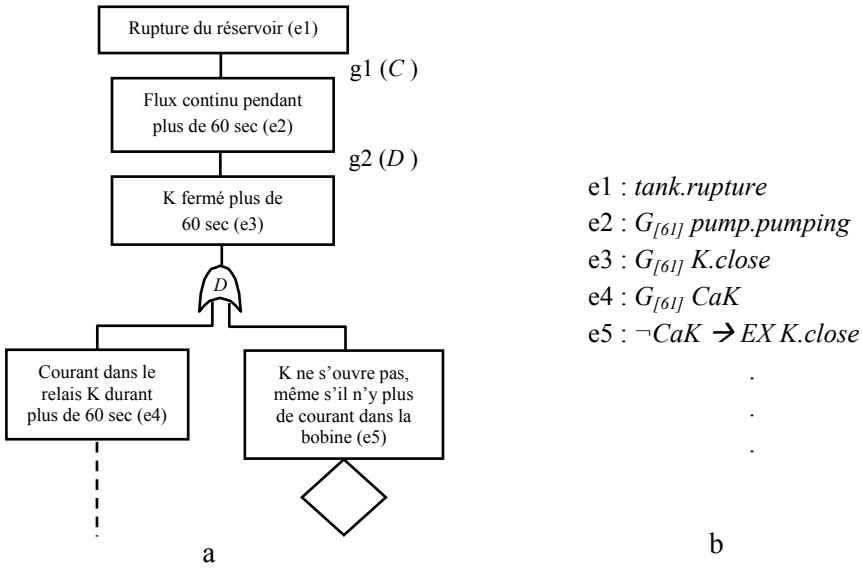


Figure 2.23. Extrait de l'AdD pour l'exemple du réservoir

Ainsi, l'EI est formalisé comme *tank.rupture*. La cause de la rupture est un flux continu pendant plus de 60 sec, qui est formalisée comme $G_{[61]}$*pump.pumping*. Donc, ces deux événements sont reliés implicitement par une porte cause conséquence (*C-porte*), représentée par g1(*C*) dans la figure 2.23.a. Les autres portes de l'arbre sont des *D-portes*.

La vérification est alors appliquée sur cette porte cause-conséquence. La formule générale pour la complétude pour une telle porte est : A(φ P ψ) et en remplaçant les termes on a alors A($ok_{pumping}$ P *tank.rupture*), avec $ok_{pumping}$ état qui confirme le flux continu pendant plus de 60 sec. De façon analogue, l'exactitude est alors vérifiée : EX $ok_{pumping}$ → EX *tank.rupture.* Les deux conditions sont prouvées avec RAVEN. On suit la même procédure pour les autres portes qui sont *D-portes*. Les conditions sont alors prouvées pour chaque porte de l'arbre, ce qui montre que l'AdD de la figure 2.23.a est **exact** et **complet**, i.e. les seules causes qui produisent l'événement indésirable sont les coupes minimales. Si, par exemple, la complétude n'est pas prouvée, le model-checking peut fournir un contre exemple et détecter la cause manquante, ce qui est très avantageux.

Ce même type d'approche est aussi abordé dans les travaux de Xiang (Xiang, 2005), où est proposée une méthode formelle qui assure que l'AdD est exact et complet dès l'étape même de sa construction. Là, l'auteur étend la méthode et obtient aussi à partir de l'AdD formalisé, des propriétés de sûreté exprimées algébriquement. Elles peuvent être vérifiées avec la technique du theorem proving (avec le langage algébrique CafeOBJ) et du model-checking (avec le model-checker Maude).

2.4 Conclusion à l'étude bibliographique

Nous avons présenté ci-avant les résultats des principaux travaux scientifiques visant à rendre la méthode de l'AdD plus adéquate pour la représentation de la dynamique des systèmes réels. Ainsi nous avons présenté en premier lieu, certaines approches qui proposent la création de nouvelles portes pour prendre en compte l'ordre d'apparition d'événements ou le temps physique. Nous considérons que cette solution peut être utile pour notre travail qui s'intéresse aux propriétés de contrôleurs logiques. Toutes les portes proposées ne seront pas retenues ; en particulier, celles introduites pour la modélisation de systèmes physiques avec des redondances ne nous sont d'aucune utilité. Il conviendra d'autre part de doter les portes retenues d'une sémantique formelle rigoureuse et appropriée à nos besoins.

Nous avons également décrit des méthodes qui font une utilisation combinée du model-checking et l'AdD, deux éléments clés dans notre approche, soit pour compléter une analyse par AdD statique, soit pour vérifier l'exactitude et la complétude d'un AdD préalablement construit manuellement. Quel que soit l'intérêt de ces travaux, en particulier pour ce qui concerne la formalisation des portes conventionnelles de l'AdD, nous n'utiliserons pas ces résultats. Notre objectif, faciliter l'obtention de propriétés formelles des contrôleurs logiques à partir d'une analyse par AdD, est en effet différent. De plus, ces deux types de méthodes

reposent sur une modélisation du système physique étudié (pompes, relais, …). Nos travaux ne s'intéressent pas au système physique commandé, mais aux contrôleurs logiques, ce qui renforce notre choix. En résumé, nous retiendrons de cette étude bibliographique :

- que plusieurs portes dynamiques prenant en compte le temps logique ou physique et dotées ou non d'une sémantique formelle, ont été proposées et peuvent être utiles à notre travail ;
- qu'il importe de doter les portes que nous utiliserons d'une sémantique formelle adaptée à notre objectif : la modélisation des fautes d'un contrôleur logique.

3 Notre approche : de l'Arbre des Défaillances aux propriétés formelles

La fin du premier chapitre nous a permis de présenter l'objectif de ce travail de thèse ainsi que d'esquisser l'approche que nous voulons développer. La figure 2.24 présente la méthode que nous adopterons pour atteindre cet objectif et ses étapes de réalisation.

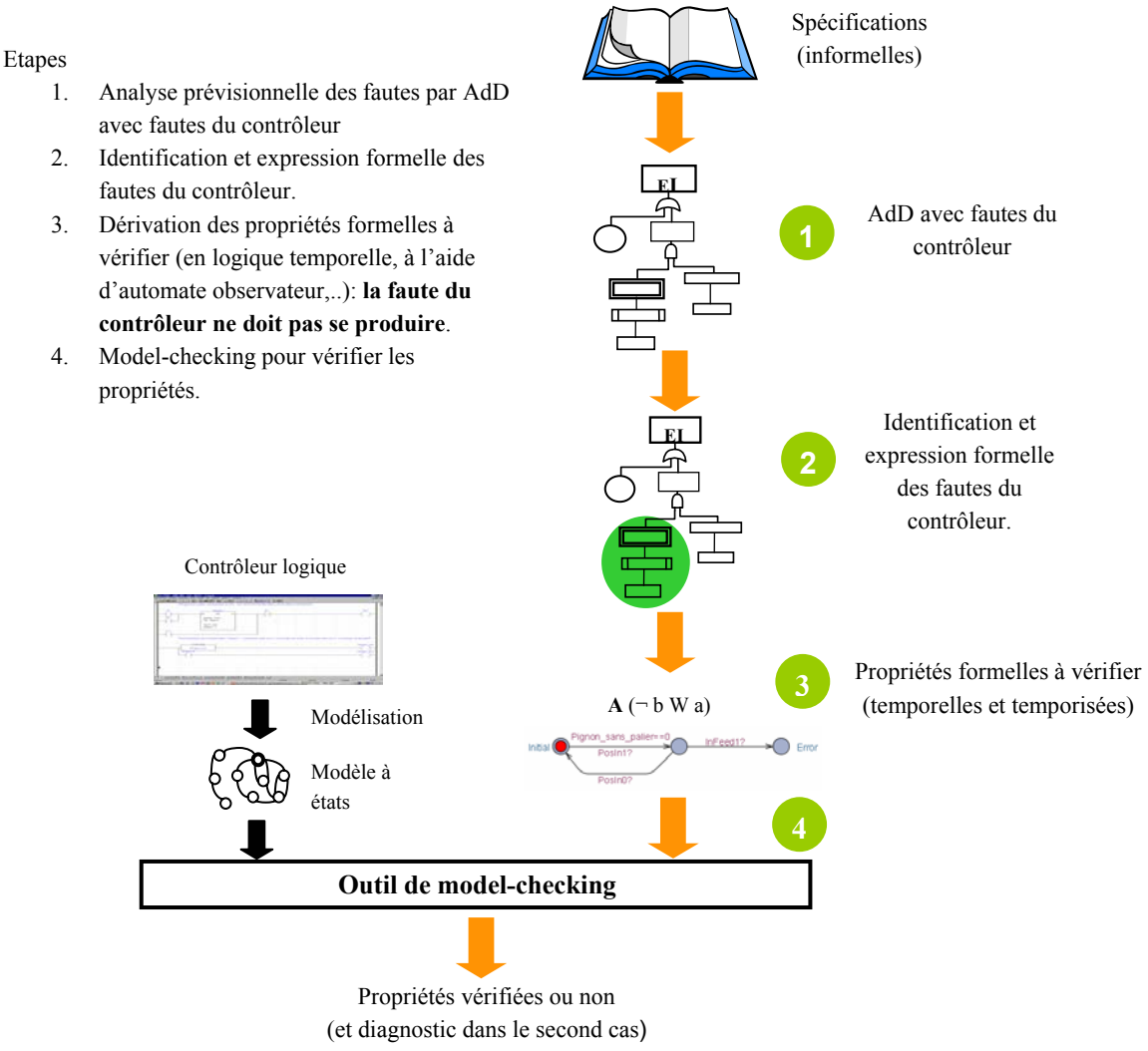


Figure 2.24. Vue générale de notre méthode

Quatre propositions sont alors faites pour mettre au point notre méthode. Elles sont décrites ci-dessous :

1. ***Proposition d'un nouveau modèle générique de la faute de composant***
   A première vue, il semble difficile de vouloir traiter les fautes systématiques avec un outil comme l'AdD qui est normalement utilisé pour analyser de défaillances de composants physiques. Il n'est en effet pas possible d'utiliser cet outil tel qu'il est pour nos fins. Pour inclure l'analyse des fautes systématiques dans la structure de l'AdD, nous proposons donc d'étendre la structure classique du modèle générique de faute de composant (figure 2.5). Ce nouveau modèle respectera les règles traditionnelles de construction mais intégrera le concept de faute systématique comme un nouvel élément à développer.

2. ***Expression des fautes systématiques avec un vocabulaire des portes***
   Pour représenter les fautes systématiques, qui reposent sur l'ordre d'événements ou le temps physique, nous préférons une solution graphique et nous choisissons donc de nous appuyer sur l'utilisation de certaines portes pour décrire l'ordre logique des événements (portes temporelles) et le temps physique (portes temporisées).

3. ***Traduction formelle du comportement des portes temporelles et temporisées***
   Une fois analysées les fautes systématiques avec l'AdD, il nous faut ensuite en déduire facilement les propriétés formelles à vérifier. Pour ce faire, il convient d'exprimer formellement le comportement décrit par chacune des portes temporelles et temporisées, c'est-à-dire de doter chacune des portes du vocabulaire retenu d'une sémantique formelle.

4. ***Obtention du modèle formel équivalent à une association cohérente de portes***
   Il importe enfin d'établir des règles qui permettent de vérifier la cohérence d'une association de plusieurs portes temporelles et/ou temporisées, ainsi qu'une méthode d'obtention du modèle formel équivalent à une association cohérente de portes.

Les propositions 1 et 2 sont de nature méthodologique et sont abordées au chapitre trois. En revanche, les propositions 3 et 4 sont de nature formelle et feront l'objet des chapitres quatre et cinq, respectivement.

Chapitre 3.

Prise en compte des fautes systématiques dans l'AdD

La fin du premier chapitre nous a permis d'introduire deux challenges scientifiques qu'il faut relever pour mettre au point notre approche. Ces challenges scientifiques sont rappelés ci-dessous:

- Les fautes systématiques peuvent être fonction de l'ordre d'apparition des événements ou du temps physique.
- Elles doivent être exprimées formellement, sous forme de formules en logique temporelle ou d'automates observateurs.

Nous avons avancé également au chapitre précédent quatre propositions pour relever ces challenges scientifiques et développer notre méthode :

- Modifier le modèle classique de la faute de composant (figure 2.5 du chapitre 2) et proposer ainsi un nouveau modèle générique qui intègre les fautes systématiques dans l'analyse de la faute de composant. Ce nouveau modèle est la première contribution de notre travail.
- Utiliser un vocabulaire de portes pour exprimer les fautes systématiques dans l'AdD.
- Traduire formellement le comportement modélisé par les portes adoptées.

- Proposer une méthode de contrôle de la cohérence d'une association de portes et d'élaboration du modèle formel équivalent à une association cohérente de portes.

L'objectif de ce troisième chapitre est de développer les deux premières propositions de la liste ci-dessus. Pour ce faire, dans la première section du chapitre, nous analysons les causes des fautes des contrôleurs logiques, pour ensuite montrer en détail le nouveau modèle générique de la faute de composant. Le chapitre se poursuit par la présentation des portes que nous avons adoptées. La dernière section illustre, avec un exemple, l'application du nouveau modèle générique que nous proposons.

1 Les fautes du contrôleur logique

La description des systèmes automatisés a déjà été abordée au premier chapitre, section 1. Ici on rappelle qu'un système automatisé logique bouclé est composé d'un contrôleur logique et d'un processus contrôlé (figure 3.1).

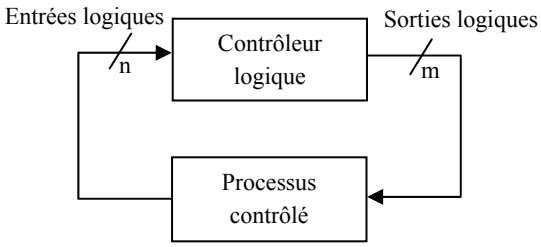


Figure 3.1. Système automatisé bouclé

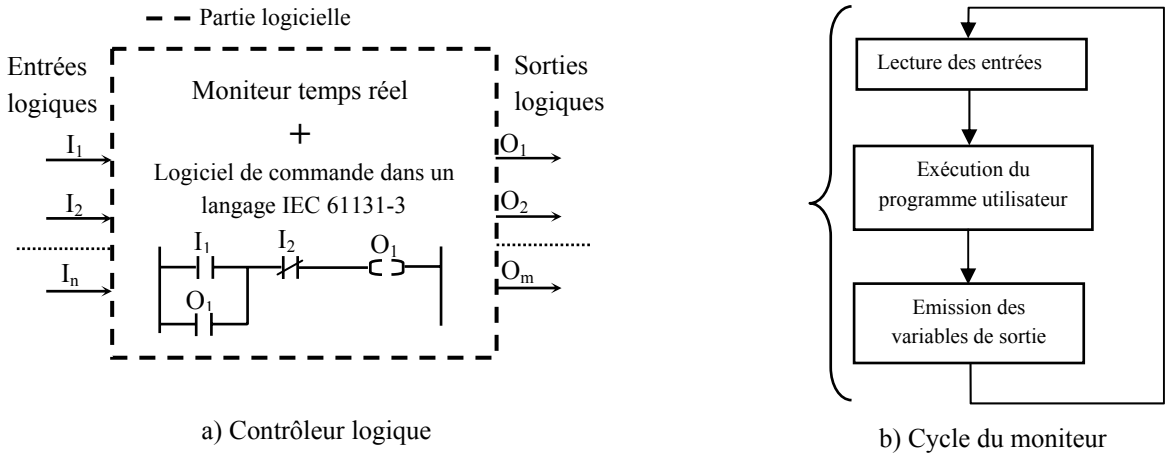
Le contrôleur est destiné à la commande du processus contrôlé (ensemble des composants physiques), appelé aussi partie opérative ou procédé. Le contrôleur envoie des ordres (sorties logiques du contrôleur) vers les pré-actionneurs, qui sont élaborés à partir de données d'entrées provenant des capteurs, de consignes opérateur et d'un logiciel de commande. Ce logiciel de commande implanté dans le contrôleur logique est écrit dans un langage normalisé (IEC 61131-3, 1993), et s'exécute sous le contrôle d'un moniteur temps réel qui réalise un cycle à trois étapes (figure 3.2):

- lecture des variables d'entrée.
- exécution du programme utilisateur.
- émission des variables de sortie.

Les fautes du contrôleur sont observables au niveau de ses variables de sortie. Il s'agit en fait de déviations des sorties. Une faute du contrôleur se manifeste donc par :

- **une sortie à zéro (FAUX) à tort (erroneous omission)**.
- **ou, une sortie à un (VRAIE) à tort (erroneous commission)**.
- **ou, une valeur de sortie fausse (valeur incorrecte)**. C'est une variable de sortie logique dans un état différent de ce qui est attendu. Ce cas regroupe les deux précédents sans préciser la valeur attendue.

- **ou, une sortie émise au mauvais moment (délai incorrect)**. C'est le changement d'état d'une variable de sortie logique se situant trop tard/trop tôt par rapport à ce qui est attendu.



a) Contrôleur logique

b) Cycle du moniteur

Figure 3.2. Structure d'un contrôleur logique

Le tableau ci-dessous montre un comparatif entre la classification des fautes d'un composant physique et celles du contrôleur logique. On voit bien que les mêmes termes peuvent être conservés pour la classification ; bien sûr, la nature de la faute change.

	Composant physique	**Contrôleur logique**
Sortie à zéro à tort	Grandeur de sortie absente alors qu'elle devrait être présente	Sortie logique mise à zéro à tort
Sortie à un à tort	Grandeur de sortie présente alors qu'elle ne le devrait pas	Sortie logique mise à un à tort
Valeur de sortie fausse	Grandeur de sortie à une valeur différente de ce qui est attendu	Sortie logique dans un état (VRAI/FAUX) différent de ce qui est attendu (FAUX/VRAI)
Sortie émise au mauvais moment	Changement de la valeur d'une grandeur de sortie trop tard ou trop tôt par rapport à une référence temporelle	Changement d'état d'une sortie logique se situant trop tard/trop tôt par rapport à ce qui est attendu

Tableau 3.1. Comparaison de la classification des fautes d'un composant physique et du contrôleur logique

On connaît donc la manière dont les fautes du contrôleur logique se manifestent. Abordons maintenant les causes qui les produisent. Les fautes du contrôleur logique peuvent être dues à des (figure 3.3) :

- **Défaillances physiques du contrôleur** (court-circuit sur carte d'entrées/sorties, processeur endommagé, etc).
- **Fautes dues à des signaux d'entrée incorrects** (dues à des défaillances des capteurs, par exemple).

- **Fautes de conception dans le programme utilisateur du contrôleur** (dues à des erreurs de programmation ou à une mauvaise interprétation des spécifications).

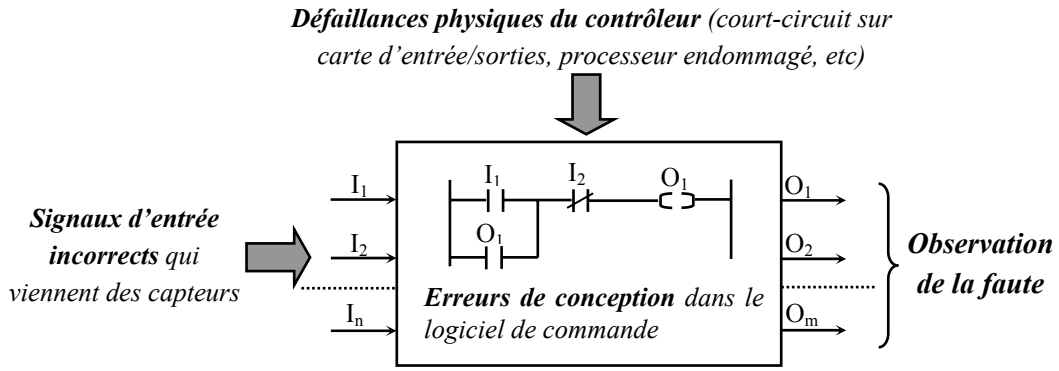


Figure 3.3. Les causes des fautes du contrôleur logique

C'est en fait la troisième catégorie de fautes que nous considérons comme systématiques car, résidant dans le logiciel de commande, elles sont reproductibles à chaque fois que les mêmes conditions apparaissent. Pour identifier cette sorte de fautes, il faut que l'analyste suppose que même si les informations d'entrée sont correctes, le contrôleur produit une faute.

L'exemple suivant permet d'illustrer les causes des fautes du contrôleur logique qu'on vient de lister. La figure 3.4 montre un programme utilisateur simple en Ladder Diagram (IEC 61131-3, 1993) implanté dans un contrôleur logique. Ce programme implante une mémoire prioritaire à 0 et détermine l'état de la sortie O_1 par l'évaluation de la combinaison logique des entrées I_1 et I_2 et de l'état précédent de cette sortie, selon la formule $O_1 = \neg I_2 \wedge (I_1 \vee O_1)$. I_1 étant l'entrée de mise à 1 et I_2 celle de mise à 0).

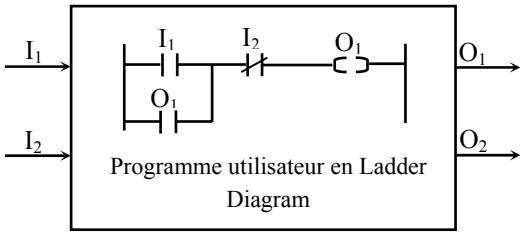


Figure 3.4. Exemple d'un programme utilisateur implanté dans un contrôleur logique

Une faute du contrôleur peut être produite, d'une part, par une défaillance des éléments physiques du contrôleur (court-circuit sur carte d'entrées/sorties, processeur endommagé, etc). D'autre part, la faute peut aussi être due à la réception d'une information erronée. Par exemple, posons qu'à l'état présent du programme, la sortie O_1 est dans l'état VRAI (i.e. $O_1 = 1$) et doit être mis à l'état FAUX (i.e. $O_1 = 0$), mais le détecteur qui élabore la variable I_2 est défaillant et équivalent à un circuit ouvert ce qui conduit à générer une valeur fausse de la variable I_2 ($I_2 = 0$). La faute est alors une **sortie mise à un à tort** (erroneous commission). Ce

cas là décrit une situation où une entrée erronée (EE) produit une sortie erronée (SE) du contrôleur. Cela peut être exprimé de façon symbolique comme (EE → SE).

Une troisième possibilité de produire une faute du contrôleur dans l'exemple de la figure 3.4 (et en fait, dans n'importe quel programme de commande), est une situation où le contrôleur produit une sortie erronée en réponse à un ensemble de signaux d'entrée corrects qui déclenchent une erreur dans la logique de commande. Ceci peut arriver si par exemple, une erreur de programmation a été introduite et au lieu de programmer une mémoire prioritaire à 0, on programme une mémoire prioritaire à 1. Cette situation peut être représentée symboliquement comme (EC → SE), c'est à dire, des entrées correctes (EC) produisent des sorties erronées (SE). C'est cette catégorie de fautes, qu'on appelle systématiques et qu'on propose d'inclure dans la structure classique de l'AdD, qui est abordée dans la section suivante.

2 Nouveau modèle générique de la faute de composant incluant les fautes systématiques

L'intégration dans la structure de l'arbre des défaillances des fautes systématiques du contrôleur aide à l'identification de telles fautes. L'objectif est d'identifier les fautes causées par des erreurs de conception pour pouvoir ensuite les éliminer.

La structure classique de la faute de composant intègre les fautes physiques et de commande, c'est-à-dire les fautes dues à un composant qui se trouve en amont dans la boucle de commande, mais omet toute évaluation et analyse des erreurs dans la logique du contrôleur. Pour pallier cette insuffisance de l'AdD traditionnel et inclure les fautes systématiques dans l'analyse, nous proposons un nouveau modèle générique qui maintient les règles de construction de l'AdD classique résumées dans la section 1.2 du chapitre 2 et dans le schéma de la figure 2.5, mais introduit les fautes systématiques.

Nous considérons que la faute d'un composant peut être décomposée en :

- fautes primaires et secondaires
- fautes de commande.

Pour les fautes de la dernière catégorie dues au contrôleur logique, deux sous-classes doivent être définies :

Fautes de commande classiques : fautes du contrôleur provoquées par une défaillance physique du contrôleur et fautes produites par des signaux d'entrée incorrects.

Fautes de commande systématiques : appelées tout simplement ***fautes systématiques***, qui correspondent à des erreurs dans le programme utilisateur. Ces fautes résident dans le logiciel

de commande et sont reproductibles à chaque fois que les mêmes conditions apparaissent (figure 3.5).

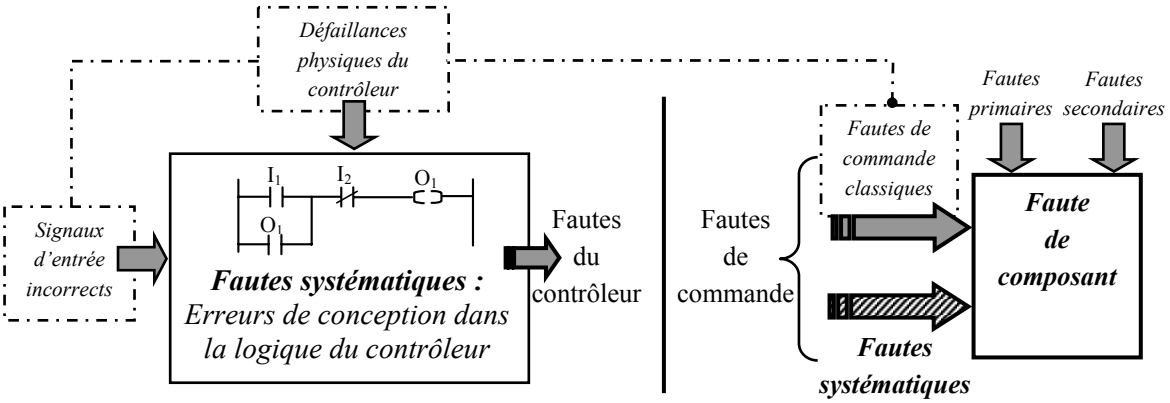


Figure 3.5. Différents types de fautes de commande

Ainsi, nous proposons dans cette approche un nouveau modèle générique de la faute de composant (Barragan, Faure et Papadopoulos, 2006). Il est représenté dans la figure 3.6.

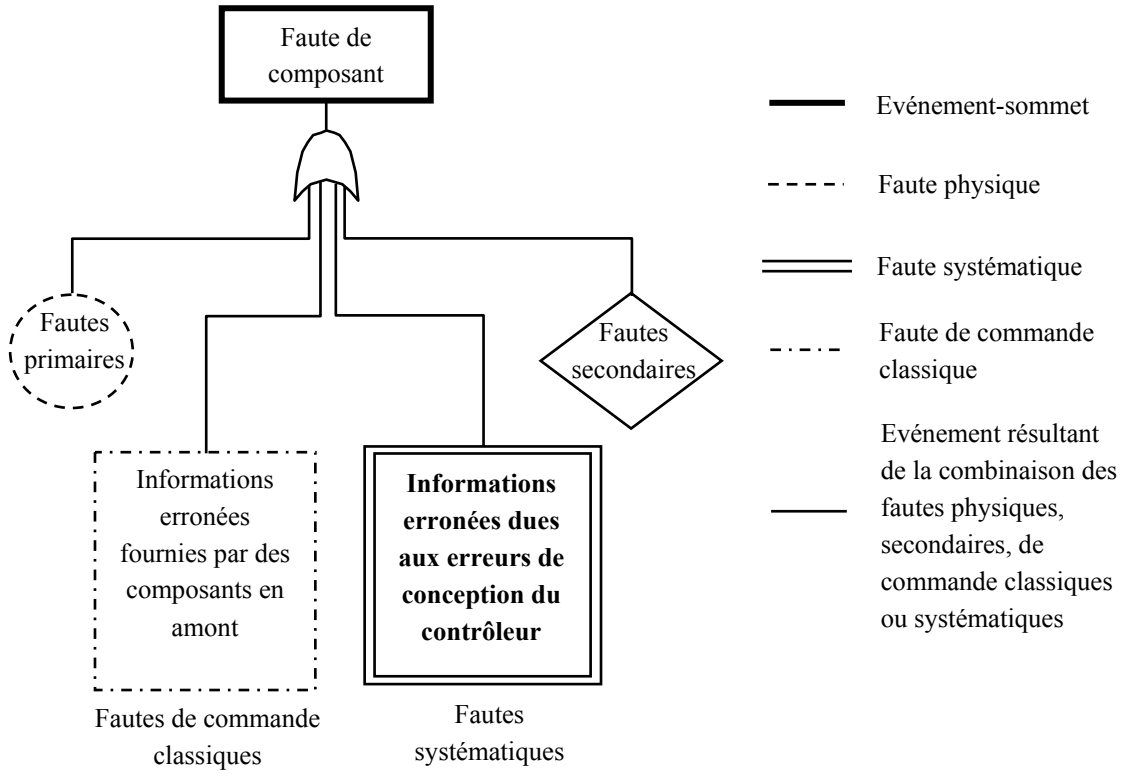


Figure 3.6. Nouveau modèle générique permettant d'inclure les fautes systématiques dans l'analyse de la faute de composant

Ce nouveau modèle générique comporte quatre types de fautes : fautes primaires, fautes secondaires, fautes de commande classiques (toutes les trois déjà prises en compte dans la structure traditionnelle de l'arbre et expliquées dans la section 1.2.2 du chapitre 2), et fautes systématiques.

Nous avons également adopté une convention graphique. Ainsi, un trait en pointillé est utilisé pour représenter les fautes primaires. Les fautes de commande classiques sont représentées par un trait mixte. Un trait double sert à représenter les fautes systématiques. Le trait continu simple est conservé pour les événements non développés, et les événements résultant d'une combinaison des fautes primaires, secondaires, de commande classiques ou systématiques, c'est à dire pour les événements intermédiaires ou pour l'événement-sommet (pour ce dernier le trait est plus épais).

Application du nouveau modèle générique au contrôleur logique

La validité du notre nouveau modèle générique sera montrée en détail à l'aide d'un exemple complet dans la section 4. Pourtant à titre illustratif, nous voulons montrer ici qu'il est possible d'appliquer le nouveau modèle de l'AdD pour un contrôleur logique, si on prend le contrôleur comme étant un composant. Ainsi, nous reprenons l'analyse du cas représenté dans le schéma de la figure 3.4. L'événement indésirable est « Le contrôleur maintient à tort la sortie O_1 à un (VRAI) ». L'AdD développé est montré en figure 3.7.

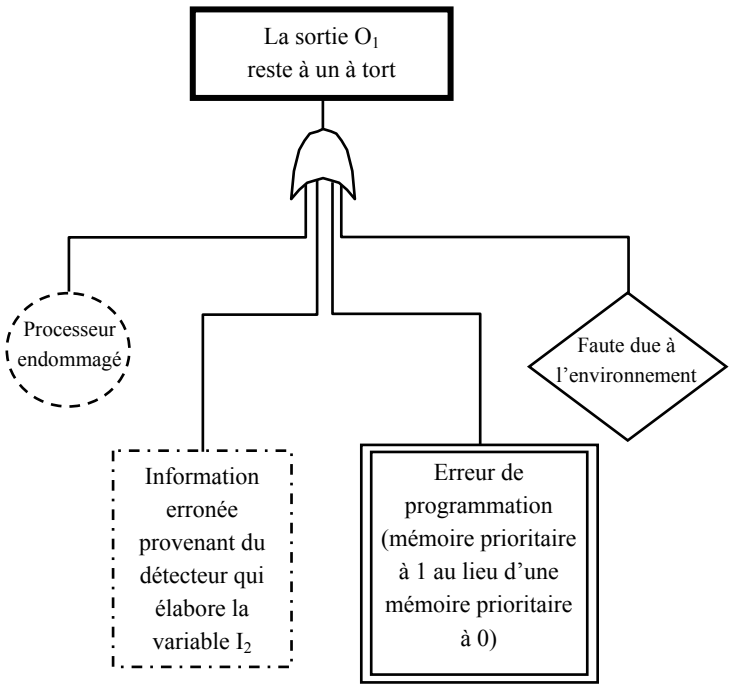


Figure 3.7. Application du nouveau modèle pour analyser les causes de la faute du contrôleur

Une porte OU est alors utilisée et on cherche les fautes primaires, secondaires, de commande classiques ou systématiques. « Processeur endommagé » est une faute primaire représentée dans un cercle en pointillés, selon la convention adoptée. « Faute due à l'environnement », est considérée comme une faute secondaire et par simplicité ne sera pas développée ici. La troisième faute « Information erronée du détecteur qui élabore la variable I_2» est une faute de commande classique car c'est la défaillance physique du détecteur qui en est la cause. Finalement, « Erreur de programmation » est la faute systématique. Elle réside dans la logique du contrôleur car une mémoire prioritaire à 1 est programmée au lieu d'une mémoire prioritaire à 0.

3 Description des fautes systématiques avec un vocabulaire de portes

Nous rappelons ici que l'objectif de l'approche que nous proposons dans ce travail de thèse, est de faciliter la tâche d'obtention des propriétés formelles des contrôleurs logiques. Pour cela, nous proposons d'inclure les fautes systématiques du contrôleur logique dans la structure de l'AdD et d'exprimer ensuite les propriétés formelles à vérifier. Cependant, les propriétés formelles ne sont pas toujours des invariants mais peuvent faire intervenir l'ordre logique des événements ou le temps physique. Par conséquent, l'AdD qui analyse les fautes systématiques du contrôleur doit être dynamique, c'est-à-dire qu'il doit permettre de représenter des séquences d'événements (prise en compte du temps logique) et le temps écoulé entre événements (temps physique).

Dans le cadre de ce travail de thèse, on adopte **l'utilisation de portes** pour représenter finement les fautes systématiques du contrôleur logique. Ces portes doivent alors permettre d'exprimer la relation de temps logique entre les événements (**portes temporelles**) et l'écart temporel entre deux événements (**portes temporisées**). Les entrées et sorties de ces portes (que nous appelons globalement **portes dynamiques**) seront des signaux logiques qui correspondent à des variables d'entrée/sortie du contrôleur logique ou à des combinaisons de ces variables, et non à des défaillances de composants physiques.

Nous avons présenté, dans la section 2.1 du chapitre deux, toute une gamme de portes dynamiques proposées par différents auteurs. Nous choisissons alors :

- de **réutiliser** les portes ***Priority AND*** et ***Priority OR*** ;
- de **définir deux nouvelles portes *WITHIN n*** et ***FORNEXT n***, à partir des portes WITHIN n et FORPAST n proposées par Palshikar (chapitre deux, sous-section 2.1.3). Même si ce choix ne peut être considéré comme absolument définitif, plusieurs traitements de cas (voir section 4 de ce chapitre et section 6.1 du cinquième chapitre) nous ont en effet montré que ce couple de portes, qui comprend une porte traduisant l'occurrence d'un événement dans un intervalle de temps et une porte qui traduit le maintien d'un signal durant un intervalle de temps, permettait de modéliser les différentes défaillances de contrôleur fonction du temps physique.

Utiliser un vocabulaire de portes pour traiter les fautes systématiques permet de préserver la simplicité graphique de l'AdD. Une description plus détaillée des portes dynamiques choisies est fournie dans ce qui suit.

3.1 La porte Priority AND (PAND)

Le symbole de cette porte et deux chronogrammes, décrivant de façon graphique informelle son comportement, sont montrés dans la figure 3.8. Cette figure montre le cas d'une porte à deux entrées et une sortie. La sortie de la porte est une défaillance résultant d'une séquence d'événements prédéfinie ; cette sortie est vraie si les événements d'entrée de la porte apparaissent dans un ordre spécifique, normalement de gauche à droite. Les flèches verticales dans les chronogrammes de la figure représentent les fronts montants des signaux.

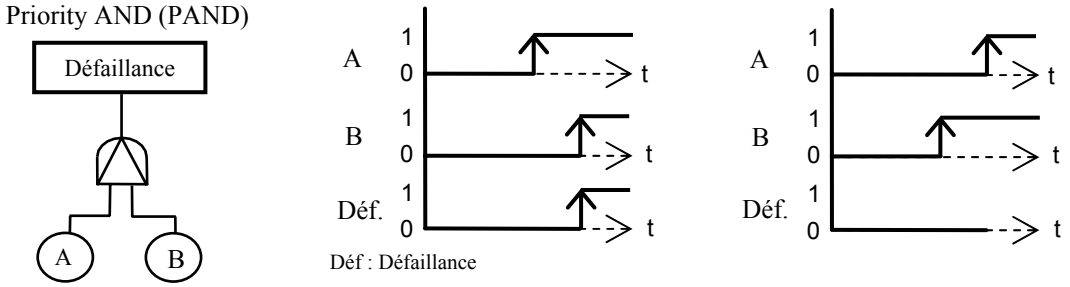


Figure 3.8. Porte Priority AND à deux entrées

3.2 Porte Priority OR (POR) avec condition de priorité sur une des entrées

La figure 3.9 présente le symbole d'une porte POR à deux entrées et deux chronogrammes, décrivant de façon informelle le comportement de cette porte. La porte POR établit un ordre entre les événements d'entrée, total si la porte possède deux entrées, partiel si elle en comporte plus. La sortie de la porte est une défaillance résultant d'un seul événement se produisant dans une condition prédéfinie.

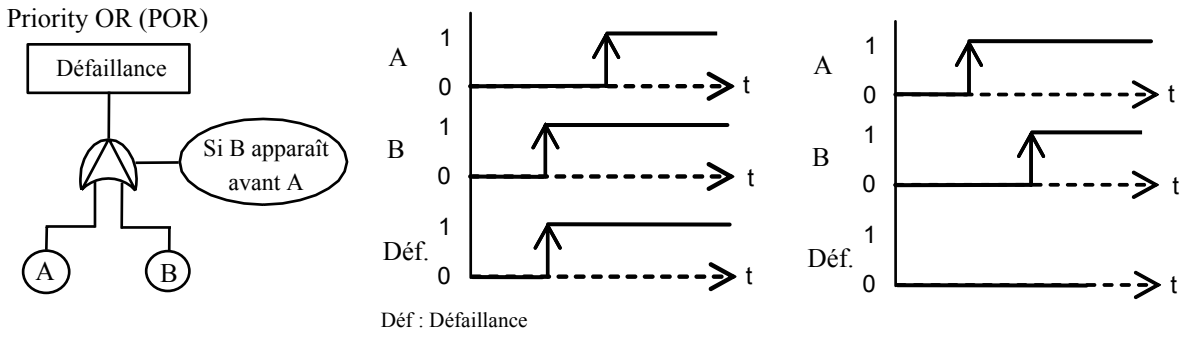


Figure 3.9. Porte Priority OR à deux entrées

Ici, l'entrée B est choisie comme l'entrée qui génère la défaillance (**entrée conditionnée**). La défaillance est VRAIE si et seulement si l'événement associé à l'entrée qui porte la condition de priorité se produit avant l'autre entrée, qui peut à son tour se produire après ou pas. Dans le cas contraire, si l'autre entrée est déjà VRAIE quand B apparaît, la défaillance ne se produit pas.

3.3 Les portes temporisées

Il faut préciser que, selon ce qui a été exposé dans la section 2.1.3 du chapitre 2, la sémantique des portes temporisées est originellement définie par Palshikar avec une logique du passé, c'est-à-dire qu'on recherche (sous-entendu « dans le passé ») les causes d'une défaillance qui vient de se produire. Cette sémantique est intéressante dans le cadre d'une *approche de diagnostic*.

Par contre, dans ce travail de thèse basé sur la prévision des fautes, nous nous intéressons aux conséquences indésirables (événements indésirables) de séquences d'événements. Notre démarche d'analyse consiste donc à prévoir la conséquence **dans le futur** d'une séquence d'événements.

Ceci explique qu'on ne puisse pas utiliser les portes FORPAST n et WITHIN n mais qu'il nous faut définir deux nouvelles portes qui expriment les mêmes concepts généraux (occurrence d'un événement dans un intervalle de temps donné, et maintien d'un signal durant un intervalle de temps précisé) mais qui sont basées sur une logique du futur.

Nous définissons donc les portes temporisées :

- **FORNEXT n** (au lieu de FORPAST n) : La sortie est VRAIE si l'entrée reste VRAIE pendant n unités de temps **après** son front montant.

- **WITHIN n** (non renommée mais redéfinie) : La sortie est VRAIE si la variable d'entrée devient VRAIE au moins une fois à l'intérieur d'un intervalle de n unités de temps **après** l'instant courant.

La figure 3.10 montre les portes ainsi renommées et/ou redéfinies.

Prise isolément, chacune des ces portes n'exprime pas une défaillance mais permet d'exprimer facilement le comportement temporisé d'une variable logique. Une défaillance sera exprimée à l'aide de plusieurs portes comme il sera montré à la section suivante.

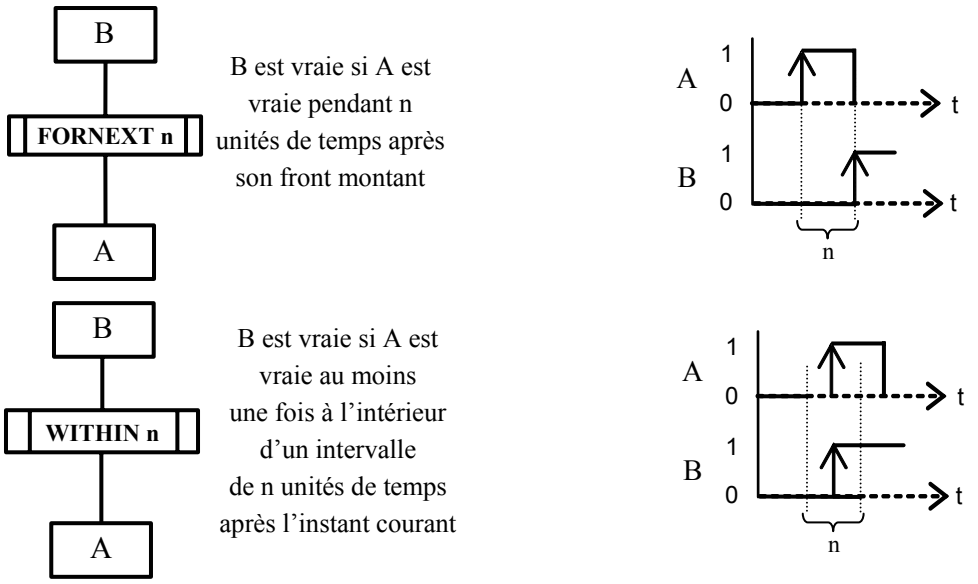


Figure 3.10. Les portes temporisées choisies

4 Exemple d'application du nouveau modèle générique

L'application du nouveau modèle générique et des portes choisies est illustrée avec un exemple qui s'appuie sur la commande d'un préhenseur pneumatique. Ce préhenseur a pour origine un sous-ensemble d'un système réel, système Festo présent au laboratoire d'enseignement d'automatique de l'ENS de Cachan. Son rôle est de prendre un pignon dans un tiroir pour le monter sur un axe. Ceci est réalisé par une préhension par aspiration et des déplacements à l'aide de deux vérins. Un contrôleur logique commande le système. Il existe différents modes de fonctionnement mais dans le cas de notre étude nous nous sommes intéressés uniquement au mode de marche automatique.

La partie opérative du préhenseur pneumatique comporte 3 chaînes fonctionnelles :
Mouvement horizontal :

1 vérin double effet.
1 distributeur 5/2 bistable.
2 détecteurs magnétiques (fin de course).

Mouvement vertical :

1 vérin double effet.
1 distributeur 5/2 monostable.
2 détecteurs magnétiques (fin de course).

Aspiration :

Ventouses.
1 système venturi.
1 distributeur 5/2 monostable.

Toutes les entrées/sorties sont récapitulées sur la figure 3.11. Il convient de remarquer qu'il n'y a pas de commande *Monter,* étant donné la technologie du distributeur commandant le vérin vertical. Cette configuration est retenue pour assurer la sécurité du système car, en l'absence d'énergie pneumatique, le vérin vertical revient en position haute pour éviter de possibles collisions lors du mouvement horizontal.

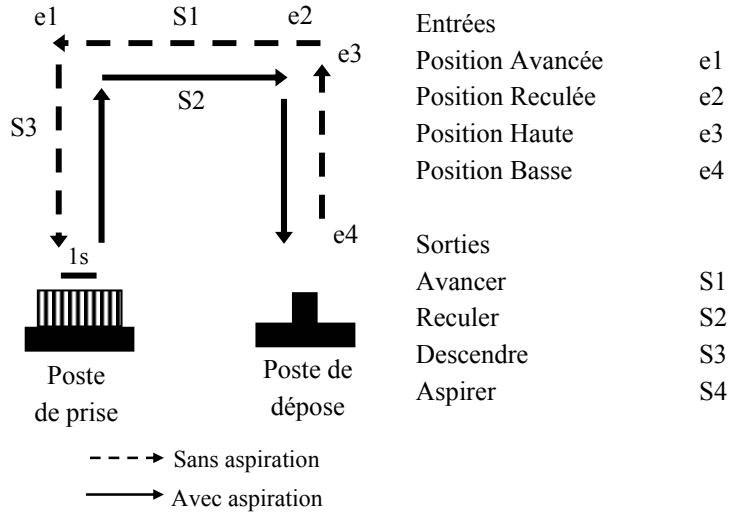


Figure 3.11. Les entrées/sorties du préhenseur pneumatique

Dans ce qui suit, deux cas seront analysés. Le premier cas correspond à l'analyse du non-respect de la condition de sûreté « l'aspiration ne doit jamais être coupée avant que le préhenseur soit en position de dépose ». Le second s'intéresse aux conditions de non-respect de la spécification fonctionnelle « le vérin doit rester 1 seconde au poste de prise pour saisir la pièce ». Pour chacun d'eux, nous montrerons le modèle générique et l'utilisation de portes appartenant au vocabulaire retenu. Le premier cas montre l'utilisation d'une porte temporelle et le second celle d'une porte temporelle et d'une porte temporisée.

4.1 Première analyse : Chute du pignon durant le déplacement entre le poste de prise et le poste de dépose

L'Evénement Indésirable (EI) à analyser est : « Chute du pignon durant le déplacement poste de prise – poste de dépose». Comme l'on peut voir (figure 3.12), trois causes produisent l'EI.

La première cause, « Pièce trop lourde » est une défaillance physique, donc c'est un événement de base élémentaire (représenté dans un cercle en pointillés). La seconde cause « Collision du pignon avec l'environnement », est une faute secondaire et ne sera pas développée ici. La troisième « Défaillance de l'aspiration durant le déplacement poste de prise – poste de dépose » est un événement intermédiaire qu'il faut développer. Dans l'état actuel de l'analyse, on ne considère pas la distinction entre faute de commande classique et systématique.

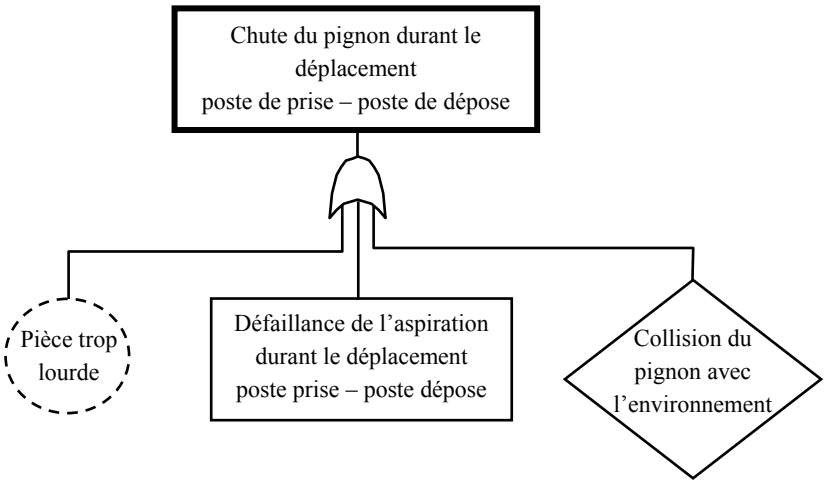


Figure 3.12. AdD pour «Chute du pignon durant le déplacement poste de prise - poste de dépose»

L'événement intermédiaire à développer peut être considéré comme la défaillance du composant « aspiration », et on applique de nouveau le modèle générique. L'arbre correspondant à « Défaillance de l'aspiration durant le déplacement poste de prise – poste de dépose » est montré dans la figure 3.13. Trois causes couplées par une porte OU peuvent être la cause de cette défaillance, car on ne considère pas de faute due à l'environnement.

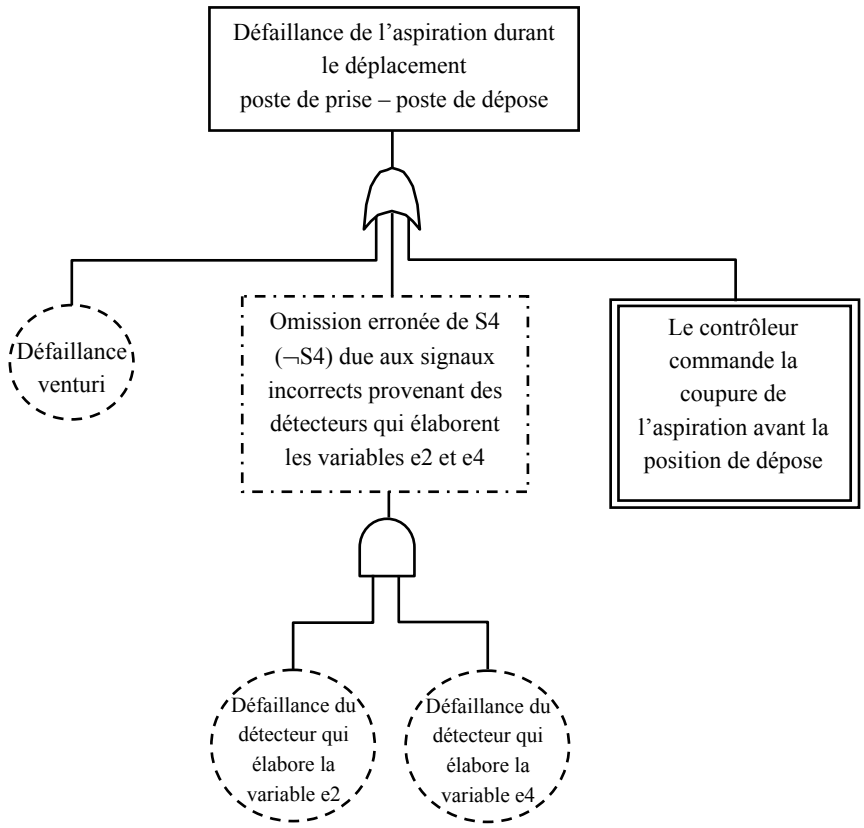


Figure 3.13. AdD pour « Défaillance de l'aspiration durant le déplacement poste de prise – poste de dépose »

Tout d'abord, « défaillance venturi » est une faute primaire, représentée dans un cercle en pointillé. La deuxième faute « Omission erronée de S4 (¬S4) due aux signaux incorrects provenant des détecteurs qui élaborent les variables e2 et e4», est considérée comme une faute de commande classique, c'est-à-dire qui est la conséquence de la défaillance d'un composant qui se trouve en amont du contrôleur. Dans ce cas précis, la faute provient de la défaillance du couple de détecteurs de position qui élaborent les variables e2 et e4. A cause de cette défaillance, les détecteurs rapportent de façon erronée au contrôleur que le préhenseur est en position de dépose, alors qu'il ne l'est pas, et le contrôleur délivre la commande de couper l'aspiration. La défaillance de chaque détecteur est représentée dans un cercle en pointillé car il s'agit d'une faute physique. Les deux défaillances sont alors reliées par une porte ET. Finalement, la dernière cause est une faute systématique car elle provient de la logique du contrôleur qui délivre la commande de couper l'aspiration avant la position de dépose. Elle est représentée par un rectangle à double trait.

L'analyse détaillée de cette faute est montrée avec l'AdD de la figure 3.14. On voit bien que, toutes les causes montrées dans son analyse sont entourées avec un trait double, pour montrer leur appartenance au développement de la faute systématique. Ici, le couple de détecteurs travaille correctement mais le contrôleur arrête l'aspiration avant que le préhenseur soit dans la position correcte : la position de dépose. Un chronogramme inclus dans la figure illustre un comportement défaillant.

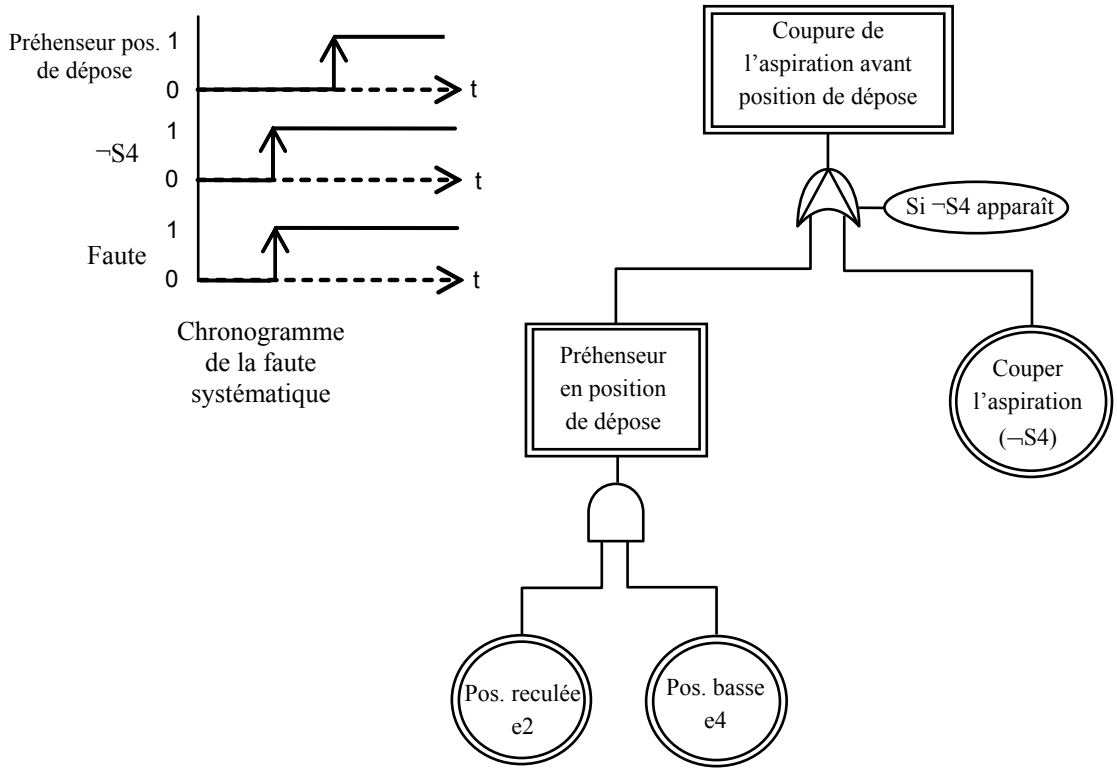


Figure 3.14. Analyse de la faute systématique « Coupure de l'aspiration avant position de dépose »

Dans ce cas, la commande est émise de façon erronée et contredit la spécification de sûreté qui établit que : « l'aspiration ne doit jamais être coupée avant que le préhenseur soit en position de dépose ». Le terme « avant » suggère l'idée d'un ordre dans les événements. On utilise ici une porte temporelle Priority OR pour exprimer cette défaillance.

4.2 Deuxième analyse : Faute lors de la saisie d'une pièce

Le second cas porte sur une faute lors de la saisie d'une pièce. Une des spécifications du système impose au vérin vertical de se maintenir une seconde au poste de prise pour prendre le pignon, avant de quitter le poste. La faute consiste ici à ne pas respecter cette spécification. L'AdD détaillant les origines de cette faute est donné sur la figure 3.15.

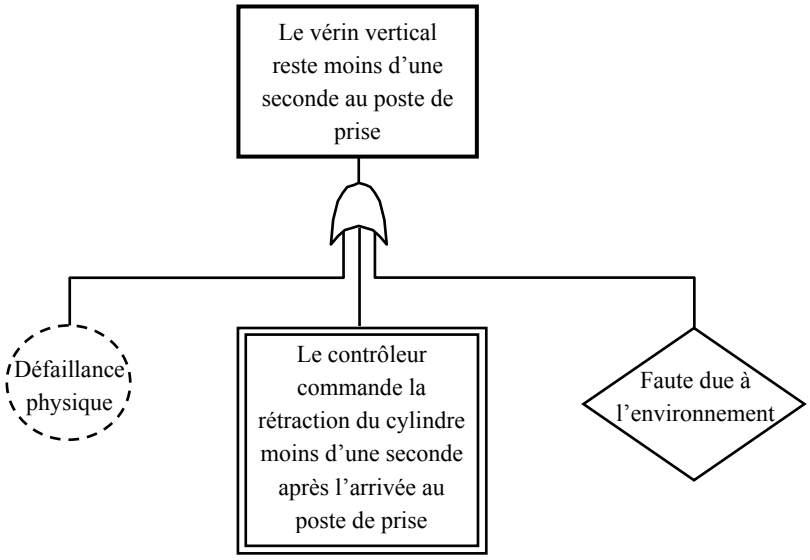


Figure 3.15. AdD pour la faute lors de la saisie d'une pièce

La faute en question est centrée sur le vérin vertical et dans ce cas là, on peut appliquer aussi le nouveau modèle générique. Donc, on utilise une porte OU et on cherche les fautes primaires, secondaires, de commande classiques ou systématiques. L'AdD de la figure 3.15 montre qu'une défaillance physique (telle qu'un fuite interne ou externe) peut produire la rétraction du vérin avant le délai. En outre, l'environnement peut contribuer aussi à produire la défaillance, donc on inclut cette cause, mais elle n'est pas développée par simplicité. La troisième faute à considérer provient surtout de la logique du contrôleur qui commande le distributeur du vérin, donc c'est une faute systématique. Aucun autre composant en amont du contrôleur dans le système est considéré intervenir, donc il n'y pas de faute de commande classique.

Maintenant, on centre l'analyse sur la faute systématique représentée dans un rectangle en double trait dans l'AdD de la figure 3.15. La faute est le résultat de commuter le distributeur et donc de produire la rétraction du vérin avant le temps imposé, donc c'est une sortie du contrôleur émise au mauvais moment. La faute est analysée avec l'AdD de la figure 3.16.

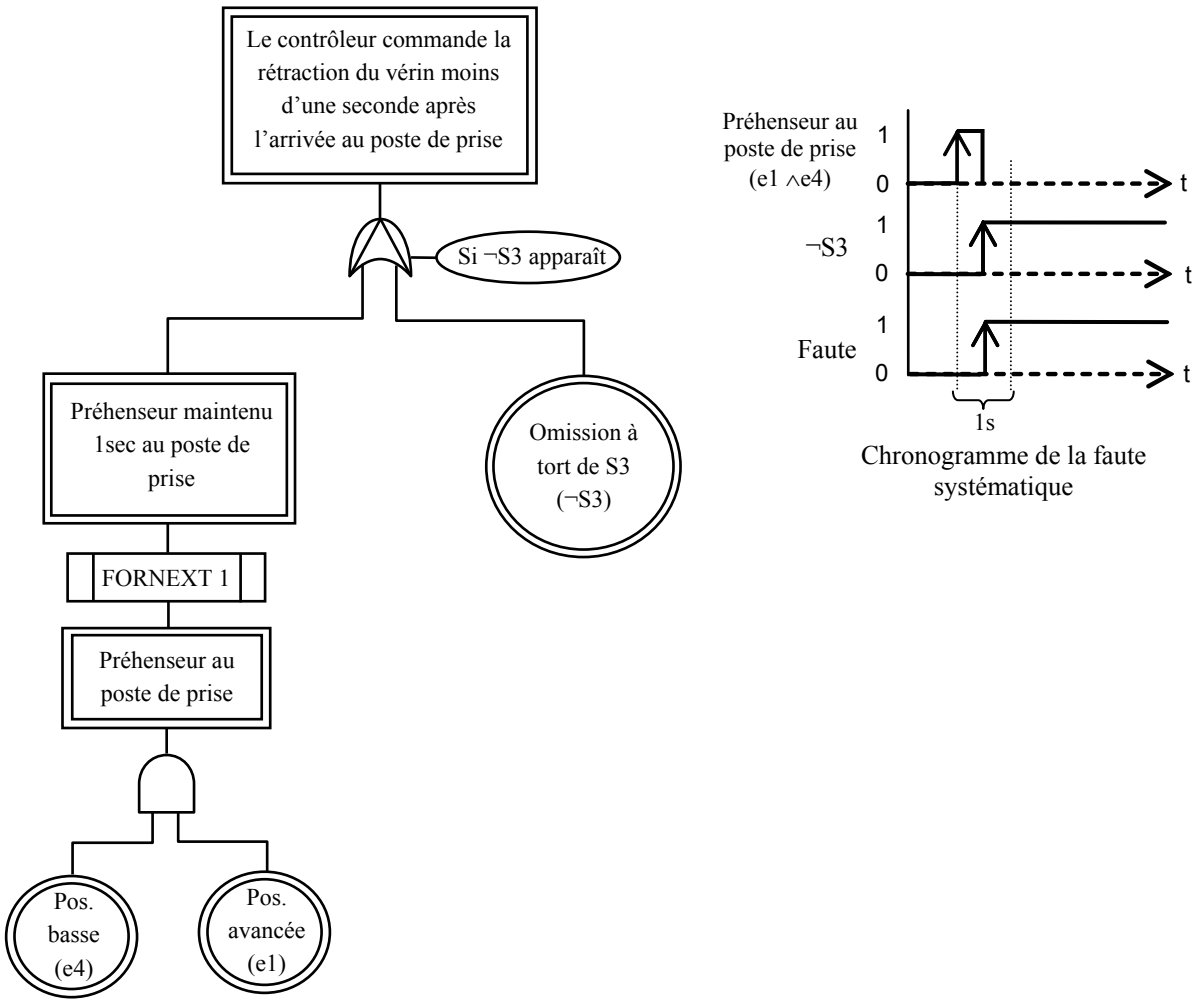


Figure 3.16. Analyse de la faute systématique «rétraction du vérin moins d'une seconde après l'arrivée au poste de prise»

Le chronogramme de droite illustre la séquence qui produit la faute. Pour la modéliser avec l'AdD, on fait appel à la porte POR et à la porte temporisée FORNEXT n (avec n = 1). Comme on peut voir dans la figure, l'entrée gauche de la porte POR établit que le préhenseur est maintenu au poste de prise pendant une seconde.

Pour établir la condition « Préhenseur maintenu 1 sec au poste de prise », nous utilisons une porte temporisée FORNEXT 1, dont l'entrée est la combinaison logique (sortie de la porte ET) des variables e1 et e4. L'entrée droite de la porte POR est l'entrée conditionnée et elle établit l'absence de commande de descente du vérin, ce qui provoque sa montée. La faute se produit si cette entrée se produit avant l'autre.

Ces deux exemples ont montré l'intérêt du modèle générique proposé pour l'introduction dans l'analyse par AdD des fautes systématiques, et du vocabulaire de portes retenu. Cependant, la sémantique formelle de ces portes n'a pas été définie. L'objet du prochain chapitre est de combler cette lacune, et de proposer une représentation formelle du comportement des portes temporelles et temporisées retenues.

Chapitre 4.

Formalisation des comportements décrits par les portes élémentaires

1 Introduction

Au chapitre précèdent, nous avons proposé un nouveau modèle générique de la faute de composant, qui inclut les fautes systématiques attribuées au contrôleur logique. Ceci est un point important pour obtenir, à partir d'un AdD, des propriétés formelles du contrôleur en vue de sa vérification. Cependant, une fois repérée, la faute systématique doit être analysée afin d'identifier les causes élémentaires qui la produisent.

Pour cette analyse des fautes systématiques, nous avons proposé d'utiliser un vocabulaire de portes (portes statiques, temporelles et temporisées), dont nous n'avons pas, jusqu'à présent, donné la sémantique précise. Ces portes serviront à construire un sous-arbre qui représentera le développement de la faute systématique (figure 4.1), et qui, s'il ne contient pas que de portes statiques, sera qualifié d'arbre dynamique.

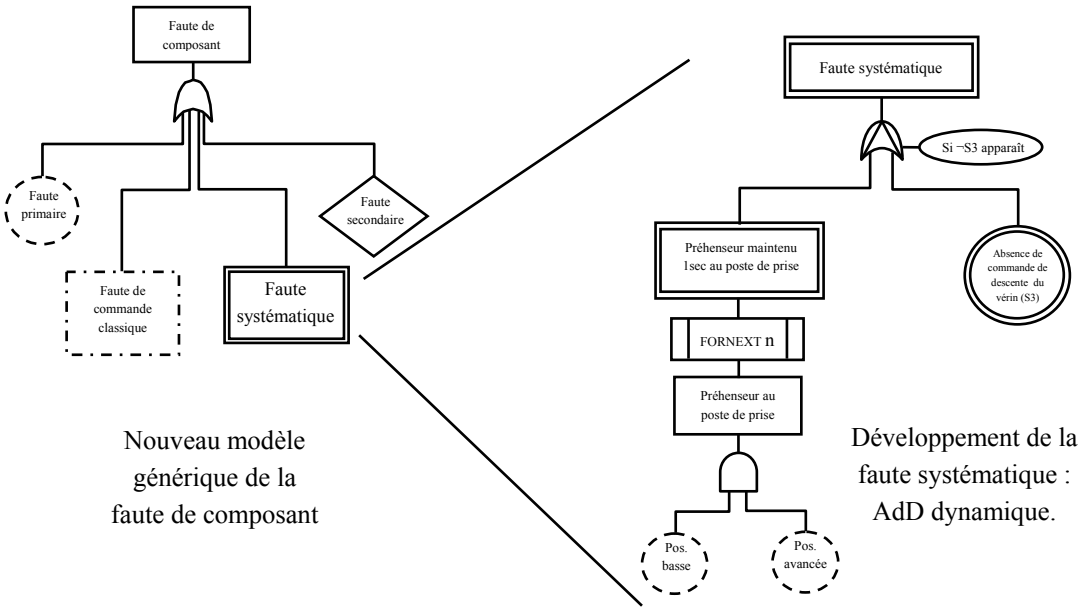


Figure 4.1. Développement d'une faute systématique

Il convient maintenant de proposer une méthode permettant de passer de l'AdD dynamique aux propriétés formelles du contrôleur logique (figure 4.2).

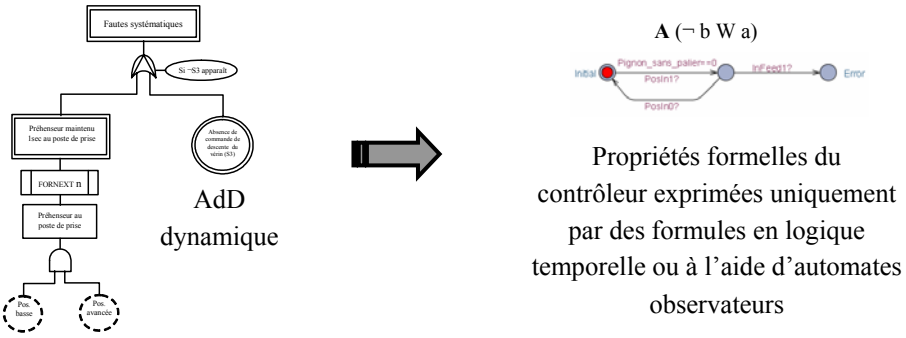


Figure 4.2. Passage de l'AdD dynamique aux propriétés formelles du contrôleur logique

Pour ce faire, nous allons commencer par traiter les portes statiques et dynamiques individuellement pour ensuite traiter l'AdD dynamique dans sa globalité. Nous proposons donc :

- une représentation formelle du comportement décrit par **chacune des portes** pouvant intervenir dans un sous-arbre exprimant une faute systématique : portes statiques, temporelles et temporisées. Pour les portes statiques et temporelles, il est possible de déduire de cette modélisation formelle du comportement une propriété formelle à vérifier. On rappelle que, par contre, une porte temporisée prise isolément ne traduit pas une faute ; il n'est donc pas possible d'obtenir une propriété formelle à partir du modèle formel d'une seule porte temporisée.
- une technique d'analyse de la cohérence **d'une association de portes** ;

- une **représentation formelle du comportement décrit par une association** cohérente de portes, permettant d'obtenir une propriété formelle qui exprime que le comportement défaillant décrit par cette association ne se produira pas.

L'ensemble de ces trois points nous permet d'accomplir les étapes 2 et 3 de la méthode énoncée dans la section 3 du deuxième chapitre et rappelée dans la figure ci-contre.

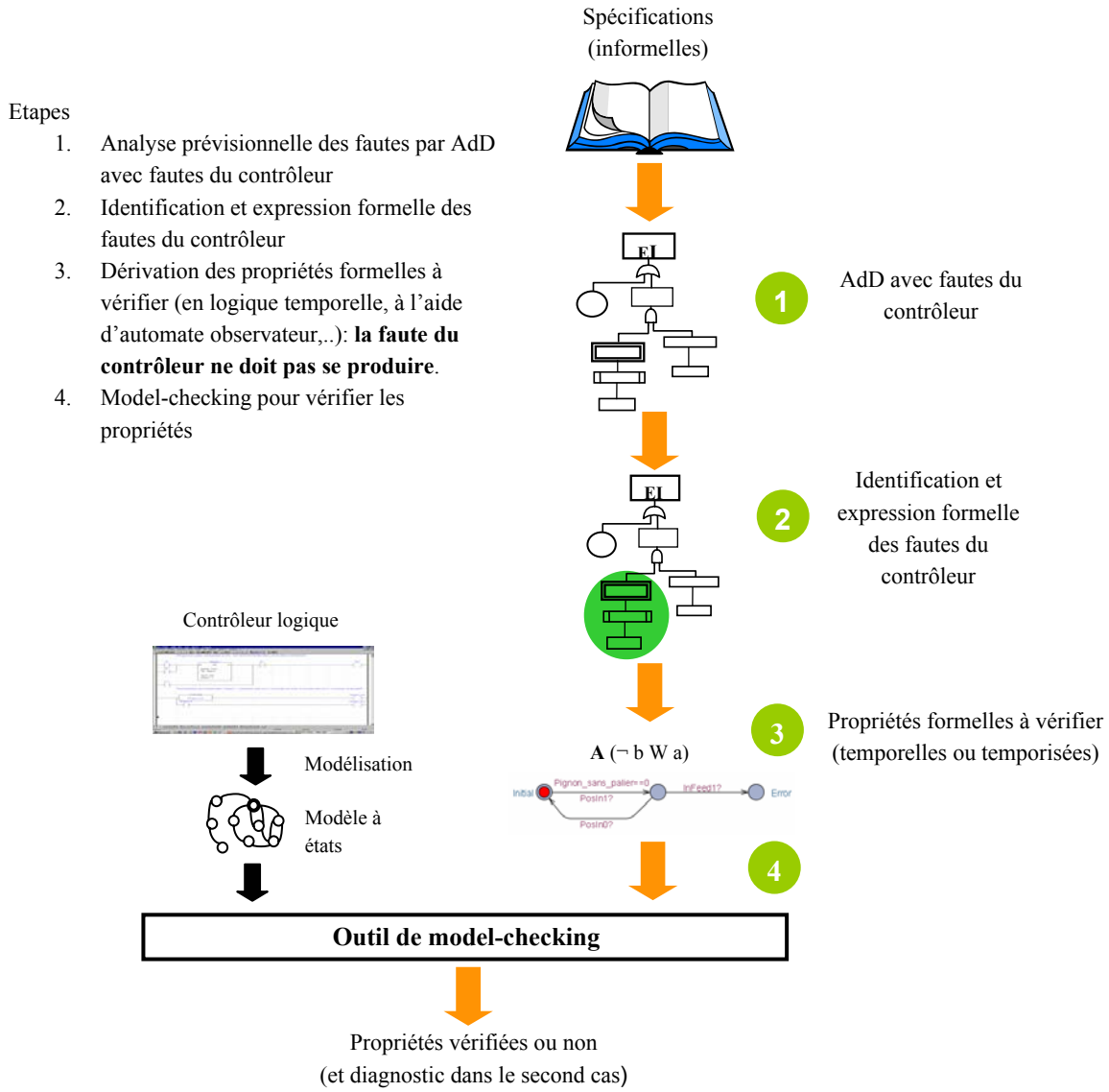


Figure 4.3. Vue générale de notre méthode

L'objectif de ce quatrième chapitre est de traiter le premier point de la liste précédente, c'est-à-dire de proposer une sémantique formelle pour chacune des portes statiques et dynamiques retenues, et de montrer comment, à partir des représentations formelles des portes statiques et temporelles, il est possible d'obtenir des propriétés formelles acceptables par des model-checkers existants (figure 4.4). Ce point constitue la troisième proposition que nous avons énoncée en fin du chapitre deux. Les deux autres points de la liste précédente, relatifs aux

associations de portes et qui constituent notre quatrième proposition, seront traités au chapitre cinq.

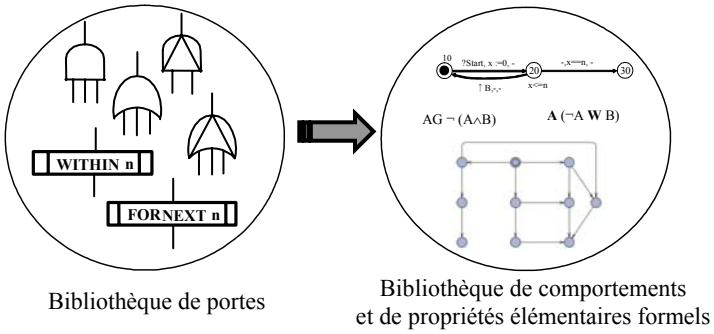


Figure 4.4. Objectif du chapitre : passage d'une bibliothèque de portes élémentaires à une bibliothèque de comportements et de propriétés formels

Ce chapitre est alors construit de la façon suivante. La section 2 décrit de façon générale la méthode à suivre pour obtenir le modèle formel d'une porte et en déduire la propriété formelle correspondante, pour les portes statiques et temporelles, et présente les formalismes retenus. Les trois autres sections sont respectivement consacrées à la construction des modèles formels des portes statiques, des portes temporelles et enfin des portes temporisées.

2 Méthode d'obtention des modèles formels des portes et des propriétés formelles déduites des portes statiques et temporelles

Nous rappelons tout d'abord que les feuilles d'un AdD dynamique exprimant une faute systématique sont des signaux logiques représentant l'évolution au cours du temps de variables d'entrée/sortie du contrôleur logique. Dans ce chapitre, les entrées d'une porte sont donc de tels signaux et la sortie est :

- un signal logique VRAI quand la faute s'est produite, pour les portes statiques et temporelles ;
- un signal logique fonction du temps physique mais qui n'exprime pas une faute, pour les portes temporisées.

Pour obtenir le modèle formel d'une porte et, si cela est possible, la propriété formelle exprimant l'absence de faute, nous proposons la méthode générale suivante (figure 4.5):

1. Exprimer de façon informelle le comportement de la porte, par exemple avec un chronogramme.
2. Exprimer ensuite de façon formelle ce comportement, dans un formalisme compatible avec les outils de model-checking (formule en logique temporelle ou automate observateur).
3. Pour les portes statiques et temporelles, à partir du modèle formel de la porte, établir la propriété à vérifier. La propriété énonce que la faute ne se produit pas.

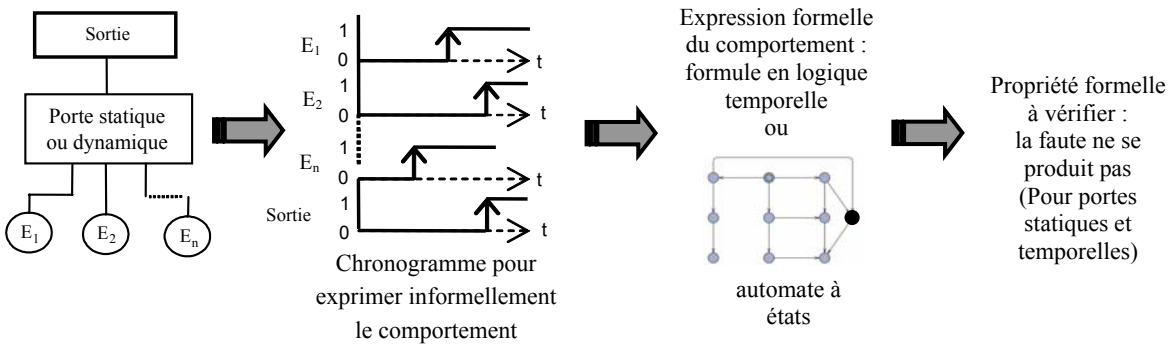


Figure 4.5. Méthode d'obtention de modèles formels à partir d'une porte

2.1 Choix du formalisme de représentation

Deux solutions sont envisageables pour exprimer formellement le comportement des portes et, lorsque cela est possible (portes statiques et temporelles), la propriété à vérifier. La première consiste à utiliser une formule en logique temporelle (sous-section 4.2.2 du chapitre 1) ; cette solution sera retenue pour les portes statiques et temporelles. L'autre solution, que nous retiendrons également pour les portes temporelles ainsi que pour les portes temporisées, est d'utiliser une représentation sous la forme d'un automate à états finis. Dans le cas des portes temporelles, le comportement défaillant et la propriété seront alors obtenus en utilisant ce modèle formel comme un automate observateur (premier chapitre, sous-section 4.2.3), c'est-à-dire un automate chargé de détecter l'occurrence d'événements sans effectuer aucun changement sur le système observé de la façon suivante (figure 4.6) :

— traduction du comportement défaillant par :
- un automate observateur ;
- **PLUS une condition sur l'activité de l'état (ou des états)** de cet automate où se produit la faute.

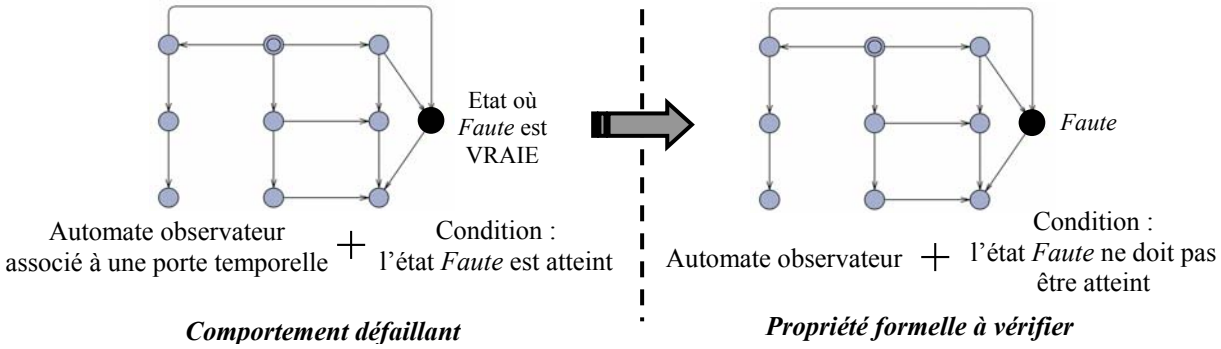


Figure 4.6. Faute et propriété formelles utilisant un automate observateur

— écriture de la propriété formelle déduite de la porte sous la forme :
- du même automate observateur ;
- **PLUS une condition** exprimant que l'état (ou les états) où se produit la faute n'est pas (ou ne sont pas) atteint(s).

Deux classes d'automates à états finis vont être utilisées dans le cadre de notre travail : les automates non temporisés et les automates temporisés. Ces derniers sont un langage basé sur les automates à états finis mais étendus avec des horloges et de contraintes de temps (Alur et Dill, 1994). Ci-dessous, nous allons donner la définition des automates à états finis non temporisés et temporisés.

2.2 Les automates à états non temporisés

Un automate non temporisé (Schnoebelen et al., 1999) est un 4-uplet $<L,l_0,A,T>$, où :
- L est un ensemble fini d'états,
- l_0 est l'état initial de l'automate,
- A est un ensemble fini d'actions,
- $T \subseteq L \times A \times L$ est l'ensemble des transitions.

L'automate de la figure 4.7 illustre ces éléments :
$L = \{Initial, l_1, l_2\}$;
$l_0 = Initial$;
$A = \{A_1, A_2\}$;
$T = \{(Initial, A_1, l_1), (l_1, A_2, l_2)\}$

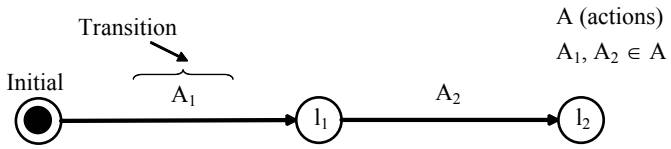


Figure 4.7. Automate à états non temporisé

2.3 Les automates à états temporisés

La définition de la sémantique des automates temporisés est présentée ici d'après (Behrmann et al. 2004). Les notations suivantes sont utilisées : C est l'ensemble des horloges et B(C) est l'ensemble des conjonctions sur des conditions simples de la forme x□c (condition portant sur une seule horloge), avec $x \in C$, $c \in N$ et $\square \in \{<, \leq, =, \geq, >\}$. Un automate temporisé est un 6-uplet $<L,l_0,C,A,T,I>$, où :
- L est un ensemble fini d'états,
- l_0 est l'état initial,
- C est l'ensemble des horloges,
- A est l'ensemble des actions,

- $T \subseteq L \times A \times B(C) \times r \times L$ est l'ensemble des transitions avec trois composants pour chaque transition (chaque composant est optionnel):
    - une action,
    - une garde (qui exprime une condition dans B(C) sur la valeur d'une horloge qui doit être satisfaite pour franchir la transition),
    - un ensemble r d'horloges à initialiser (avec $r \subseteq C$).
- $I: L \rightarrow B(C)$ assigne des invariants aux états. Un invariant est une condition sur valeurs d'horloge qui doit être toujours satisfaite quand l'état auquel l'invariant est associé est actif.

La figure 4.8 montre l'exemple d'un automate temporisé. La première transition, qui permet de passer de l'état *Initial* à l'état l_1 est composée d'une action A_1 et de l'initialisation de l'horloge x. Dans cette transition il n'y a pas de garde (à sa place on met un trait d'union pour ne pas laisser l'espace vide). L'invariant $x \leq 5$ est associé à l'état l_1. La deuxième transition, qui permet de passer de l'état l_1 à l'état l_2 est composée d'une action A_2 et d'une garde (x==5) ; il n'y a pas d'initialisation d'horloge (trait d'union à sa place). Si cette garde est respectée et que l'action se produit la transition peut être franchie.

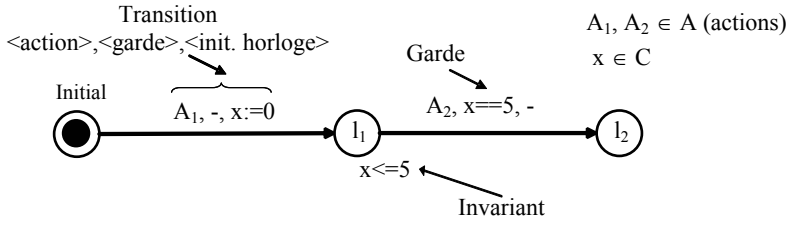


Figure 4.8. Automate à états temporisé

3 Obtention de propriétés formelles à partir de portes statiques

Nous traitons deux portes statiques dans cette section : la porte OU et la porte ET, mais la démarche est applicable aux autres portes statiques. La faute, c'est-à-dire la sortie de la porte, est une fonction purement combinatoire de ses entrées. Elle ne dépend ni du temps physique, ni de la séquence logique des événements. Ainsi, à partir de ces portes, il est possible d'exprimer en logique temporelle CTL, des propriétés de type invariants. Nous rappelons ici qu'en vérification formelle un invariant est une propriété qui doit être vraie à chaque instant.

La propriété à vérifier doit donc établir que pour tous les chemins, tous les états de l'automate modélisant le contrôleur vérifient que l'expression combinatoire issue de la porte (la faute) ne doit jamais se produire (jamais être VRAIE). C'est à dire, en CTL :

AG ¬ (Expression combinatoire de la porte)

Pour une porte statique, le passage de l'AdD à la propriété en logique temporelle se fait par l'écriture de cette expression (Henry et Faure, 2003 ; Barragan et Faure, 2005). La figure 4.9 montre les propriétés dérivées des portes ET et OU à deux entrées.

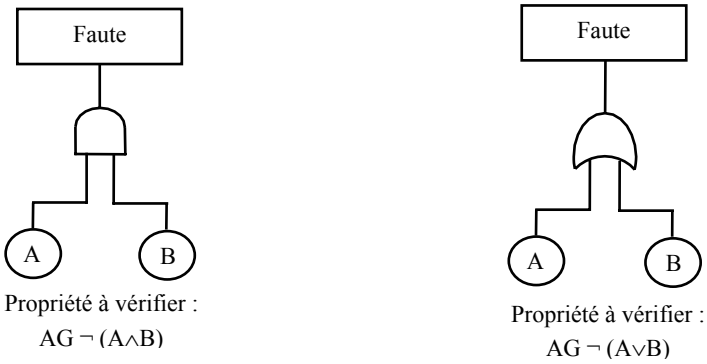


Figure 4.9. Propriétés dérivées des portes statiques ET et OU à deux entrées

4 Obtention de propriétés formelles à partir de portes temporelles

Cette section traite de l'obtention des propriétés formelles à partir des portes PAND et POR, seules portes temporelles retenues dans ce travail. Pour les deux portes, nous avons déjà décrit de façon graphique informelle le comportement au chapitre 3.

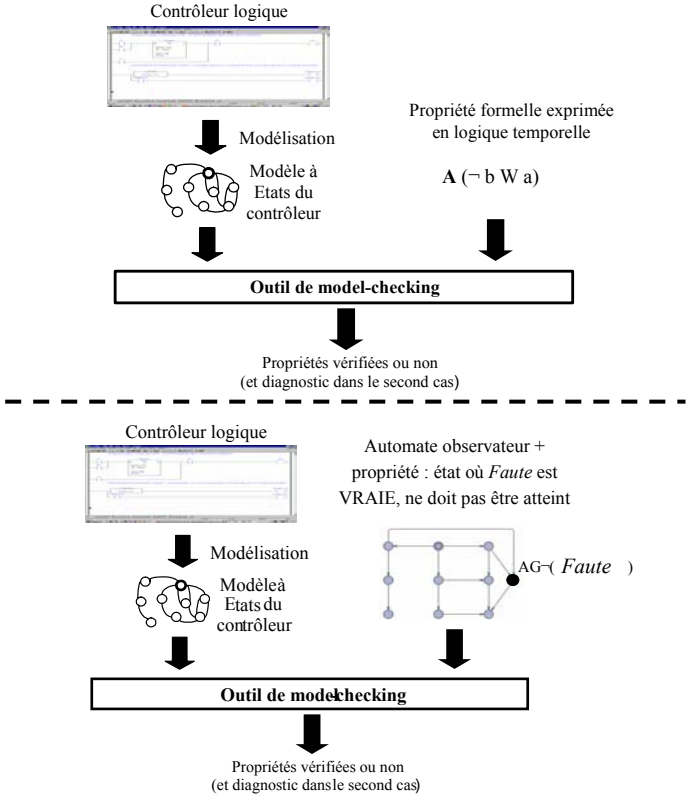


Figure 4.10. Les deux manières d'exprimer une propriété déduite d'une porte temporelle

Dans ce qui suit, nous exprimerons de façon formelle le comportement de chaque porte et ensuite, à partir du comportement défini formellement, nous déduirons la propriété à vérifier. Le comportement de chaque porte et la propriété déduite seront exprimés de deux manières : par une formule en logique temporelle CTL et à l'aide d'un automate observateur non temporisé (figure 4.10).

Le choix de l'un ou l'autre de ces modes d'expression lors de la vérification est laissé à l'utilisateur.

4.1 La porte PAND

4.1.1 Expression formelle du comportement et de la propriété déduite en CTL

En supposant que toutes les entrées de la porte sont initialement à l'état FAUX, l'expression de la faute énonce qu'il existe un état où la première entrée est VRAIE et qu'à partir de cet état il existe un chemin qui contient un état où les deux premières entrées sont vraies et ainsi de suite. Par exemple, pour le cas d'une PAND à trois entrées (A, B, C), l'expression du comportement défaillant est la suivante :

$$A \Rightarrow \mathbf{EF}\ (AB \Rightarrow \mathbf{EF}\ ABC)$$

Une représentation graphique d'une évolution conduisant à la faute est montrée en figure 4.11.

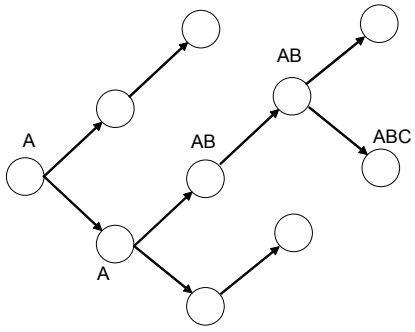


Figure 4.11. Représentation graphique d'une évolution conduisant à la faute

La propriété formelle déduite doit alors établir que ce type d'exécution n'est jamais VRAI. Donc :

$$\mathbf{AG}\ \neg\ (A \Rightarrow \mathbf{EF}\ (AB \Rightarrow \mathbf{EF}\ ABC))$$

Ce qui établit qu'il n'y pas d'état où A est VRAIE et à partir duquel il existe un chemin qui contient un état où AB est VRAIE et à partir duquel il est existe un chemin où ABC est VRAIE.

4.1.2 Expression formelle du comportement et de la propriété déduite à l'aide d'automates observateurs non temporisés

Prenons le cas général d'une porte PAND (figure 4.12) à *n* entrées différentes et une sortie (avec $n \in N$) ; ce sera la base pour ensuite analyser les cas particuliers d'une porte à deux et trois entrées.

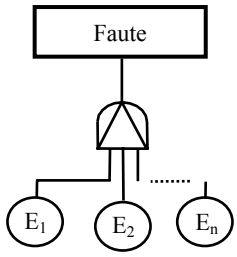


Figure 4.12. Porte PAND à n entrées

Nous allons montrer ci-après comment construire systématiquement l'automate observateur associé à cette porte. Cette construction est basée sur la méthode suivante, divisée en quatre étapes :

- Etape 1. Générer tous les automates modélisant les séquences élémentaires possibles. En supposant que toutes les entrées sont initialement à l'état FAUX et en ne considérant que les événements du type front montant, il existe *n*! séquences élémentaires possibles du type $\uparrow E_1 \uparrow E_2 \uparrow E_{n-1} \uparrow E_n$, où chaque événement apparaît une et une seule fois dans la séquence.
- Etape 2. Fusionner les états correspondant aux mêmes valeurs des entrées et à la même valeur de la sortie (faute ou absence de faute). Un seul automate est alors obtenu. Il contient une seule séquence conduisant à un état absorbant qui représente l'existence de la faute et plusieurs séquences conduisant toutes à un même état absorbant pour lequel la faute ne s'est pas produite.
- Etape 3. Fusionner les états à partir desquels il n'est pas possible d'atteindre l'état absorbant qui représente l'existence de la faute ; l'état absorbant résultant de cette fusion représentera l'impossibilité de faute.
- Etape 4. Rajouter, sauf pour les états absorbants, les évolutions qui correspondent aux passages à l'état FAUX possibles des (n-1) premières variables. Ces évolutions comportent des transitions dont les actions sont des événements du type front descendant et éventuellement des états intermédiaires ne représentant pas l'existence ou l'absence de faute, mais des combinaisons des valeurs des entrées de la porte. Elles sont nécessaires pour prendre en compte la contrainte exposée précédemment : la faute ne se produit que si les entrées restent vraies après leur front montant ; la suppression de cette contrainte conduit bien évidemment à ne pas effectuer cette étape.

Remarque : cette modélisation suppose:

- que les entrées sont toutes fausses dès l'état initial ;
- qu'il n'y a pas de changements de valeur simultanés des entrées.

Cette méthode est illustrée ci-après sur les exemples des portes PAND à deux et à trois entrées.

Cas 1 : porte PAND à deux entrées

Etape 1 : avec une porte PAND à deux entrées, deux séquences sont alors possibles (figure 4.13).

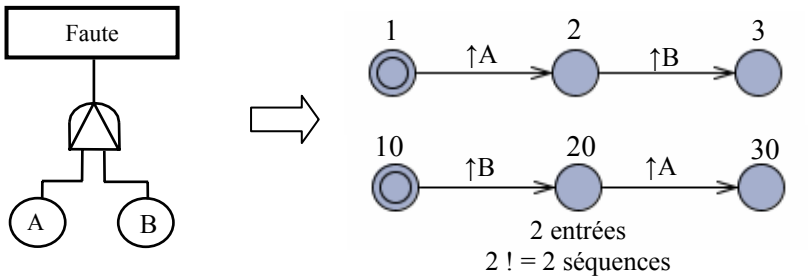


Figure 4.13. Séquences d'une PAND à deux entrées

Etape 2 : Les états 1 et 10, qui correspondent aux mêmes valeurs des entrées (A=B=0), et à la même valeur de la sortie (absence de faute), sont fusionnés. Par contre, les états *3* et *30* ne seront pas fusionnés, car même s'ils correspondent aux mêmes valeurs des entrées (A=B=1), la valeur de la sortie est différente (absence de faute dans l'état 30 et existence de faute dans l'état *3*) (figure 4.14.a). Donc, une seule séquence (dessinée en trait épais) mène à la faute.

Etape 3 : Les états 20 et 30 sont fusionnés (figure 4.14.b).

(a)

(b)

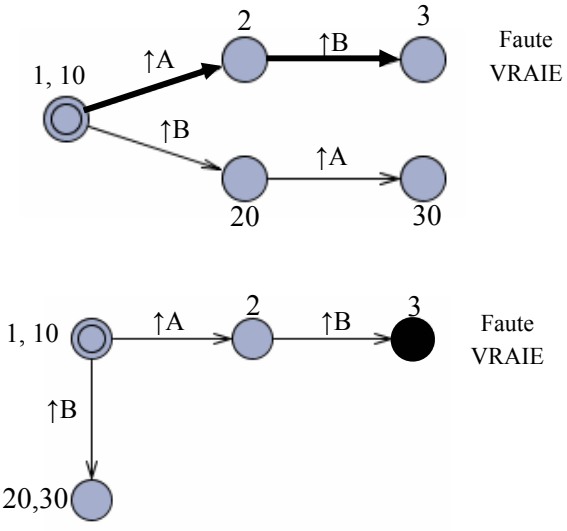


Figure 4.14. Construction du modèle formel de la porte PAND

Etape 4 : On rajoute une transition étiquetée par le front descendant de A (à l'issue de cette étape, ↑A peut apparaître plusieurs fois si l'un des états absorbants n'est pas atteint) et on obtient alors l'automate de la porte PAND à deux entrées (figure 4.15). L'état (*1,10*) est renommé *Initial* et l'état *3 Faute* (dessiné en noir). L'état 20,30 est renommé F*aute impossible*. **Nous remarquons qu'une fois atteinte la faute, elle est persistante pour toutes les évolutions.**

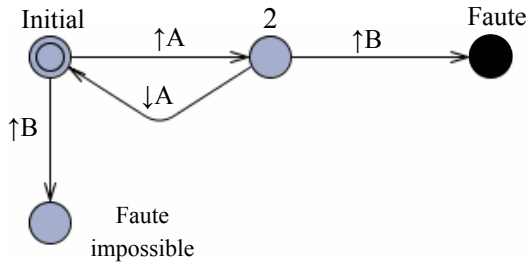


Figure 4.15. Automate associé à la porte PAND à deux entrées

Cas 2 : porte PAND à trois entrées

On applique maintenant la même méthode à une porte PAND à trois entrées (A, B, C). Ici, le nombre de séquences de base est de 3! = 6. A l'état *0* la valeur des variables est A=B=C=0. L'automate résultant des deux premières étapes est montré à la figure 4.16.a. La seule séquence qui mène à l'état où la faute est VRAIE (état *3*, en noir dans la figure), est dessinée en trait plus épais.

Etape 3 : On fusionne les états 10, 11, 12, 13, 20 et 21. Le résultat de la fusion est un état absorbant, qui représentera l'impossibilité de faute. On obtient le modèle de la figure 4.16.b.

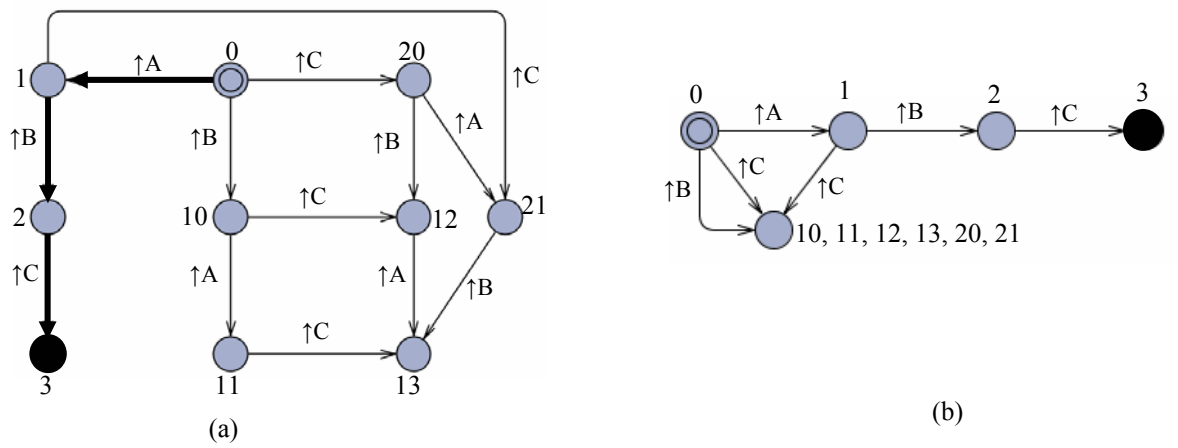


Figure 4.16. Automate à l'issue des étapes 1 et 2 (a) et de l'étape 3 (b)

Etape 4 : On rajoute les évolutions induites par les fronts descendants de A et B à partir des états *1* et *2* (ajout de l'état supplémentaire *4*). On obtient alors l'automate de la porte PAND à

trois entrées (figure 4.17). L'état *0* est nommé *Initial* et l'état *3*, comme *Faute* (dessiné en noir). L'état *20,30* est renommé *Faute impossible*.

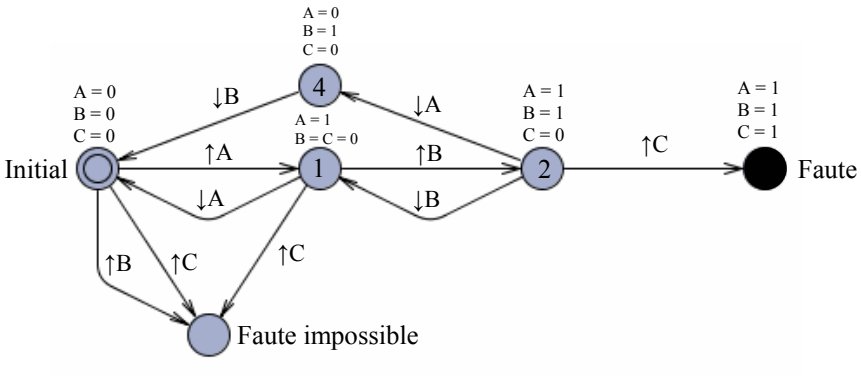


Figure 4.17. Automate associé à la porte PAND à trois entrées

Suivant la même procédure, on peut construire les automates associés à d'autres portes PAND. Nous allons maintenant analyser l'automate associé à la porte à trois entrées. Cet automate possède six états et dix transitions. Il est ainsi défini :

- L = {Initial, 1, 2, 4, Faute, Faute impossible}. Dans la figure 4.17 sont montrées les valeurs des variables à chaque état.
- A = {↑A, ↑B, ↑C, ↓A, ↓B}, est l'ensemble des actions déduites des variables logiques d'entrée de la porte. ***On rappelle que l'on suppose la non simultanéité d'occurrences d'événements.*** Cette hypothèse nous permet de garantir le déterminisme de l'automate, qui se perd, par exemple si à l'état 2, l'apparition simultanée de ↓B et ↑C est autorisée.
- l_0 = Initial.
- T = {(Initial,↑A,1), (1,↑B,2), (2,↑C,Faute), (1,↓A,Initial), (2,↓B,1), (2,↓A,4), (4,↓B,Initial), (Initial,↑B, Faute impossible), (Initial,↑C, Faute impossible), (1,↑C, Faute impossible)}.

La faute, c'est à dire la sortie de la porte, est représentée de **façon formelle par l'automate plus une condition sur l'activité de l'état *Faute***. Cet état est atteint si une occurrence de ↑A se produit suivie d'une occurrence de ↑B, elle-même suivie d'une occurrence de ↑C, et à condition que les variables A et B restent VRAIES après que les événements ↑A et ↑B se soient produits. Au contraire, si les valeurs de A et B changent (occurrence de ↓A ou de ↓B dans les états *1* ou *2*) avant l'occurrence de ↑C, alors on retourne soit à l'état *Initial*, en passant par l'état intermédiaire *4* (attente des prochaines occurrences de ↑A et de ↑B), soit à l'état *1* (attente de la prochaine occurrence de ↑B). Cette modélisation nous paraît être la plus pertinente mais il est aisé de proposer un modèle formel ne tenant pas compte de la non-persistance des variables, en supprimant les transitions étiquetées par les fronts descendants.

Il importe de souligner la différence entre cette hypothèse sur les changements possibles de la valeur d'une entrée d'une porte et celle retenue par l'AdD classique. Pour un composant physique en effet (cas de l'AdD classique), l'état *Faute* est souvent considéré comme

persistant. Par contre, cette situation change lorsqu'on s'intéresse à des fautes systématiques d'un contrôleur logique ; les entrées des portes sont alors des signaux logiques qui ne changent pas forcément une seule fois de valeur.

Il s'agit enfin d'établir la propriété formelle (troisième point de la méthode énoncée dans la section 2 du présent chapitre) à vérifier. Cette propriété doit établir que la faute ne se produit pas. Il faut donc vérifier que l'état *Faute* n'est jamais atteint, ce qui peut être exprimé par la formule CTL :

AG ¬(état Faute)

4.2 Porte POR avec condition de priorité sur une des entrées

Nous avons indiqué au chapitre 2, qu'une configuration générique à *n* entrées différentes de la porte POR est valable (Vesely et al. 2002). Pour exprimer formellement le comportement défaillant et la propriété à vérifier, nous proposons une solution utilisant une formule en logique temporelle CTL et une solution basée sur un automate observateur.

Comme pour la porte PAND, nous supposons :

- que toutes les variables d'entrée sont à l'état FAUX initialement ;
- la non simultanéité d'occurrences d'événements.

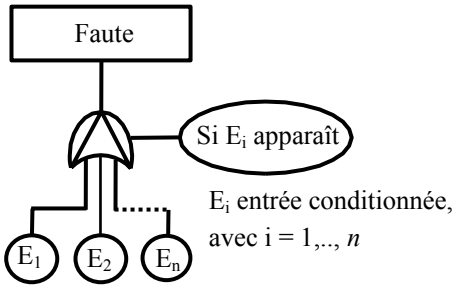


Figure 4.18. Porte POR à n entrées

4.2.1 Expression en logique temporelle CTL

La propriété à vérifier établit la façon dont **la faute ne se produit pas**. Cette propriété formelle s'énonce donc :

$$\mathbf{A}\ (\neg E_i\ \mathbf{W}\ (E_1 \vee E_2 \vee \ldots.\ E_{i-1} \vee E_{i+1} \vee \ldots.\vee E_n))$$

c'est-à-dire que pour tous les chemins, l'entrée conditionnée n'est pas VRAIE jusqu'à ce que la somme logique des autres entrées soit VRAIE ; cette somme devient en effet vraie dès qu'une des variables qui y figure devient vraie.

A titre d'exemple, la propriété formelle déduite d'une porte POR à deux entrées A et B (A est l'entrée conditionnée) s'énonce en CTL :

$$\mathbf{A}\ (\neg A\ \mathbf{W}\ B)$$

Une représentation graphique de cette expression est montrée en figure 4.19.

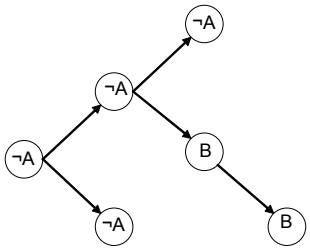


*Figure 4.19. Représentation graphique de l'expression **A** (¬A **W** B)*

4.2.2 Expression avec automate observateur

Nous traitons d'abord le cas général d'une porte POR à *n* entrées (figure 4.20.a), que nous instantions pour le cas particulier d'une porte à 2 entrées. Pour le cas général, on choisit E_i comme étant l'entrée conditionnée parmi les *n* entrées. Il suffit que l'entrée conditionnée se produise avant les autres entrées pour produire la faute. Il n'est pas obligatoire que les autres entrées se produisent après.

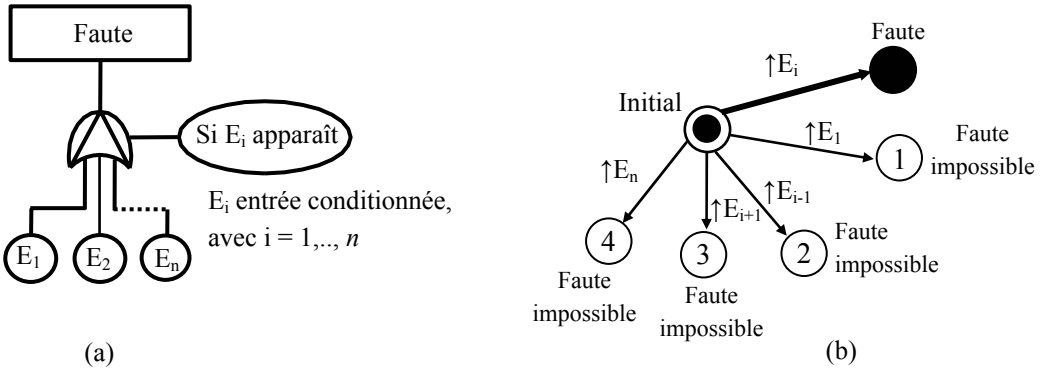


Figure 4.20. La porte POR à n entrées et son automate associé

Il est alors possible d'obtenir très facilement l'automate associé à la porte. Seule l'apparition de ↑ E_i à partir de l'état initial amène à la faute (figure 4.20.b). L'apparition des autres entrées mène aux états définis comme de *Faute impossible,* qui peuvent d'ailleurs être fusionnés.

L'automate pour le cas particulier d'une porte POR à deux entrées (variables) A et B (étant A la variable conditionnée), est montré en figure 4.21.

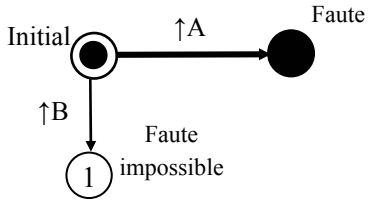


Figure 4.21. Automate associé à la porte POR à deux entrées avec A l'entrée conditionnée

Cet automate détecte alors le front montant de A, ce qui permet de franchir la transition qui mène à l'état *Faute*. Donc, la sortie de la porte est représentée de façon formelle par l'automate observateur plus la condition d'activation de l'état *Faute*. L'apparition de ↑B mène à l'état *Faute impossible*.

Maintenant, il s'agit d'établir la propriété formelle à vérifier. La propriété déduite doit alors établir que la faute ne se produit pas, c.a.d. qu'elle ne doit jamais être VRAIE. Il faut donc vérifier que l'état *faute*, n'est jamais atteint, ce qui peut être exprimée avec la formule en CTL :

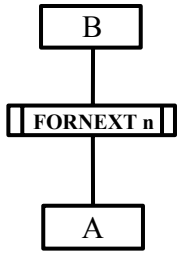
AG ¬(état Faute)

5 Formalisation du comportement des portes temporisées retenues

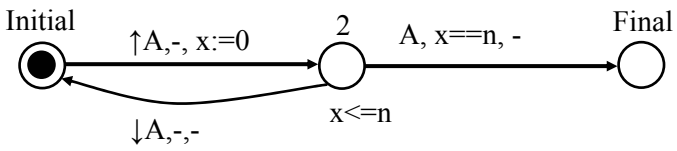
Muni du formalisme des automates temporisés, il est possible de proposer un modèle formel du comportement des portes temporisées FORNEXT n et WITHIN n (Barragan et al. 2006). Les deux cas sont présentés ci-après. Pour chacun d'eux, le symbole de la porte et l'automate associé sont présentés ; on suppose dans ces modèles que l'entrée de chaque porte est à l'état FAUX initialement.

5.1 La porte FORNEXT n

Dans l'automate de la figure 4.22, le front montant de A permet de franchir la première transition où l'horloge x est initialisée et d'arriver dans l'état 2 où la valeur de l'horloge doit être inférieure ou égale à *n* (voir l'invariant). Depuis cet état, deux transitions sont possibles : l'une des deux renvoie à l'état *Initial* si la variable A n'est pas persistante et redevient FAUSSE avant que l'horloge atteigne la valeur *n*. La seconde transition mène à l'état *Final* si x est égale à *n* et A est toujours VRAI.



B est vraie si A est vraie pendant n unités de temps après son changement d'état (front montant)

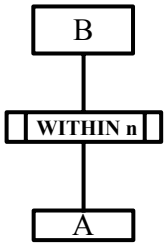


A est restée vraie pendant n unités de temps depuis l'instant de franchissement de la transition issue de état *Initial*, si l'état *Final* est atteint

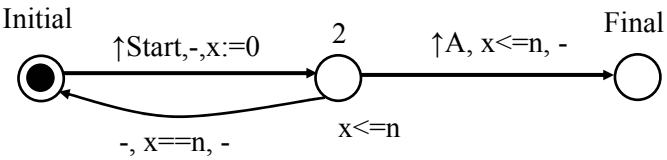
Figure 4.22. Automate associé à la porte FORNEXT n

5.2 La porte WITHIN n

L'automate associé à cette porte (figure 4.23) décrit le comportement suivant. Dans la première transition, un événement appelé « ↑Start » détermine le début de l'intervalle de temps compris entre x = 0 et x = n. L'événement « ↑Start » n'étant pas représenté dans le symbole de la porte est introduit ici pour permettre la modélisation. L'état *Final* est atteint si le front montant ↑A se produit dans cet intervalle. Dans le cas contraire, si ↑A n'apparaît pas dans l'intervalle, l'automate revient à l'état *Initial*.



B est vraie si A est vraie au moins une fois à l'intérieur d'un intervalle de n unités de temps après l'instant courant



Une occurrence de ↑A se produit dans un intervalle de n unités de temps après ↑Start si l'état Final est atteint

Figure 4.23. Automate associé à la porte WITHIN n

Ces figures montrent que les automates ne comportent pas d'état caractéristique d'une faute, ce qui est normal car une porte temporisée isolée ne traduit que le comportement temporisé d'une variable logique. En conséquence, il ne sera pas proposé d'expression formelle de faute, ni de propriété à vérifier.

6 Conclusion

Nous avons proposé dans ce chapitre quatre une représentation formelle du comportement décrit par chacune des portes pouvant intervenir dans un sous-arbre exprimant une faute systématique : portes statiques, temporelles et temporisées. Cette modélisation formelle des comportements a permis d'obtenir des modèles formels de fautes et de propriétés à vérifier. Le résultat du chapitre est donc une bibliothèque de comportements et de propriétés formelles qui constitue notre troisième contribution.

Il nous reste maintenant à exposer notre quatrième contribution, à savoir une méthode de contrôle de la cohérence d'une association de portes et d'obtention du modèle formel équivalent à cette association. Cela est l'objet du chapitre suivant.

Chapitre 5.

Obtention de propriétés formelles à partir d'associations de portes

Au chapitre précédent, nous avons formulé l'objectif de passer de l'AdD dynamique qui représente la faute systématique aux propriétés formelles du contrôleur logique (figure 5.1). Pour y aboutir, nous avons proposé de faire :

- Une représentation formelle du comportement défaillant décrit par **chacune des portes** pouvant intervenir dans un sous-arbre exprimant une faute systématique : portes statiques, temporelles et temporisées. Cette modélisation formelle de la faute permet d'obtenir la propriété formelle à vérifier.
- Une technique d'analyse de la cohérence **d'une association de portes** statiques, temporelles et temporisées.
- Une **représentation formelle d'une association** cohérente de portes permettant d'obtenir une propriété formelle exprimant que le comportement défaillant décrit par cette association ne se produira pas.

Le premier point a été abordé au chapitre quatre. Tout au long de ce cinquième chapitre, nous allons développer les deux points restants.

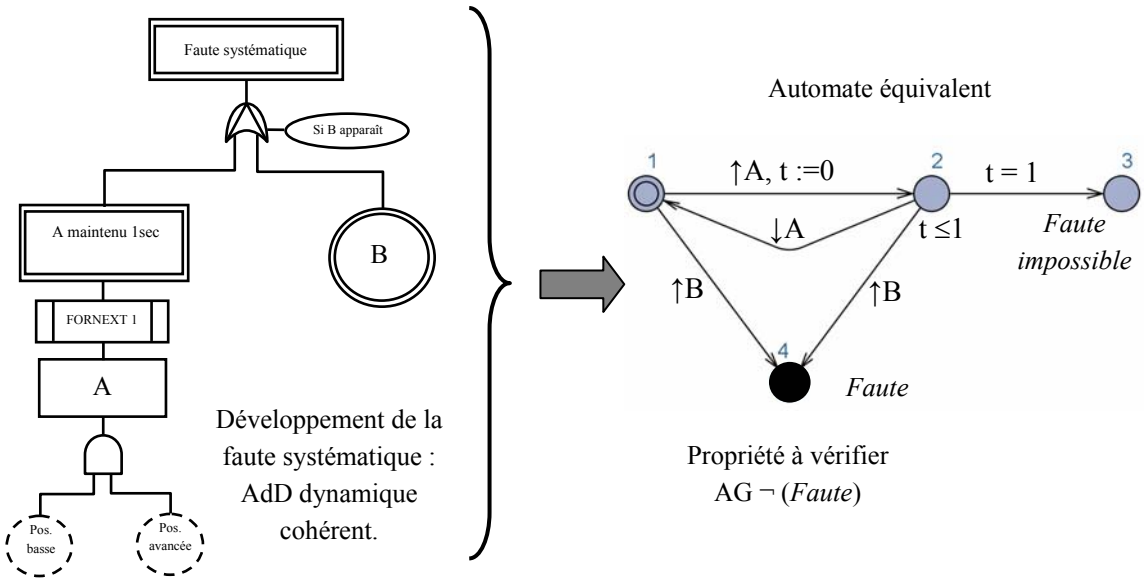


Figure 5.1. Passage de l'AdD dynamique cohérent en automate équivalent et obtention des propriétés

L'organisation de ce chapitre est la suivante. Nous présentons dans la première section la liste des associations des portes que nous avons étudiées, puis nous expliquons comment contrôler la cohérence des entrées de chaque porte d'une association. Le chapitre se poursuit par l'obtention de la représentation formelle des associations de portes listées et des propriétés formelles déduites de ces associations.

1 Associations étudiées

On appelle Arbre des Défaillances dynamique, un AdD qui comporte des portes statiques, temporelles et temporisées. Cet arbre est un outil capable de représenter les fautes de conception des contrôleurs logiques. Un arbre dynamique est un tuple (Ps, Pt, Pti, V) :

- **Ps** est un ensemble des **portes statiques** ;
- **Pt** est un ensemble des **portes temporelles** ;
- **Pti** est un ensemble des **portes temporisées** ;
- **V** est un ensemble de **variables logiques** (entrées et sorties du contrôleur).

Etant donné que les entrées et les sorties de toutes les portes sont des signaux logiques, toutes les connexions entre les portes sont donc possibles. Cependant nous traiterons d'une part les associations ne comportant que de portes statiques et temporelles, qui ne prennent en compte que le temps logique, et d'autre part les associations faisant intervenir des portes temporisées, qui intègrent une modélisation du temps physique nécessitant l'introduction d'horloges.

D'autre part, nous limiterons notre analyse à des associations particulières correspondant à des arbres à **deux niveaux**, c'est-à-dire des arbres dont le sommet est fonction d'événements

intermédiaires qui eux-mêmes ne dépendent que des feuilles de l'arbre, et non d'autres événements intermédiaires.

Plus précisément, les associations-types étudiées seront classées en trois catégories :

- **Associations dont le sommet est une porte statique**
- **Associations dont le sommet est une porte temporelle**
- **Associations faisant intervenir des portes temporisées**

qui sont détaillées ci-dessous.

Associations dont le sommet est une porte statique

1. Association dont le sommet est une porte ET dont les entrées sont les sorties de portes PAND.
2. Association dont le sommet est une porte OU dont les entrées sont les sorties de portes PAND.
3. Association dont le sommet est une porte ET dont les entrées sont les sorties de portes POR.
4. Association dont le sommet est une porte OU dont les entrées sont les sorties de portes POR.
5. Association dont le sommet est une porte ET dont les entrées sont des sorties de portes PAND et des sorties de portes POR.
6. Association dont le sommet est une porte OU dont les entrées sont des sorties de portes PAND et des sorties de portes POR.
7. Association dont le sommet est une porte ET dont les entrées sont des sorties de portes PAND et des variables.
8. Association dont le sommet est une porte OU dont les entrées sont des sorties de portes PAND et des variables.
9. Association dont le sommet est une porte ET dont les entrées sont des sorties de portes POR et des variables.
10. Association dont le sommet est une porte OU dont les entrées sont des sorties de portes POR et des variables.

Associations dont le sommet est une porte temporelle

11. Association dont le sommet est une porte PAND dont les entrées sont des sorties de portes PAND.
12. Association dont le sommet est une porte POR dont les entrées sont des sorties de portes PAND.
13. Association dont le sommet est une porte PAND dont les entrées sont des sorties de portes POR.
14. Association dont le sommet est une porte POR dont les entrées sont des sorties de portes POR.

Associations faisant intervenir des portes temporisées

15. Modélisation de l'association dont le sommet est une porte ET dont les entrées sont les sorties de portes FORNEXT n et WITHIN n.
16. Modélisation de l'association dont le sommet est une porte POR dont les entrées sont la sortie d'une porte FORNEXT n et une variable.

Pour chacun des cas retenus, nous effectuerons le contrôle de la cohérence de l'association, si cela est nécessaire, puis déterminerons le modèle formel de cette association et en déduirons la propriété à vérifier.

Il est important de souligner que, pour obtenir la représentation formelle des associations ne comportant que des portes statiques et temporelles, nous nous baserons sur la **composition d'automates**. En revanche, les associations faisant intervenir les portes temporisées, seront traitées au cas par cas.

2 Contrôle de la cohérence de chaque porte temporelle

Les portes statiques n'imposent pas d'ordre des occurrences au niveau des entrées et ne peuvent donc pas donner lieu à des incohérences lors de la conception de l'arbre. Par contre, les portes temporelles peuvent présenter des incohérences qui sont analysées ci-dessous. Quant aux portes temporisées, chacune d'elle ne comporte qu'une entrée et ne peut donc être incohérente prise isolément.

Pour une porte PAND

Soit $E = (E_1, E_2,\ldots, E_{n-1}, E_n)$ l'ensemble des entrées d'une porte PAND. Il est impossible d'appliquer la même variable logique V_j sur deux entrées distinctes d'une porte PAND, c'est à dire qu'il n'est pas possible qu'un événement se produise avant lui-même.

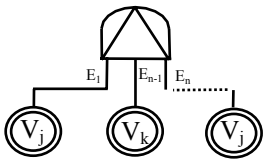


Figure 5.2. Exemple d'incohérence pour une porte PAND

Pour une porte POR

Soit $E = (E_1, E_2,\ldots, E_{n-1}, E_n)$ l'ensemble des entrées d'une porte POR. Il est impossible d'appliquer la même variable logique V_j sur deux entrées distinctes d'une porte POR et d'avoir V_j comme l'entrée conditionnée, car il n'est pas possible qu'un événement se produise avant lui-même.

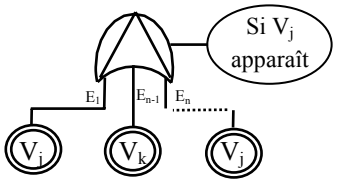


Figure 5.3. Exemple d'incohérence pour une porte POR

3 Composition d'automates

Nous avons montré au chapitre quatre que le comportement de chaque porte temporelle peut être modélisé formellement à l'aide d'un automate à états plus une condition sur l'activité d'un état de cet automate où la défaillance est vraie. Si une seule porte est ainsi représentée, la représentation formelle du comportement d'une association ne faisant intervenir que des portes statiques ou temporelles peut être obtenue par composition des automates associés à chacune des portes de cette association, plus une condition sur certains états de l'automate obtenu par composition. La composition d'automates est une méthode qui est formellement définie et qui permet de traiter le cas général de *m* automates à *n* états.

Dans ce qui suit, nous utiliserons les définitions proposées par (Julliand, 2003) qui propose plusieurs méthodes de composition (ou produit) d'automates listées dans ce qui suit :

- Composition libre ou parallèle, ou produit cartésien : composition libre des composants,
- Composition synchrone : composition des composants évoluant simultanément,
- Produit asynchrone : composition des composants évoluant indépendamment.

3.1 La composition parallèle

Un automate G_i est un 4-uplet $<L_i,l_{0i},A_i,R_i>$, où :

- L_i est un ensemble fini d'états,
- l_{0i} est l'état initial de l'automate,
- A_i est un ensemble fini d'actions,
- $R_i \subseteq L \times A \times L$ est l'ensemble des transitions.

Le produit libre ou composition parallèle de deux automates $G_1 = <L_1, A_1, l_{01}, R_1>$ et $G_2 = <L_2, A_2, l_{02}, R_2>$ notée $G_1||G_2$ est un automate $G_c = <L_1xL_2, A_1 \vee A_2 \vee A_1xA_2, (l_{01}, l_{02}), R>$ où :

— R est l'ensemble de transitions défini par $(l_1, l_2) \xrightarrow{A} (l'_1, l'_2)$ ssi :

- soit $l_1 \xrightarrow{A_1} l'_1 \in R_1$, alors $l'_2 = l_2$ alors $(l_1, l_2) \xrightarrow{A} (l'_1, l_2) \in R$
- soit $l_2 \xrightarrow{A_2} l'_2 \in R_2$, alors $l'_1 = l_1$ alors $(l_1, l_2) \xrightarrow{A} (l_1, l'_2) \in R$
- soit $A=(A_1,A_2)$ et $l_1 \xrightarrow{A_1} l'_1 \in R_1$ et $l_2 \xrightarrow{A_2} l'_2 \in R_2$
  alors $(l_1, l_2) \xrightarrow{A_1,A_2} (l'_1, l'_2) \in R$

Exemple : soit deux automates à composer par cette méthode

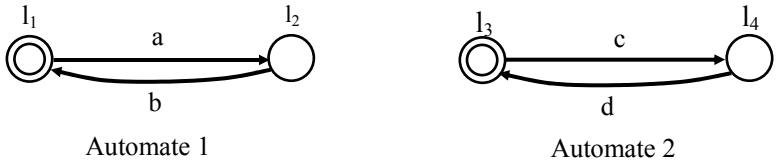


Figure 5.4. Automates à composer

Avec les notations précédentes les deux automates sont définis formellement de la façon suivante :

a) Automate 1

$L_1 = \{l_1,l_2\}$, $l_{01} = l_1$, $A_1 = \{a, b\}$

$R_1 = \{(l_1, a, l_2), (l_2, b, l_1)\}$

b) Automate 2

$L_2 = \{l_3, l_4\}$, $l_{02} = 1_3$, $A_2 = \{c, d\}$

$R_2 = \{(l_3, c, l_4), (l_4, d, l_3)\}$

Le produit libre des automates 1 et 2 est :

- $L = L_1 x\ L_2 = \{(l_1,1_3), (l_1,1_4), (1_2,1_3), (l_2,1_4)\}$
- $l_0 = (l_{01}, l_{02}) = (1_1,1_3)$
- $A = A_1 \vee A_2 \vee A_1 x A_2 = \{a, b, c, d, ac, bc, ad, bd\}$
- $R = \{(l_1,1_3)$ a (l_2,l_3) ; (l_1,l_3) c $(l_1,1_4)$; $(l_1,1_3)$ ac $(l_2,1_4)$
  $(l_2,1_3)$ b $(l_1,1_3)$; $(l_2,1_3)$ c (l_2,l_4) ; $(l_2,1_3)$ bc $(l_1,1_4)$
  $(l_1,1_4)$ a (l_2, l_4) ; (l_1, l_4) d (l_1, l_3) ; (l_1, l_4) ad (l_2, l_3)
  (l_2, l_4) b (l_1, l_4) ; (l_2, l_4) d (l_2, l_3) ; (l_2, l_4) bd $(l_1, l_3)\}$

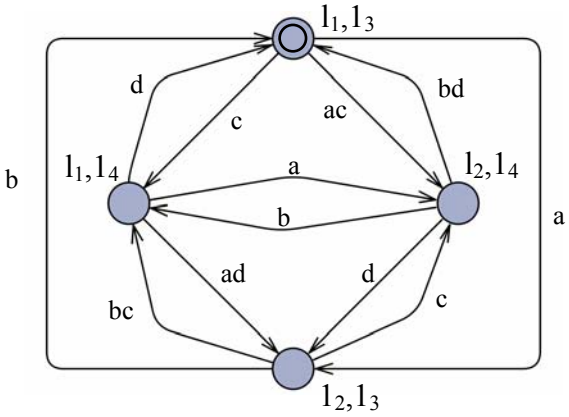


Figure 5.5. Automate résultant du produit libre des automates 1 et 2 de la figure 5.4

3.2 La composition synchrone

Le produit synchrone de deux automates $G_1 = <L_1, A_1, l_{01}, R_1>$ et $G_2 = <L_2, A_2, l_{02}, R_2>$ noté $G_1||G_2\backslash A_1xA_2$ est un automate $G_c = <L_1xL_2, A_1xA_2, (l_{01}, l_{02}), R'>$ où :

— R' est la restriction de R (du produit libre) aux transitions étiquetées par un élément de A_1xA_2 : { ac, bc, ad, bd} dans le cas étudié

La figure 5.6 montre le produit synchrone des automates de la figure 5.4.

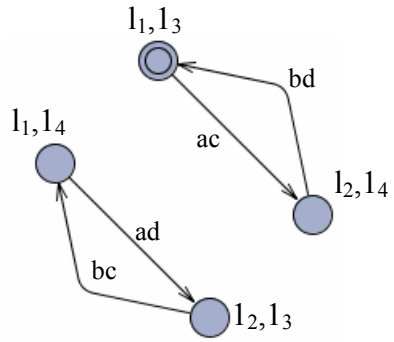


Figure 5.6. Produit synchrone des automates de la figure 5.4

3.3 La composition asynchrone

Le produit asynchrone de deux automates $G_1 = <L_1, A_1, l_{01}, R_1>$ et $G_2 = <L_2, A_2, l_{02}, R_2>$ noté $G_1||G_2\backslash A_1 \vee A_2$ est un automate $G_c = <L_1xL_2, A_1 \vee A_2, (l_{01}, l_{02}), R'>$ où :

— R' est la restriction de R (produit libre) aux transitions étiquetées par un élément de $A_1 \vee A_2$: {a, b, c, d} dans le cas étudié

La figure 5.7 montre le produit asynchrone des automates de la figure 5.4.

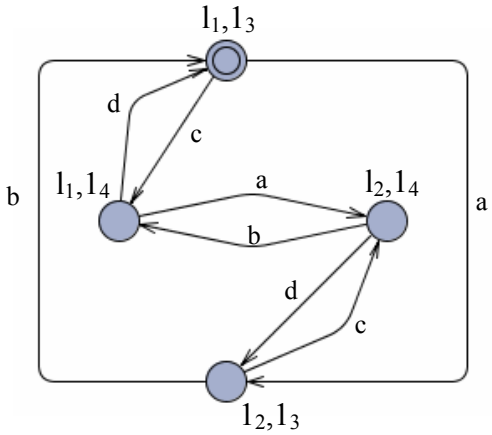


Figure 5.7. Produit asynchrone des automates de la figure 5.4

3.4 Utilisation des techniques de composition d'automates dans ce mémoire

Pour obtenir le modèle formel du comportement des associations de portes étudiées, deux cas sont à distinguer selon que les portes temporelles situées à la base de l'association (portes dont les entrées sont les feuilles de l'arbre) possèdent des variables communes ou non. Nous définirons donc tout d'abord ce concept.

Variable commune à deux portes temporelles

Soit deux portes temporelles P_{t1} et P_{t2} possédant respectivement *m* et *n* entrées : (E_1, E_2,…, E_{m-1}, E_m) et (E'_1, E'_2,…, E'_{n-1}, E'_n). On dira que ces portes possèdent une variable commune V_j si : $\exists(E_i, E_j')$ avec $i\in[1,m]$ et $j\in[1,n]$, tel que le même signal logique V_j est appliqué à E_i et à E_j. La figure 5.8 illustre ce concept, V_b étant la variable commune.

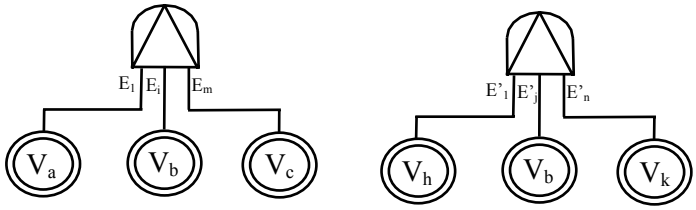


Figure 5.8. Variable commune à deux portes temporelles

Si les portes temporelles situées à la base de l'association n'ont pas de variable commune, nous **adopterons** pour notre travail la **composition asynchrone**, car nous considérons la non simultanéité des événements d'entrée des portes. Si ce n'est pas le cas, il faudra veiller à synchroniser les franchissements des transitions qui portent la même action, c'est-à-dire que :

- si $\exists\, a \in$ A1 et $a \in$ A2, avec $l_1 \xrightarrow{a} l'_1$ et $l_2 \xrightarrow{a} l'_2$ alors $(l_1, l_2) \xrightarrow{a} (l'_1, l'_2)$.

Enfin il convient de remarquer que l'automate obtenu par composition devra nous permettre de déterminer les **séquences minimales** conduisant à la faute. Dans un AdD contenant des portes temporelles, le concept de coupe minimale utilisé en analyse de défaillances conventionnelle n'a plus aucun sens, car ce qui nous importe ce n'est pas seulement l'état des variables mais l'ordre d'occurrence des événements. Ce concept doit être remplacé par celui de **séquence minimale,** plus petite séquence d'événements conduisant à la faute.

4 Associations de portes dont le sommet est une porte statique

Chacune des associations étudiées dans cette section comprend au sommet une porte statique (ET ou OU) dont les entrées sont les sorties de portes temporelles ou des variables. Pour obtenir le modèle formel du comportement décrit par une association, on procède de la façon suivante :

I. On effectue la composition des automates décrivant le comportement des portes temporelles comme indiqué ci-dessus.
II. Dans le cas où la porte située au sommet de l'arbre est un ET, il y a un seul état absorbant caractéristique de la faute située au sommet et plusieurs états absorbants indiquant que cette faute ne peut pas se produire ; ces derniers états peuvent alors être fusionnés. Dans le cas où la porte située au sommet de l'arbre est un OU, il y a un ou plusieurs états absorbants caractéristiques de la faute sommet ; ils peuvent être fusionnés par souci de simplification.

Cette méthode est appliquée ci-après sur les différents cas retenus après que le contrôle de cohérence de l'association, si nécessaire, ait été effectué.

4.1 Association dont le sommet est une porte ET dont les entrées sont les sorties de portes PAND

4.1.1 Contrôle de cohérence

La figure 5.9 illustre le cas général de ce type d'association, c'est-à-dire *m* portes PAND à plusieurs entrées.

Premier cas d'incohérence

Soit V_{com} l'ensemble des variables communes aux portes, tel que $V_{com} \subseteq V$. Pour tout couple de variables (V_j,V_k), avec V_j et $V_k \in V_{com}$, si, pour une des portes V_j *avant* V_k et en même temps pour une autre porte V_k *avant* V_j, alors il y a une *incohérence*.

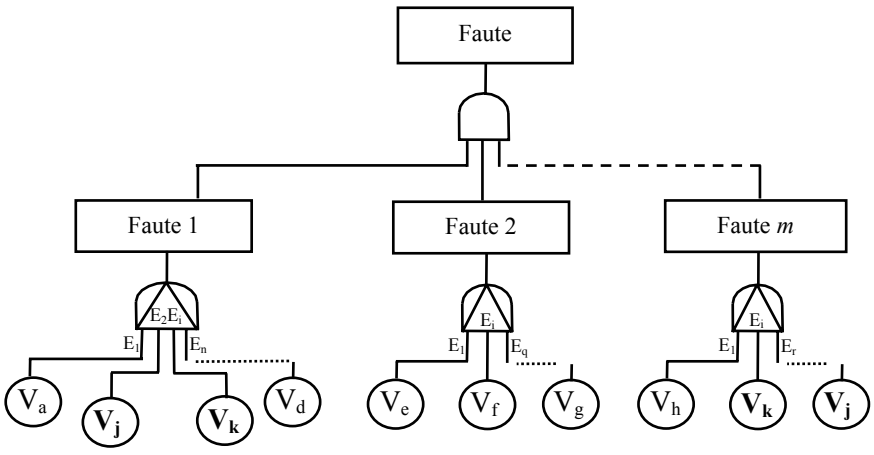


Figure 5.9. Exemple d'incohérence pour l'association dont le sommet est une porte ET dont les entrées sont les sorties de portes PAND

Deuxième cas d'incohérence : la redondance cyclique

Pour la redondance cyclique (Walker et Papadoupolus, 2006), trois portes PAND sont considérées au minimum. Dans la figure 5.10 par exemple, $(V_j,V_k,V_l) \in V$ et on souhaite avoir V_j *avant* V_k, V_k *avant* V_l, et V_l *avant* V_j.

La combinaison présente une incohérence car chaque événement doit se produire avant chacun des autres.

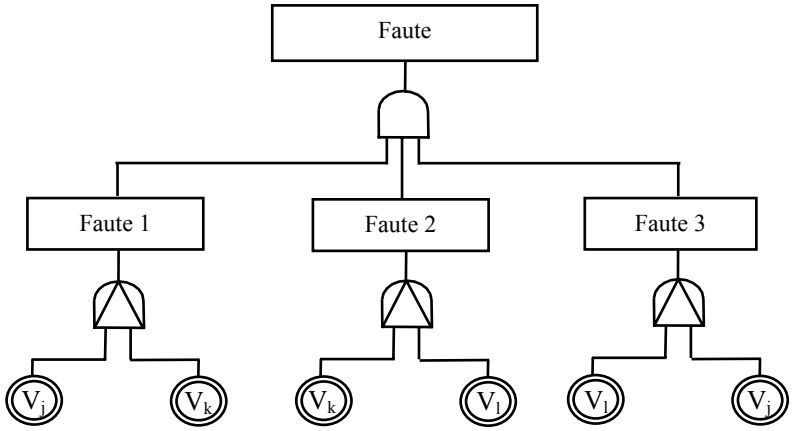


Figure 5.10. La redondance cyclique

4.1.2 Modélisation de l'association

La composition d'automates permet alors de traiter le cas général (figure 5.9) et d'obtenir l'automate modélisant l'association dans son ensemble. Bien sûr, la taille de cet automate dépendra du nombre de portes PAND et du nombre d'états de chaque automate associé à chaque PAND. Par souci de simplicité, nous nous limitons ici, et pour toutes les catégories d'associations qui suivent, au cas particulier de deux portes à deux entrées.

Ainsi, la figure 5.11 montre deux portes PAND à deux entrées, dont les sorties sont associées à une porte ET. L'automate observateur modélisant chaque porte temporelle est aussi montré dans la figure.

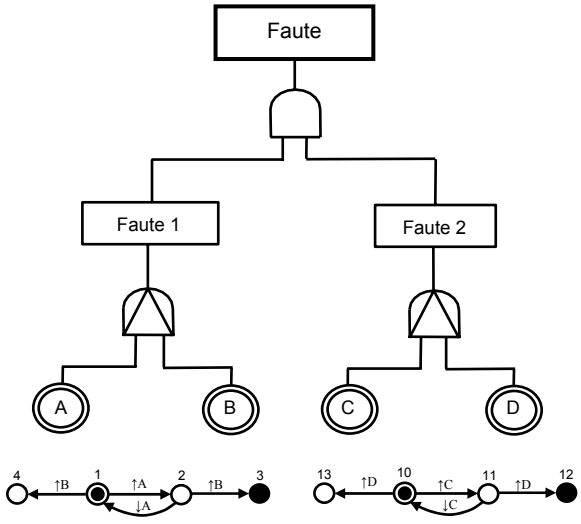


Figure 5.11. Association de deux portes PAND par une porte ET

On effectue alors la composition asynchrone des deux automates, selon la définition énoncée dans la section 3. On a alors :

a) Automate associé à la porte PAND de gauche:
$L_1 = \{1,2,3,4\}$, $l_{01} = 1$, $A_1 = \{\uparrow A, \uparrow B, \downarrow A\}$
$R_1 = \{(1, \uparrow A, 2), (2, \uparrow B, 3), (2, \downarrow A, 1), (1, \uparrow B, 4)\}$

b) Automate associé à la porte PAND de droite:
$L_2 = \{10,11,12,13\}$, $l_{02} = 10$, $A_2 = \{\uparrow C, \uparrow D, \downarrow C\}$
$R_2 = \{(10, \uparrow C, 11), (11, \uparrow D, 12), (11, \downarrow C, 10), (10, \uparrow D, 13)\}$

La figure 5.12 montre l'automate obtenu. Il est composé de 16 états, dont (1,10) est l'état initial.

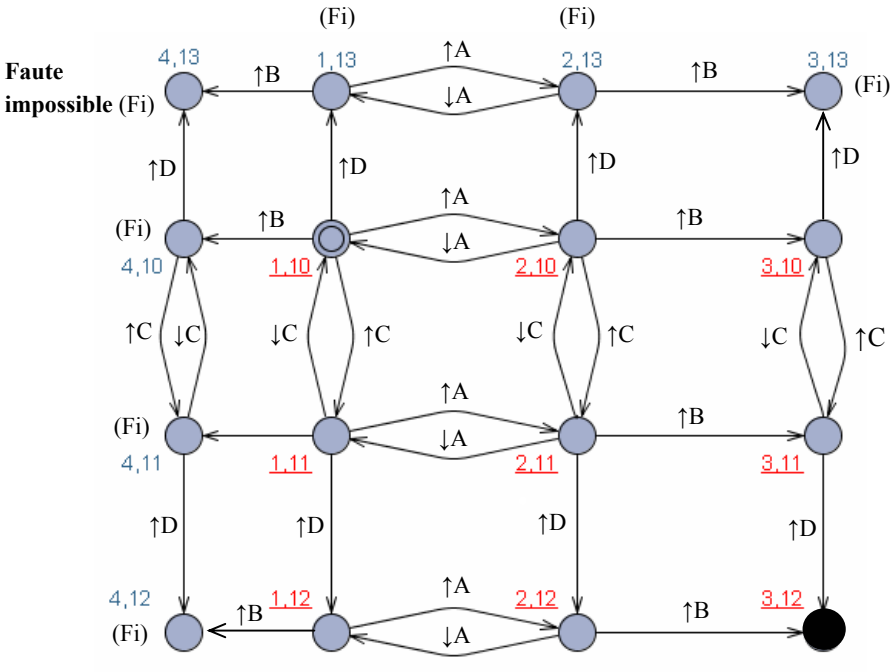


Figure 5.12. Automate déduit de l'association de la figure 5.11

Cet automate peut être réduit, car plusieurs transitions amènent à des états caractérisant l'absence de faute (Fi) ; ces états peuvent être fusionnés.

La figure 5.13 montre l'automate ainsi réduit. Les éléments de cet automate sont listés ci-après:

L = {(1,10), (1,11), (1,12), (2,10), (2,11), (2,12), (3,10), (3,11), (3,12), (Faute impossible)} ;
l_0 = (1,10) ;
A = {↑A, ↑B, ↓A, ↑C, ↑D, ↓C} ;

R' = {(1,10) ↑A (2,10) (1,10) ↑C (1,11) (1,11) ↑A (2,11) (1,11) ↑D (1,12)
(1,11) ↓C (1,10) (1,12) ↑A (2,12) (2,10) ↑B (3,10) (2,10) ↓A (1,10)
(2,10) ↑C (2,11) (2,11) ↑B (3,11) (2,11) ↓A (1,11) (2,11) ↑D (2,12)
(2,11) ↓C (2,10) (2,12) ↑B (3,12) (2,12) ↓A (1,12) (3,10) ↑C (3 ,11)
(3,11) ↑D (3,12) (3,11) ↓C (3,10) (1,10 ; 2,10 ; 3,10) ↑D (Faute impossible)
(1,10 ; 1,11 ; 1,12) ↑B (Faute impossible)}.

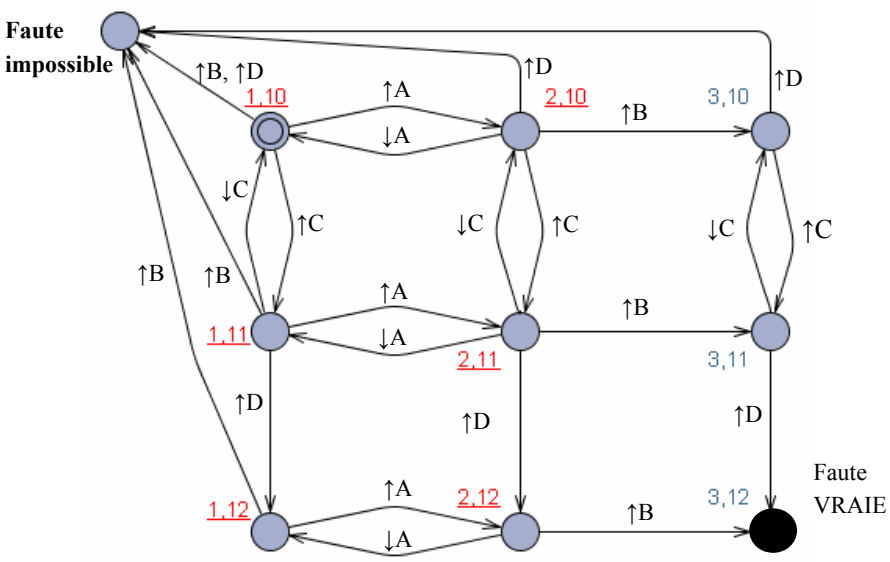


Figure 5.13. Automate réduit de l'association de la figure 5.11

L'état (*3,12*), représenté en noir dans la figure ci-dessus, est le seul état où la faute est VRAIE, ce qui correspond à la définition de la porte ET qui oblige à avoir comme vraies les deux fautes 1 et 2. C'est effectivement quand on arrive à l'état (*3,12*) que cela est vrai prenant en compte l'ordre établi par les portes PAND. Il existe six séquences minimales qui mènent de l'état initial à l'état de faute: ↑A↑B↑C↑D, ↑A↑C↑B↑D, ↑A↑C↑D↑B, ↑C↑A↑B↑D, ↑C↑A↑D↑B, ↑C↑D↑A↑B.

Dans ce cas là, comme dans tous les cas traités dans ce chapitre, **une fois obtenu l'automate formalisant le comportement de l'association, la propriété formelle à vérifier porte sur la non atteignabilité de(s) état(s) de faute.** Ici, la propriété à vérifier établit alors que l'état (*3,12*) ne doit jamais être atteint. La formule **AG** ¬ (*3,12*), exprime que par tous les chemins, dans tous les états, la faute ne se produit pas.

4.1.3 Cas particulier : variable commune aux entrées

La figure ci-contre montre un cas particulier de cette association, où la variable d'entrée B est commune aux deux portes PAND.

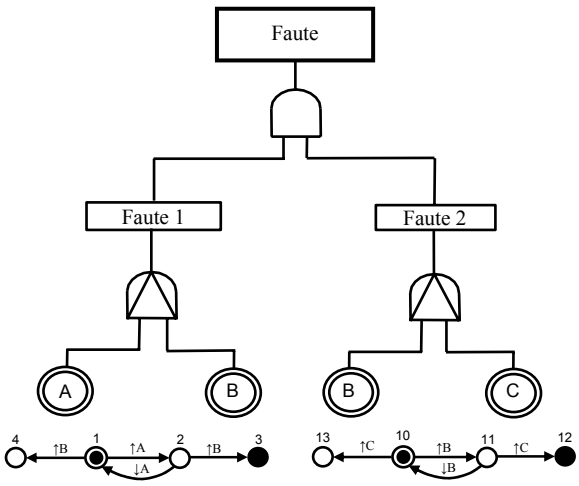


Figure 5.14. Association dont le sommet est une porte ET dont les entrées sont les sorties de portes PAND, avec la variable B commune aux portes PAND

Le modèle formel du comportement de l'association est obtenu comme indiqué en 3.4, en tenant compte des simultanéités de franchissement possibles. Ce modèle est représenté en figure 5.15.

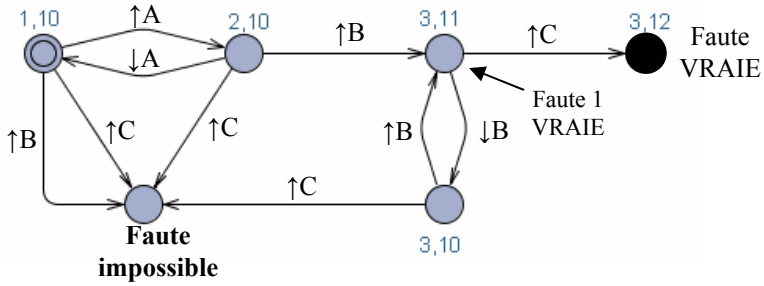
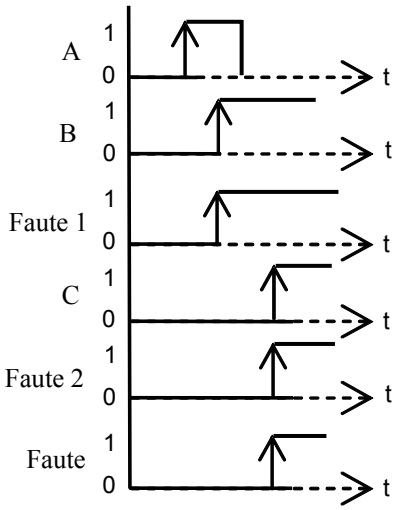


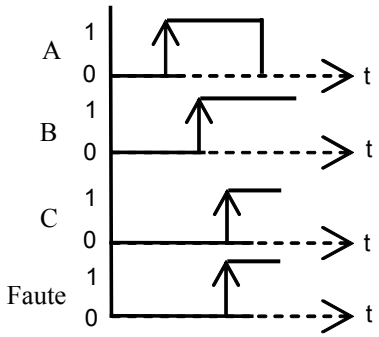
Figure 5.15. Automate déduit de l'association de la figure 5.14

Il convient de remarquer que ce modèle diffère de celui de la porte PAND à trois entrées présenté au chapitre précédent. Ceci s'explique facilement. La porte PAND à trois entrées (que nous noterons A, B et C, l'ordre des événements conduisant à la faute étant l'ordre alphabétique) impose que les deux entrées A et B soient à l'état VRAI lorsque se produit l'occurrence du front montant de C qui provoque la faute. Ce comportement n'est pas celui de l'association étudiée. Dans cette association en effet, la faute Faute est vraie si Faute1 et Faute2 sont toutes deux vraies. Si la séquence ↑A puis ↑B se produit, C restant à l'état FAUX, Faute1 devient vraie et **reste vraie** car, une fois atteinte, la faute est persistante pour toutes les évolutions (remarque faite au chapitre 4, sous-section 4.1.2 suite à l'élaboration du modèle formel du comportement d'une PAND à deux entrées). Si, par la suite, A passe à l'état FAUX, puis C devient VRAI, B n'ayant pas changé de valeur (séquence ↓A puis ↑C), la faute Faute2 devient vraie ainsi que la faute Faute. La séquence ↑A, ↑B, ↓A, ↑C conduit donc à la faute pour cette association (figure 5.16.a), alors que ce n'est pas le cas pour une porte PAND avec la définition que nous avons choisie et qui consiste à considérer qu'une faute résulte

d'une séquence d'événements avec maintient des valeurs des variables après ces événements (figure 5.16.b).



(a) Chronogramme du comportement de l'association de la figure 5.14



(b) Chronogramme du comportement de la porte PAND à trois entrées A,B,C

Figure 5.16. Comparaison entre (a) le comportement de l'association de la figure 5.14 et (b) celui d'une porte PAND à trois entrées

4.2 Association dont le sommet est une porte OU dont les entrées sont les sorties de portes PAND

Pour cette association (figure 5.17), nous n'effectuerons pas de contrôle de cohérence, car la porte OU n'impose pas l'apparition stricte des deux fautes 1 et 2 pour produire la faute. Il n'y a pas de contrainte sur les entrées des portes temporelles, même si une association du type (A avant B) OU (B avant A) peut paraître surprenante.

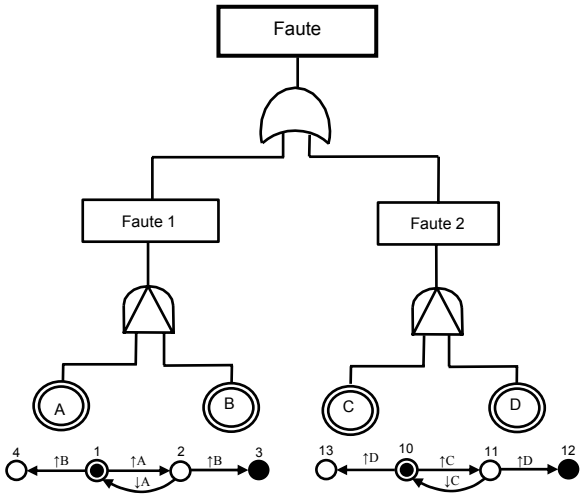


Figure 5.17. Association de deux portes PAND par une porte OU

Le fait d'utiliser la porte OU comme élément d'union conduit à un automate à plusieurs états de faute (figure 5.18) car il suffit que chaque automate élémentaire atteigne son état défaillant et cela indépendamment de l'évolution de l'autre automate (loi de la porte OU).

Cet automate comprend 7 états de faute (*1,12* ; *2,12* ; *3,12* ; *3,10* ;*3,11* ;*3,13* ;*4,12*) et un seul état où la faute ne peut se produire, car il faut avoir les fautes 1 et 2 fausses pour que la défaillance ne se produise pas. Les états de faute peuvent être fusionnés.

Dix séquences minimales sont possibles : ↑A↑B, ↑A↑C↑B, ↑A↑D↑B, ↑A↑C↑D, ↑B↑C↑D, ↑C↑A↑B, ↑C↑A↑D, ↑C↑B↑D, ↑C↑D, ↑D↑A↑B. La propriété formelle à vérifier porte sur la non atteignabilité des états défaillants, **AG** ¬ (états défaillants).

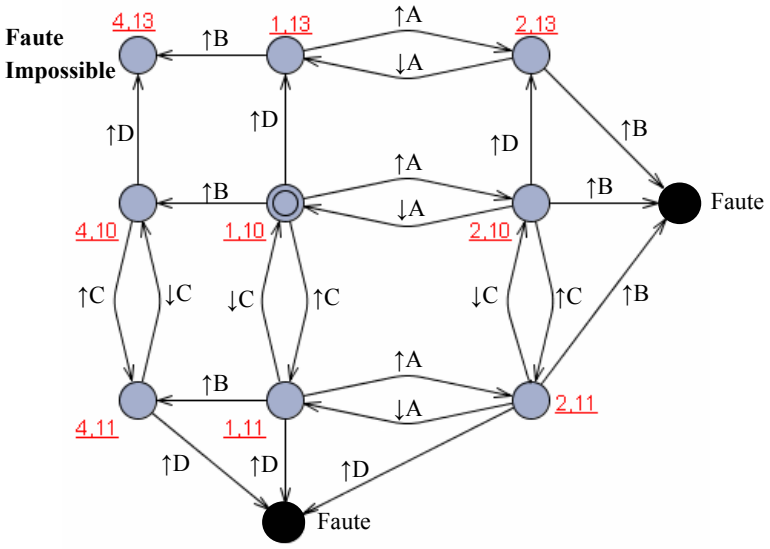


Figure 5.18. Automate équivalent à l'association de la figure 5.17

4.3 Association dont le sommet est une porte ET dont les entrées sont les sorties de portes POR

4.3.1 Contrôle de cohérence

Soit V_{com} l'ensemble des variables communes à deux portes POR, tel que $V_{com} \subseteq V$. Pour tout ensemble de portes POR qui contiennent un couple de variables d'entrées $(V_j,V_k) \in V_{com}$, si on a que V_j est *l'entrée conditionnée* d'une des portes et que V_k est aussi *l'entrée conditionnée* de l'autre porte (où V_j apparaît également), alors il existe une *incohérence*.

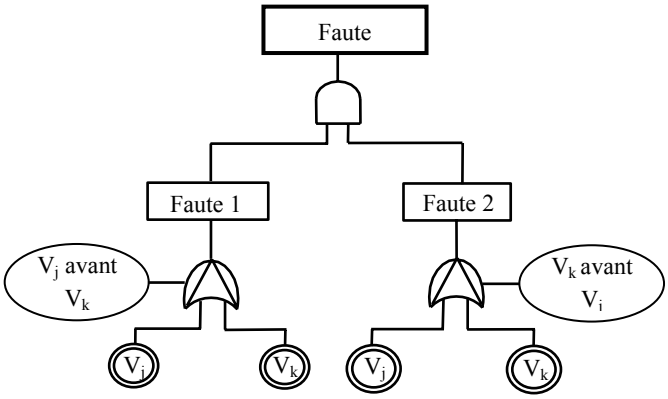


Figure 5.19. Exemple d'incohérence pour un couple de portes POR associées par une porte ET

4.3.2 Modélisation de l'association

L'automate résultant est montré dans la figure 5.20. Il existe alors un seul état défaillant (*2,11*). Pour l'atteindre, il y a deux séquences minimales : ↑B↑C, ↑C↑B.

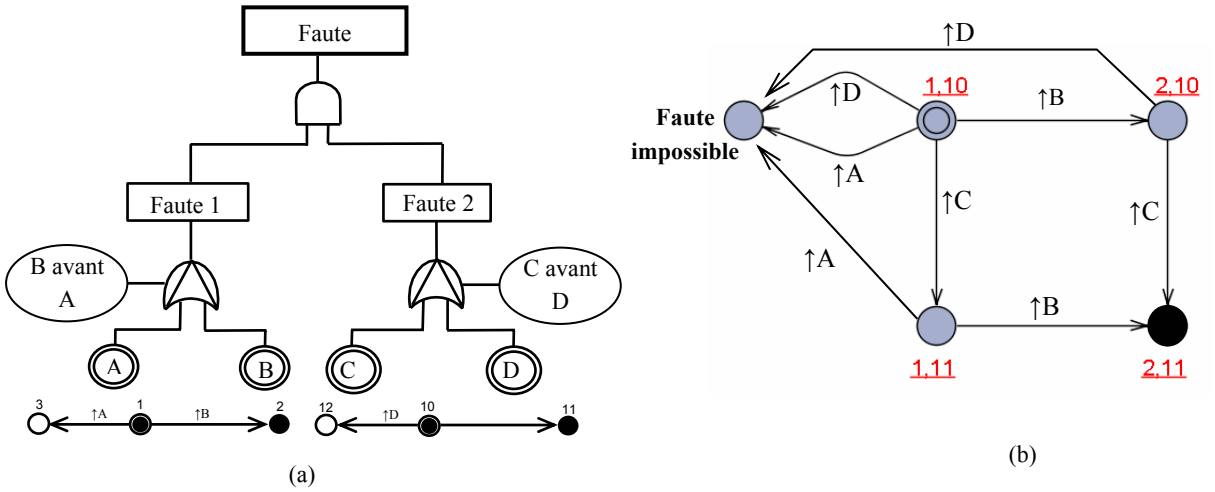


Figure 5.20. Association de deux portes POR par une porte ET

4.4 Association dont le sommet est une porte OU, et dont les entrées sont les sorties de portes POR

Comme à la sous-section 4.2, nous n'effectuerons pas de contrôle de cohérence. L'AdD et l'automate sont montrés dans la figure 5.21. La faute Faute est VRAIE soit, si Faute1 est vraie ou soit si Faute2 est vraie. Donc, il suffit que chaque automate modélisant les portes POR atteigne son état de faute. Par conséquent, dans l'automate équivalent, il existe 4 états de faute (fusionnés). Quatre séquences minimales sont alors possibles : ↑B, ↑C, ↑D↑B, ↑A↑C.

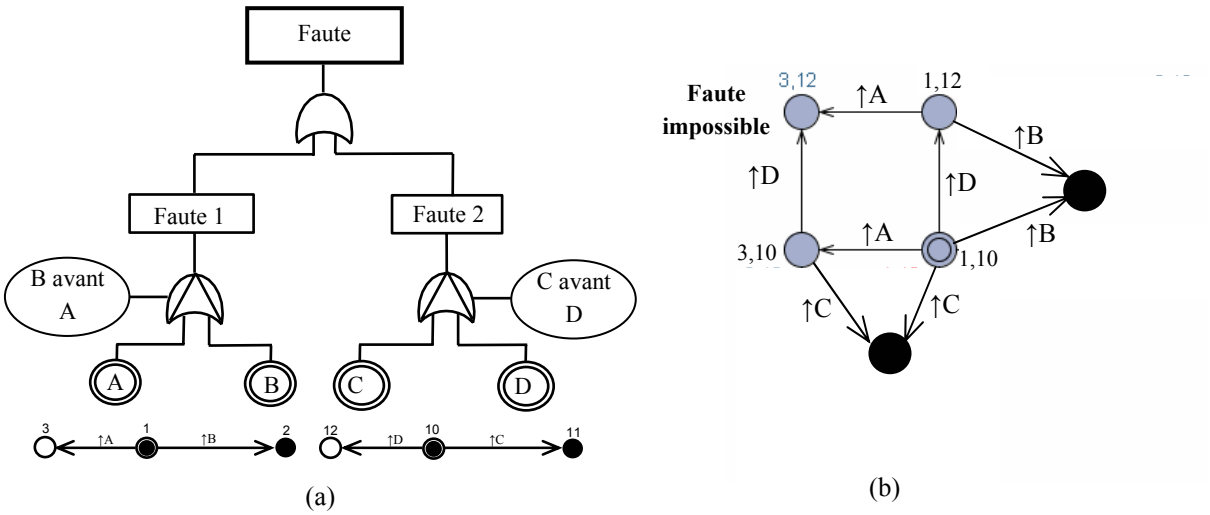


Figure 5.21. (a) Association dont le sommet est une porte OU dont les entrées sont les sorties de portes POR et (b) automate décrivant son comportement

4.5 Association dont le sommet est une porte ET, et dont les entrées sont des sorties de portes PAND-POR

4.5.1 Contrôle de cohérence

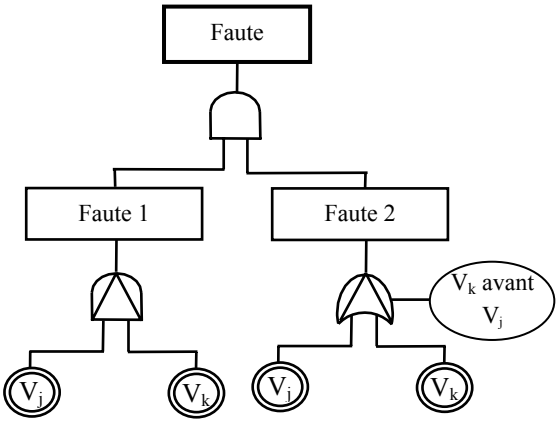


Figure 5.22. Incohérence d'une association dont le sommet est une porte ET dont les entrées sont les sorties de portes PAND-POR

Soit V_{com} l'ensemble des variables communes à deux portes PAND et POR (figure 5.22), tel que $V_{com} \subseteq V$. Pour le couple de portes POR-PAND et pour tout couple d'entrées $(V_j,V_k) \in V_{com}$, si on vérifie que dans la porte PAND V_j *avant* V_k et que dans la porte POR *l'entrée conditionnée* est V_k, alors il existe une *incohérence*.

4.5.2 Modèle de l'association

La défaillance se produit si la sortie de la porte PAND et celle de la porte POR sont vraies (figure 5.23).

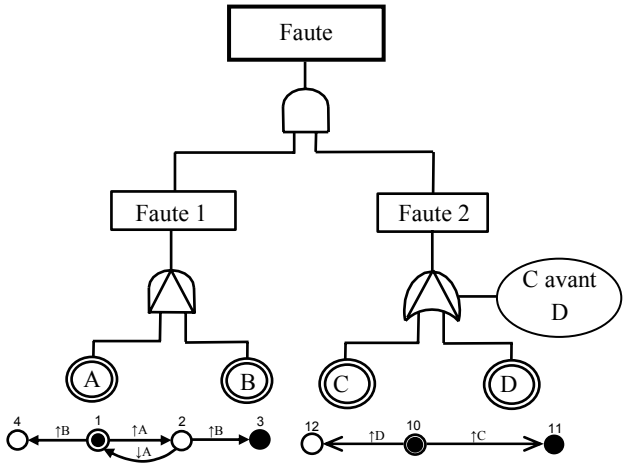


Figure 5.23. Association de portes PAND-POR par une porte ET

L'automate décrivant le comportement de l'association est représenté dans la figure 5.24. Il est composé de 7 états, dont *(1,10)* est l'état initial. Il existe un seul état de faute, (*3,11)* représenté par un disque noir dans la figure, ce qui est normal car la porte ET oblige les fautes Faute1 et Faute2 à être tous les deux vraies. On peut définir alors trois séquences minimales : ↑A↑B↑C, ↑A↑C↑B, ↑C↑A↑B.

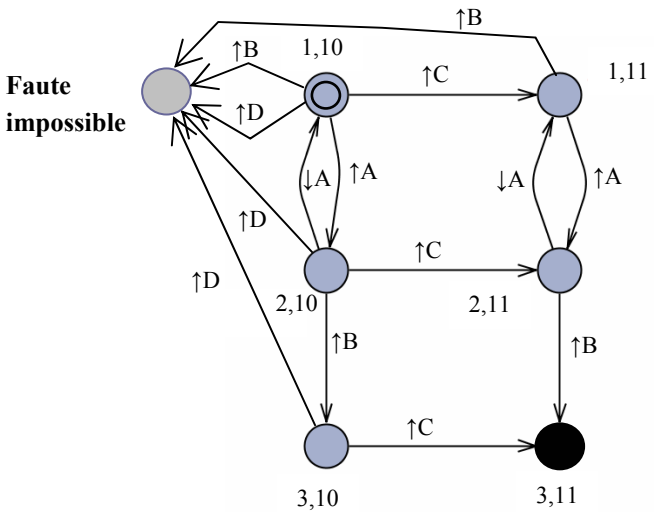
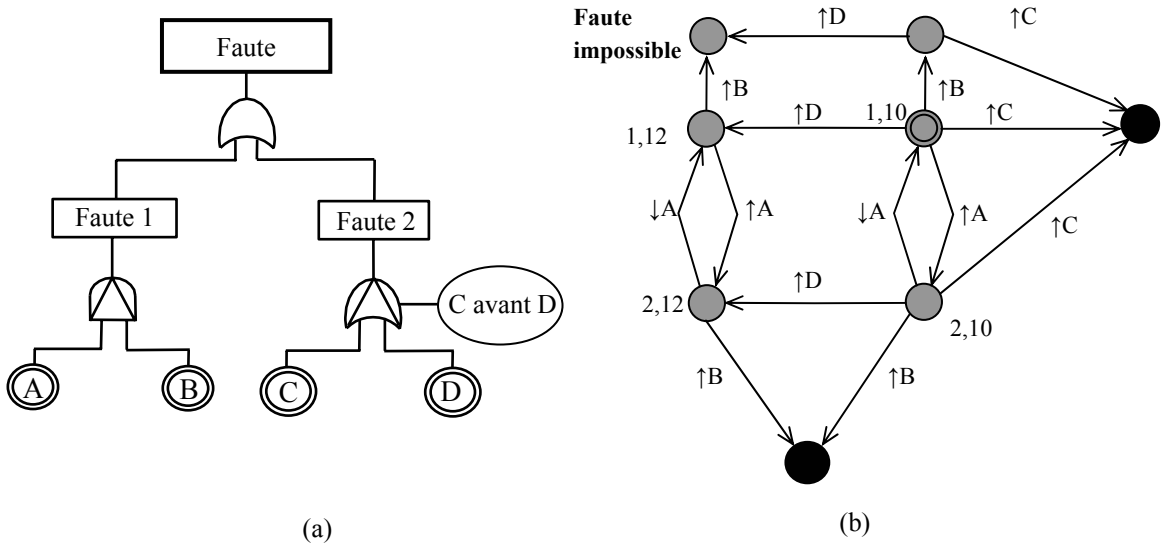


Figure 5.24. Automate formalisant le comportement de l'association de la figure 5.23

4.6 Association dont le sommet est une porte OU dont les entrées sont des sorties de portes PAND-POR

Il n'y a pas de contrôle de cohérence nécessaire pour cette association (figure 5.25.a). L'automate obtenu est à l'origine composé de 11 états, dont cinq sont des états de faute. Ces états sont alors fusionnés. L'automate réduite est montré en figure 5.25.b. Six séquences minimales sont alors possibles : ↑C, ↑A↑C, ↑A↑B, ↑B↑C, ↑D↑A↑B, ↑A↑D↑B.



(a) (b)

Figure 5.25. (a) AdD et (b) automate de l'association de portes PAND-POR par une porte OU

4.7 Association dont le sommet est une porte ET, et dont les entrées sont des sorties de portes PAND et des variables

La variable V_j , que l'on suppose différente des entrées de la porte temporelle, est modélisée comme un automate à deux états et deux transitions (figure 5.26.a). On peut alors appliquer la méthode décrite au début de la section.

L'automate obtenu possède un seul état de faute (*3,11*), tel que l'on peut le voir dans la figure 5.26.b.

Trois séquences minimales sont alors possibles :

$\uparrow V_j \uparrow A \uparrow B$, $\uparrow A \uparrow V_j \uparrow B$, $\uparrow A \uparrow B \uparrow V_j$ (V_j peut apparaître avant, après ou au milieu de la séquence ↑A↑B).

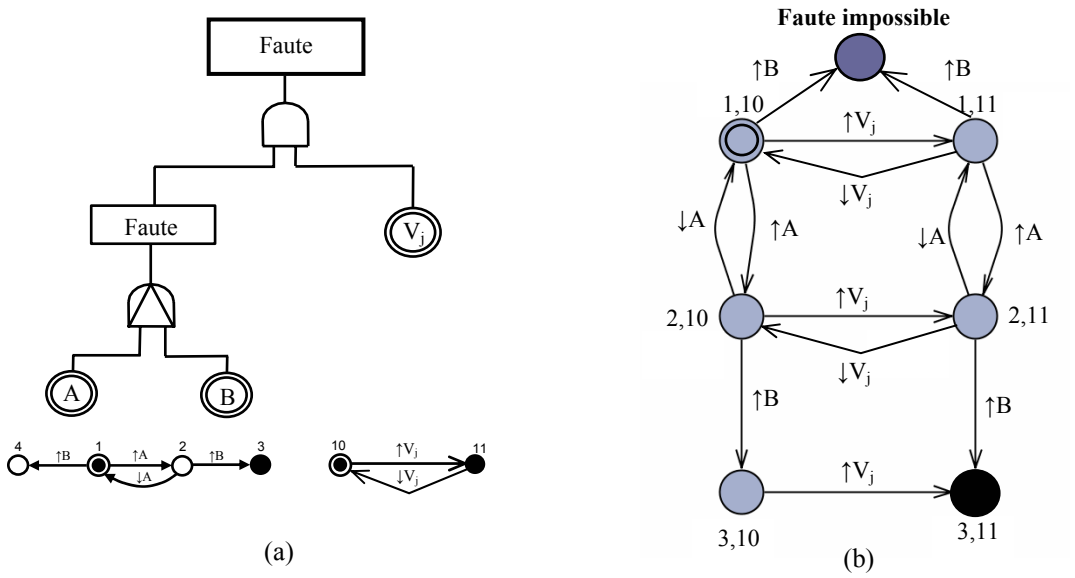


Figure 5.26. (a) AdD et (b) automate équivalent de l'association porte PAND ET une variable logique

4.8 Association dont le sommet est une porte OU, et dont les entrées sont des sorties de portes PAND ou des variables

L'automate obtenu de façon similaire possède à l'origine sept états, dont quatre défaillants (fusionnés dans la figure 5.27.b). On peut remarquer qu'aucun état ne traduit l'impossibilité de faute, l'apparition de ↑V_j conduisant obligatoirement à un état de faute. Quatre séquences minimales sont alors possibles : ↑V_j, ↑A↑V_j , ↑A↑B, ↑B↑V_j.

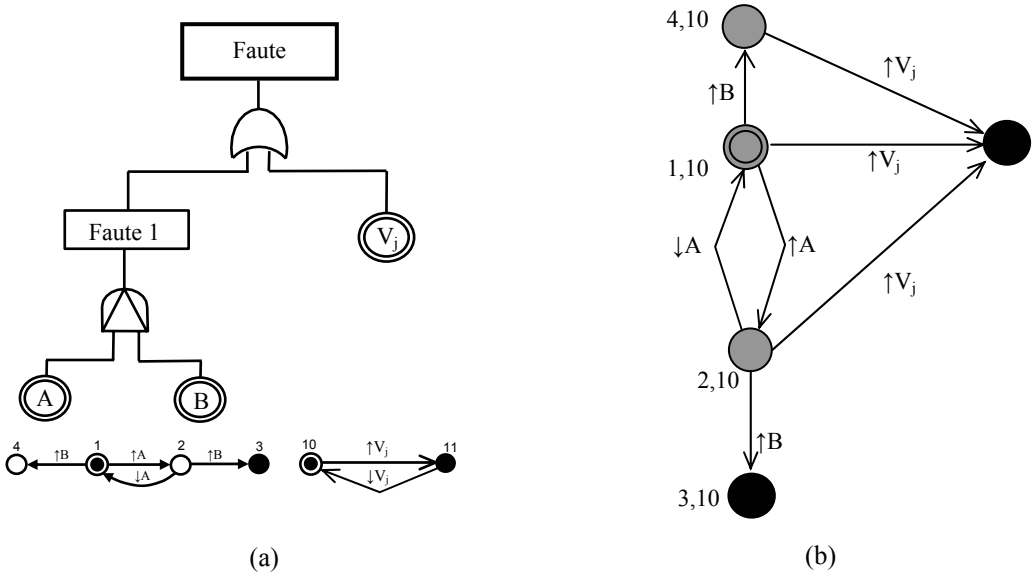


Figure 5.27. (a) AdD et (b) automate de l'association porte PAND OU une variable logique

4.9 Association dont le sommet est une porte ET dont les entrées sont des sorties de portes POR et des variables

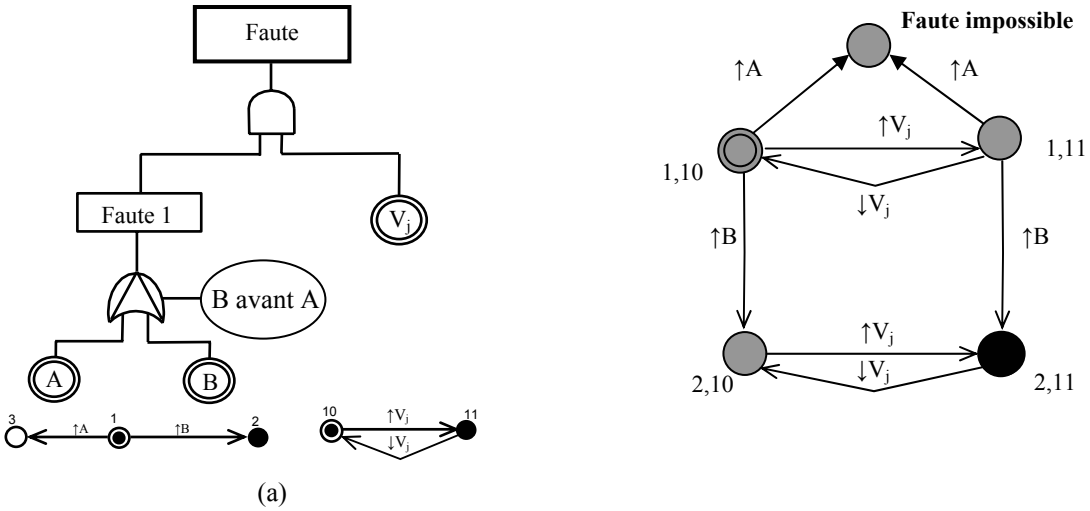


Figure 5.28. (a) AdD et (b) automate de l'association porte POR ET une variable logique

L'automate équivalent (figure 5.28.b) possède cinq états, dont un seul défaillant. Deux séquences minimales sont alors possibles : $\uparrow V_j \uparrow B$, $\uparrow B \uparrow V_j$.

4.10 Association dont le sommet est une porte OU dont les entrées sont des sorties de portes POR ou des variables

L'automate équivalent (figure 5.29.b) possède cinq états, dont trois caractéristiques de la faute (fusionnés dans la figure). Comme en 4.8, on peut remarquer qu'aucun état ne traduit l'impossibilité de faute. Trois séquences minimales sont alors possibles : $\uparrow V_j$, $\uparrow B$, $\uparrow A \uparrow V_j$.

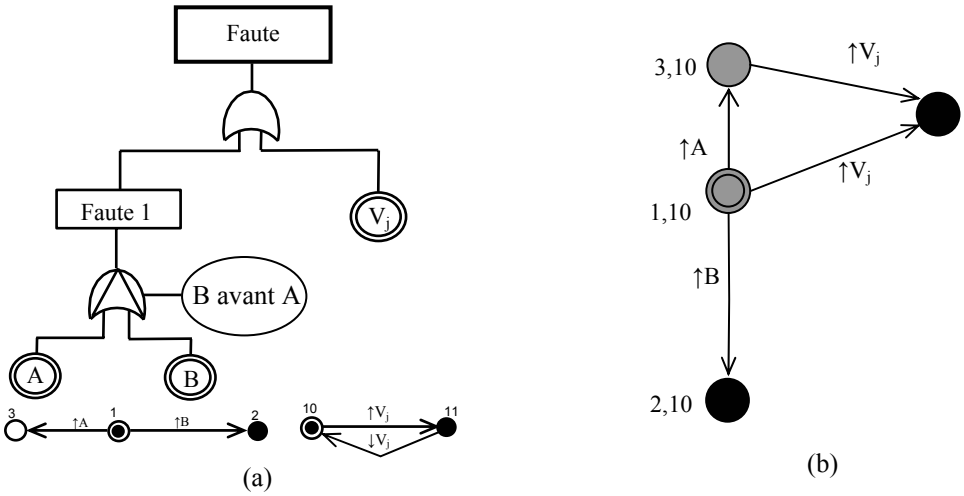


Figure 5.29. (a) AdD et (b) automate de l'association porte POR OU une variable logique

5 Associations dont le sommet est une porte temporelle

Les associations étudiées ci-après en 5.1, 5.2, 5.3, 5.4 se déduisent de celles étudiées en 4.1, 4.2, 4.3, 4.4 en remplaçant la porte statique située au sommet par une porte temporelle (porte ET remplacée par une porte PAND, porte OU remplacée par une porte POR). Ces portes PAND et POR introduites au sommet des associations **imposent l'ordre** d'apparition des fautes qui sont leurs entrées. Pour établir les modèles formels de ces nouvelles associations, sous forme d'automates, nous nous appuierons sur les résultats obtenus en section 4. A partir des automates obtenus alors, il conviendra de :

- **Vérifier que les états défaillants le sont toujours avec la logique introduite par la porte temporelle** ; sinon ces états correspondent à l'absence de faute et peuvent être fusionnés avec d'autres états du même type ;
- **Supprimer les transitions conduisant aux états défaillants qui correspondent à des séquences d'événements contraires à l'ordre imposé par la porte sommet**. Les états non atteignables depuis l'état initial suite à ces suppressions, s'ils existent, seront également supprimés.

Cette démarche est illustrée sur les quatre exemples qui suivent.

5.1 Association dont le sommet est une porte PAND, et dont les entrées sont des sorties de portes PAND

Dans ce cas (figure 5.30.a) l'automate obtenu en 4.1 et servant de base à la construction de l'automate modélisant le comportement de cette nouvelle association est rappelé en 5.30.b.

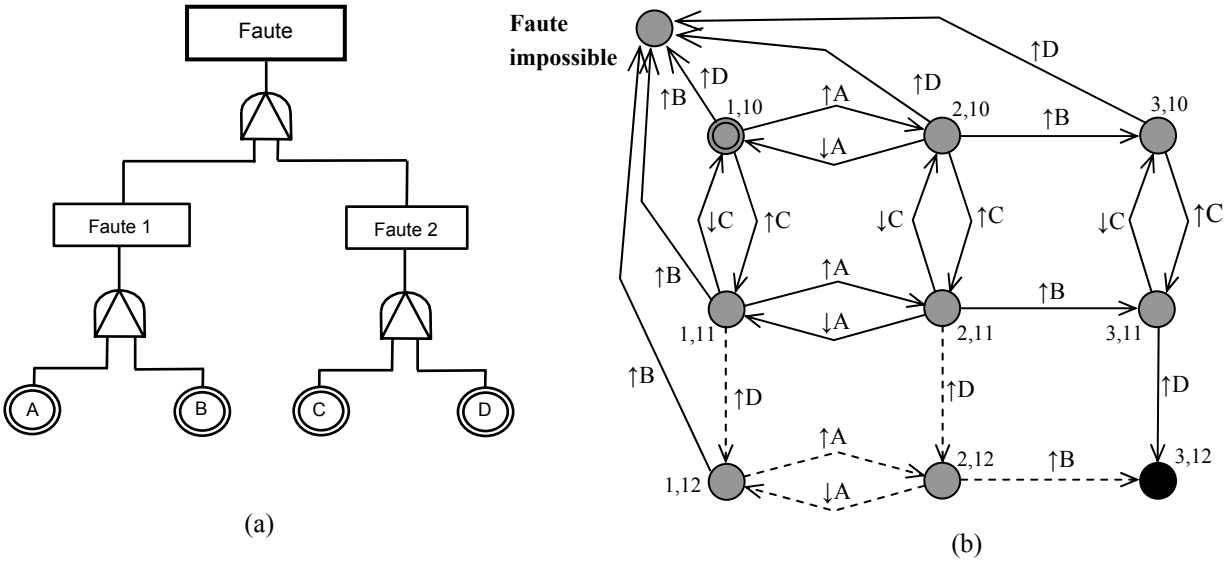


Figure 5.30. (a) AdD et (b) automate obtenu en 4.1

Les transitions en traits pointillés sont celles contraires à la logique de la porte PAND sommet et qui doivent être éliminées.

Après suppression des transitions inadéquates et des états non atteignables depuis l'état initial, on obtient le modèle de la figure 5.31 qui comporte un état de faute (disque noir dans la figure) et trois séquences minimales : ↑A↑B↑C↑D, ↑A↑C↑B↑D et ↑C↑A↑B↑D.

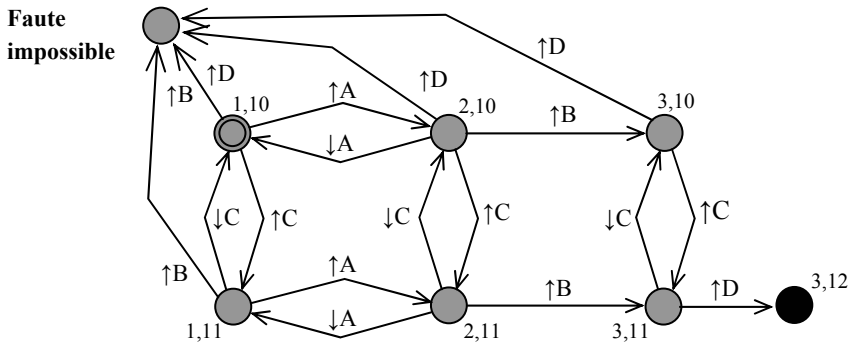


Figure 5.31. Automate équivalent de la figure 5.30.a

5.2 Association dont le sommet est une porte POR, et dont les entrées sont des sorties de portes PAND

Cette association est représentée en figure 5.32.

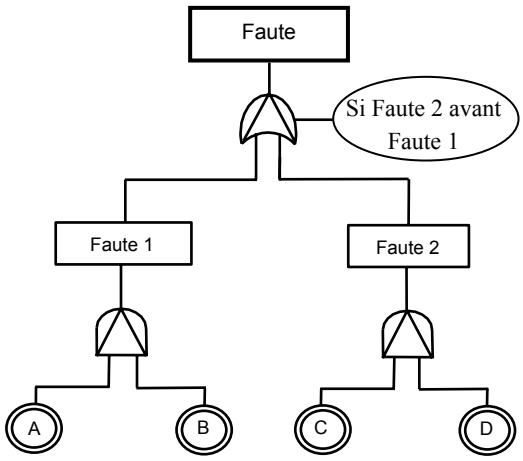


Figure 5.32. Association de portes PAND par une porte POR

L'automate obtenu en 4.2 est rappelé en figure 5.33. On obtient finalement le modèle de la figure 5.34 qui est composé de huit états dont un état rassemblant les états de faute et un autre rassemblant les états où la faute est impossible.

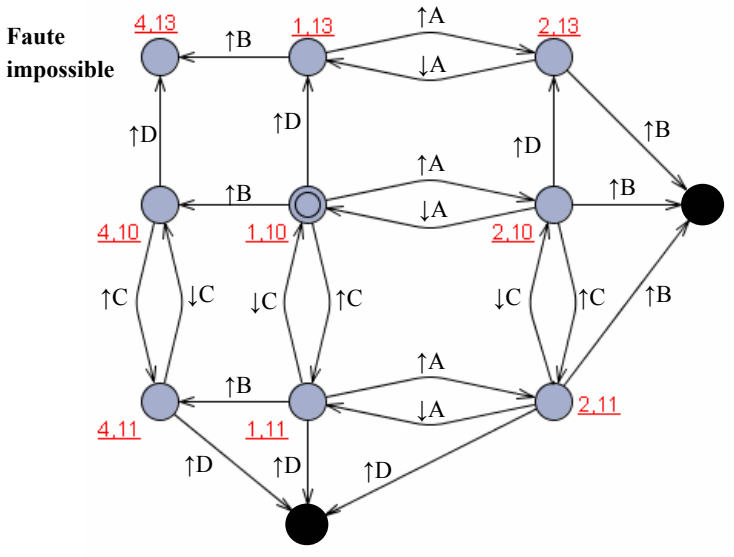


Figure 5.33. Automate obtenu en 4.2

Cinq séquences minimales : ↑C↑D, ↑A↑C↑D, ↑C↑A↑D, ↑B↑C↑D, ↑C↑B↑D peuvent alors être identifiées.

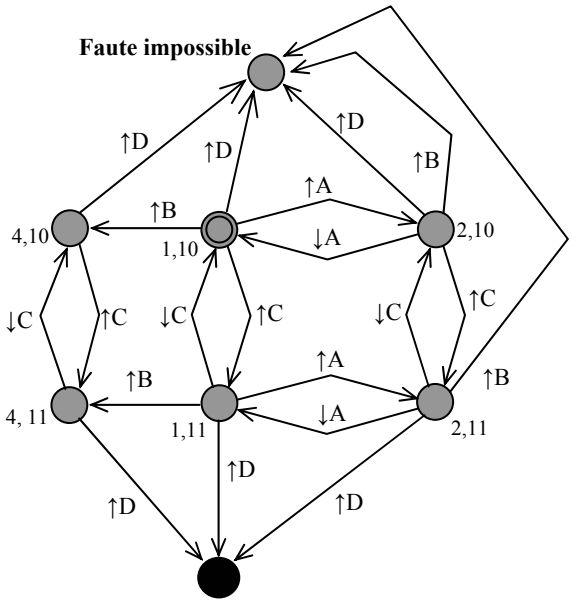


Figure 5.34. Automate de l'association de la figure 5.32

5.3 Association dont le sommet est une porte PAND, et dont les entrées sont des sorties de portes POR

De manière similaire, on peut obtenir l'automate de la figure 5.35.b, qui possède quatre états, dont un seul de faute. Une seule séquence minimale est alors possible : ↑B↑C.

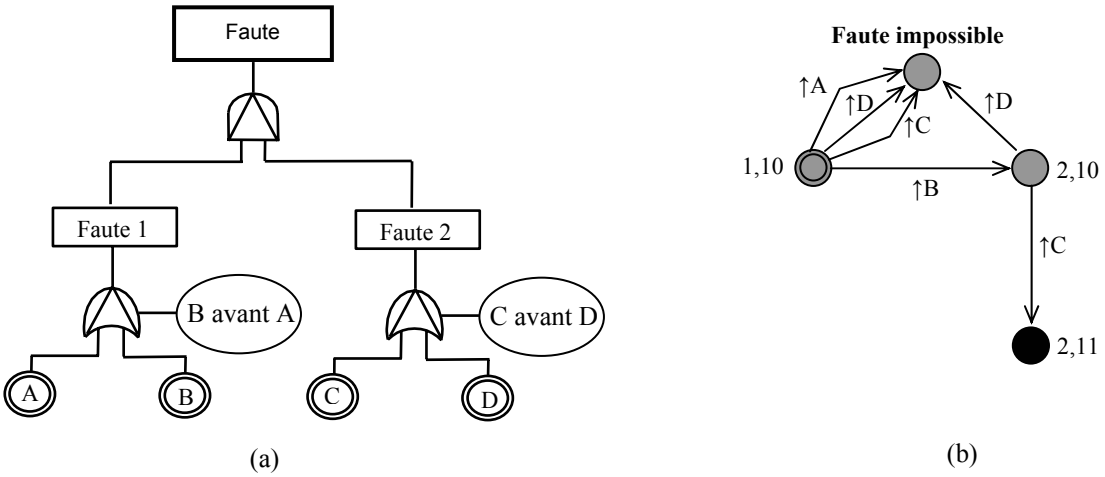


Figure 5.35. (a) AdD et (b) automate de l'association portes POR par une porte PAND

5.4 Association dont le sommet est une porte POR dont les entrées sont des sorties de portes POR

L'automate modélisant le comportement de cette association (figure 5.36.b) possède quatre états (deux états de faute ont été fusionnés). Deux séquences minimale sont possibles : ↑C, ↑A↑C.

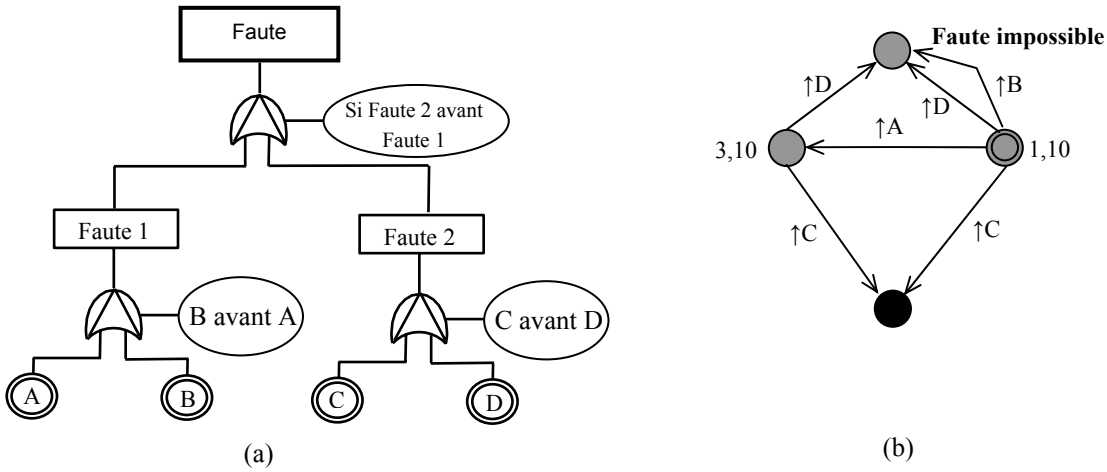


Figure 5.36. (a) AdD et (b) automate de l'association deux portes POR par une porte POR

6 Associations faisant intervenir des portes temporisées

Nous avons déjà indiqué qu'une porte temporisée seule ne modélise pas une défaillance mais le comportement temporisé d'une variable logique. Cependant, les études de cas que nous avons conduites (chapitre 3, sous-section 4.2, et chapitre 6) nous ont montré que des associations comportant ces portes pouvaient être très utiles pour modéliser des fautes fonction du temps physique. Il convient donc d'élaborer les modèles formels de ces associations, sous la forme d'automates temporisés, afin de pouvoir en déduire des propriétés formelles à vérifier.

6.1 Association des portes FORNEXT n et WITHIN n avec une porte ET

Un exemple d'utilisation de cette association (figure 5.37) sera donné au prochain chapitre. Pour l'instant, nous nous contenterons d'indiquer que la faute se produit si l'occurrence d'un événement (↑B dans le cas étudié) ne se produit pas dans l'intervalle de temps de n unités de temps suivant l'occurrence d'un autre événement (↑A dans le cas étudié), et à condition que la variable A demeure vraie dans cet intervalle de temps.

Les chronogrammes de la figure 5.38 illustrent de façon informelle le comportement de cette association.

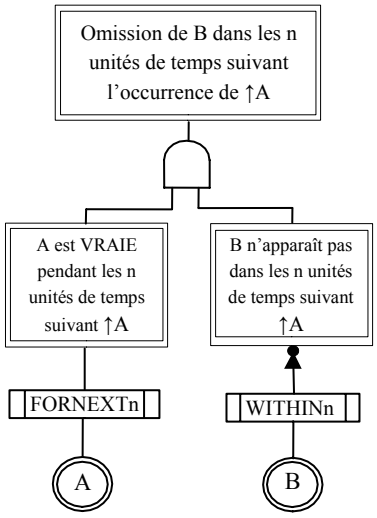


Figure 5.37. AdD de l'association des portes Fornext n-Within n

Pour élaborer l'automate décrivant formellement le comportement de l'association, il faut :

— Elaborer l'automate correspondant à la négation d'une porte WITHIN n
— Elaborer l'automate correspondant à la combinaison par une porte AND des portes FORNEXT n et WITHIN n complementée.

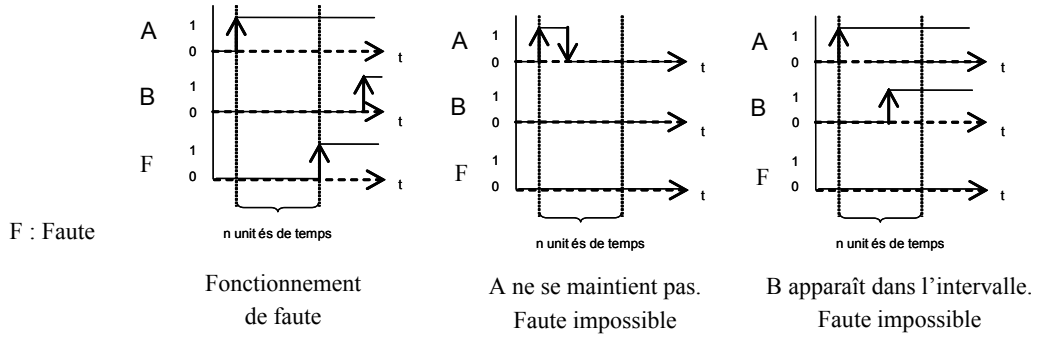


Figure 5.38. Chronogrammes de fonctionnement de l'association de la figure 5.37

Elaboration de l'automate correspondant à une porte WITHIN n complémentée.

Cet automate doit exprimer que le front montant de B **ne se produit pas** dans l'intervalle de n unités de temps suivant l'instant courant, qui est représenté par la date d'occurrence de l'événement ↑Start dans l'automate associé à la porte WITHIN. Il s'obtient donc à partir de l'automate traduisant le comportement de la porte WITHIN (Cf. chapitre 4, sous-section 5.2) en permutant les transitions issues de l'état 2 (figure 5.39).

Remarque : Dans le modèle de la figure 5.39, comme dans les autres modèles formels de cette section, seuls les champs non vides des transitions seront représentés.

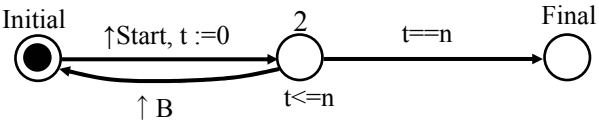


Figure 5.39. Automate correspondant à la porte WITHIN n complémentée

Elaboration du modèle formel de l'association

Nous utiliserons pour ce faire les modèles obtenus pour les portes *WITHIN n* complémentée et *FORNEXT n* (figure 5.40).

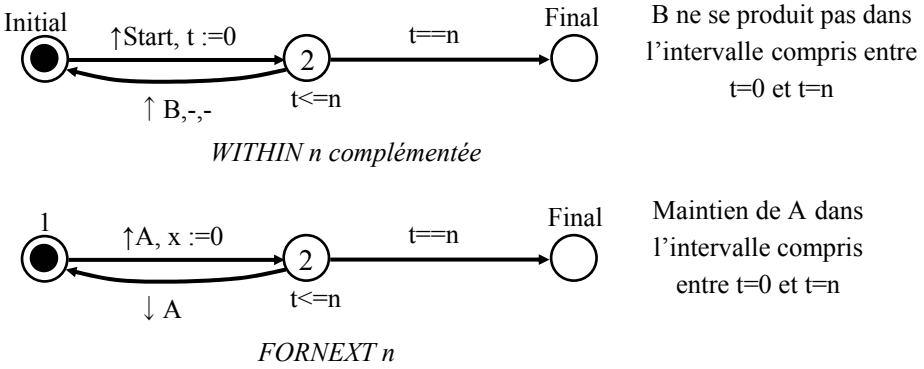


Figure 5.40. Automates correspondants aux portes WITHIN n complémentée et FORNEXT n

Le modèle formel correspondant à l'association étudiée s'obtient en synchronisant les deux automates, c.a.d. en remplaçant l'événement ↑Start par ↑A dans l'automate déduit de WITHIN complémenté (la combinaison par une porte AND implique en effet d'utiliser l'événement ↑A comme événement déclencheur des évolutions de cet automate) et en respectant l'hypothèse d'asynchronisme des événements que nous avons choisie, qui conduit à considérer que les deux événements ↑A et ↑B ne peuvent être simultanés..

L'automate obtenu est représenté en figure 5.41. La première transition mène à l'état 2 auquel est associé un invariant fonction de *n*. Trois évolutions sont alors possibles : si A ne se maintient pas n unités de temps ou si l'événement ↑B se produit durant cet intervalle de temps, l'automate retourne à l'état *Initial*, sinon l'automate va à l'état *Faute* lorsque la valeur de l'horloge est égale à n.

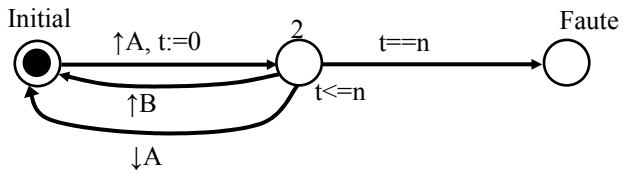


Figure 5.41. Automate correspondant à l'association de la figure 5.37

6.2 Association dont le sommet est une porte POR, et dont les entrées sont des sorties d'une porte FORNEXT n et des variables

Cette association (figure 5.42) a été rencontrée au chapitre trois, section 4.2. La faute se produit si l'événement ↑B se produit dans un intervalle de n unités de temps suivant l'occurrence de l'événement ↑A, à condition que A reste vraie.

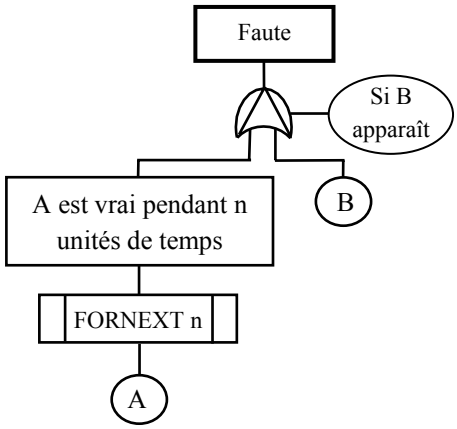


Figure 5.42. Association étudiée

L'automate proposé pour modéliser cette association est donné à la figure 5.43. A partir de l'état initial, l'occurrence de ↑A déclenche l'évolution et initialise l'horloge. De l'état 2, on peut évoluer vers l'état de faute si ↑B se produit. Si ↑A se produit alors que la valeur de l'horloge est inférieure à n, on revient à l'état initial. Si aucun des deux événements ne s'est produit à la fin de l'intervalle de n unités de temps, on évolue alors vers l'état 3 qui indique que la faute ne s'est pas produite.

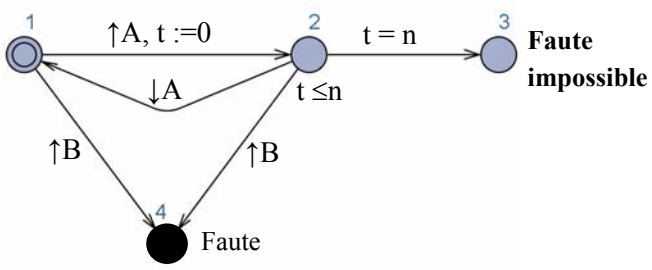


Figure 5.43. Automate modélisant le comportement de l'association de la figure 5.42

7 Conclusion

Nous avons montré dans ce chapitre qu'il était possible d'obtenir le modèle formel du comportement décrit par une association comprenant des portes statiques et temporelles à partir des modèles formels des comportements de ces portes et en se basant sur des techniques de composition d'automates. La méthode proposée peut être étendue à d'autres associations plus complexes.

Pour les associations types faisant intervenir des portes temporisées, des modèles formels, sous forme d'automates temporisés, ont également été obtenus à partir des modèles formels élémentaires. L'utilisation de ces modèles d'associations sera montrée dans le chapitre 6.

Chapitre 6.

Etude de cas

La méthode présentée dans les chapitres précédents sera maintenant appliquée à un système réel. L'étude de cas développée dans ce sixième et dernier chapitre du mémoire de thèse permettra donc de montrer la validité de notre méthode, c'est à dire la possibilité d'obtenir les propriétés formelles en vue de la vérification d'un contrôleur logique à partir d'une AdD. Ce chapitre comprend trois sections :

- Dans la première, le système servant de support à l'étude (partie opérative et contrôleur logique) sera présenté.
- La deuxième section traite de l'analyse par AdD de trois événements indésirables choisis parmi l'ensemble des fautes possibles du système. A partir des arbres de défaillance construits, les propriétés formelles à vérifier seront déterminées.
- La dernière section est dédiée à la vérification formelle du contrôleur. Dans cette section, nous ferons tout d'abord une présentation du model-checker UPPAAL, qui a été choisi pour notre étude, pour exposer ensuite la démarche de construction du modèle du système (contrôleur logique et partie opérative). A la fin de la section, les résultats de la vérification seront commentés.

1 Le système étudié

1.1 Partie opérative

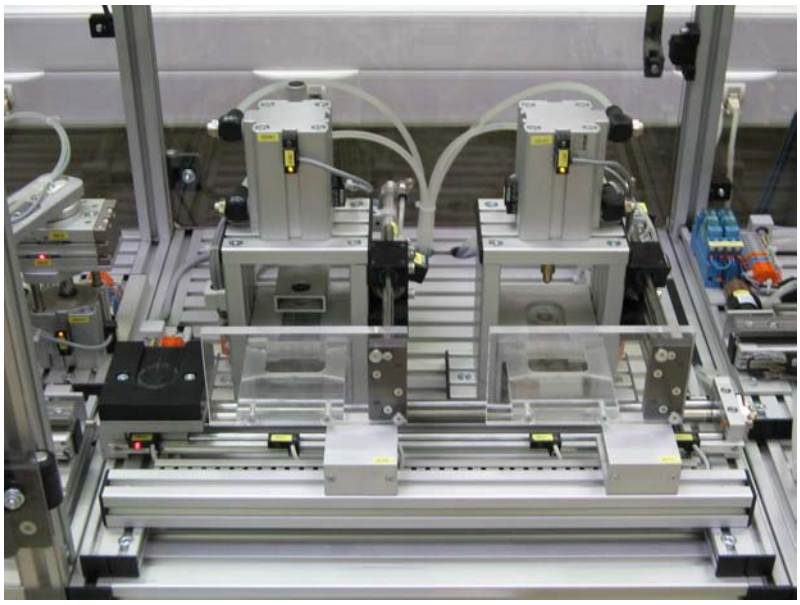
Le système qui servira d'objet à l'étude de cas fait partie d'un ensemble mécatronique Bosch-Rexroth disponible au LURPA. Cet ensemble didactique est composé de quatre stations de travail (figure 6.1). La fonction principale de cet ensemble est la classification de pignons selon leur matériau ainsi que l'extraction/insertion de paliers dans les pignons.

Figure 6.1. Ensemble mécatronique Bosch-Rexroth

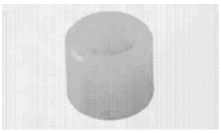
L'étude se focalisera sur la station de travail 3 (figure 6.2) dont l'objectif est l'extraction/insertion de paliers dans les pignons. Cette station extrait en effet les paliers des pignons qui en comportent initialement un, et insère un palier dans tout pignon initialement sans palier.

Cette station est elle-même divisée en trois sous-systèmes. Le premier est une presse d'insertion de paliers (figure 6.3.a), qui est composée d'un vérin vertical pour insérer les paliers (vérin d'insertion InPre) et d'un vérin horizontal qui introduit les pignons à l'intérieur de la presse (vérin d'alimentation InFeed). Le deuxième est une presse d'extraction de paliers (figure 6.3.b), avec un vérin vertical pour extraire les paliers (vérin d'extraction OutPre) et un vérin horizontal qui introduit les pignons à l'intérieur de la presse (vérin d'alimentation OutFeed). Le troisième est composé d'un chariot transporteur de pignons qui se déplace tout au long d'un convoyeur linéaire comportant quatre détecteurs : un pour indiquer que le chariot est à gauche (signal PosLeft), un pour indiquer que le chariot est en face de la presse d'insertion (signal PosIn), un pour la position en face de la presse d'extraction (signal PosOut)

et un pour indiquer que le chariot est en position de transfert vers la station suivante (signal PosRight), ainsi que deux vérins bloqueurs pour l'arrêt du chariot en face des presses (figure 6.3.c).

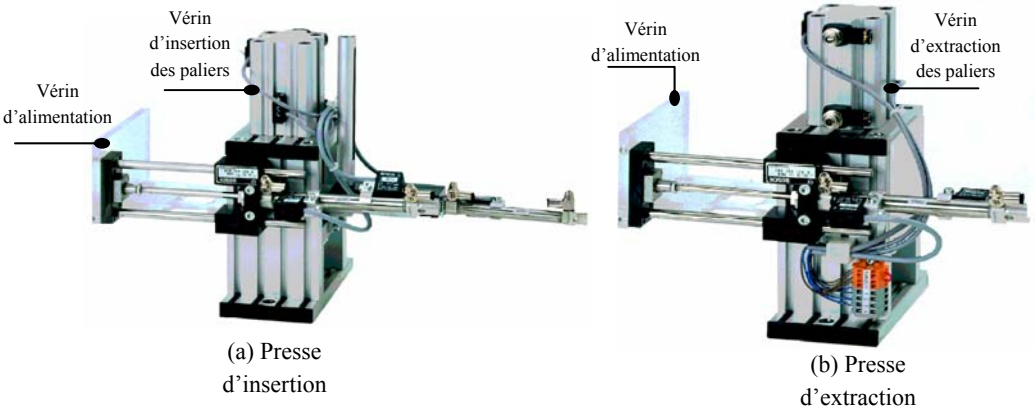


Pignon



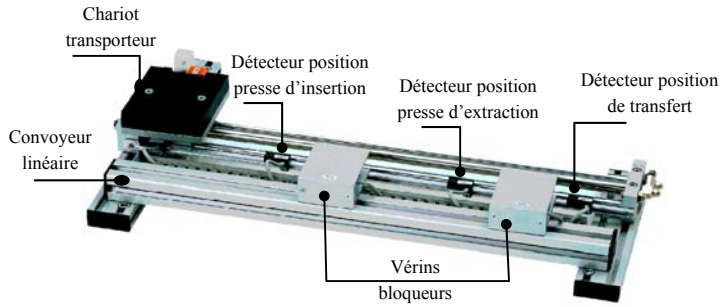
Palier

Figure 6.2. La station 3 de la machine Bosch, pignon et palier



(a) Presse d'insertion

(b) Presse d'extraction



(c) Chariot transporteur et convoyeur

Figure 6.3. Les trois sous-systèmes de la station 3

1.2 Le contrôleur logique

La station est contrôlée par un Automate Programmable Industriel (API), dont on veut vérifier le programme utilisateur. Les entrées et les sorties de l'API sont montrées dans le schéma de la figure 6.4 et listées dans le tableau 6.1.

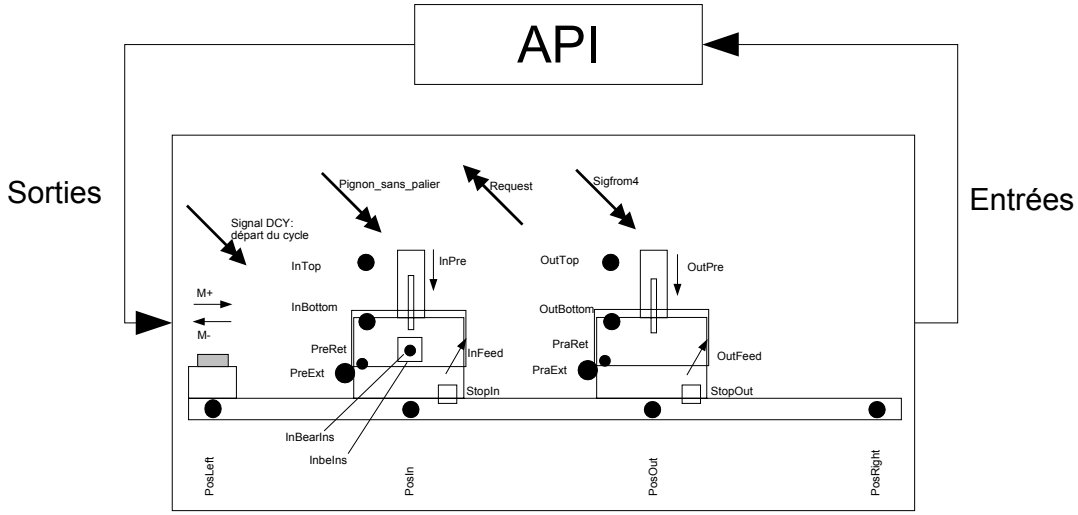


Figure 6.4. Schéma de la station 3

Sorties		**Entrées**	
Mouvement vers la droite	M+	Position initiale	PosLeft
Mouvement vers la gauche	M-	Position devant la presse d'insertion	PosIn
Sortir vérin d'insertion	InPre	Position devant la presse d'extraction	PosOut
Rentrer vérin d'alimentation de la presse d'insertion	InFeed	Position de transfert	PosRight
Sortir vérin d'alimentation des paliers	InbeIns	Vérin d'insertion rentré	InTop
Sortir vérin d'extraction	OutPre	Vérin d'insertion sorti	InBottom
Rentrer vérin d'alimentation de la presse d'extraction	OutFeed	Vérin d'alimentation de la presse d'insertion sorti	PreExt
Rentrer vérin d'arrêt devant la presse d'insertion	StopIn	Vérin d'alimentation de la presse d'insertion rentré	PreRet
Rentrer vérin d'arrêt devant la presse d'extraction	StopOut	Vérin d'alimentation des paliers sorti	InBearIns
Signal envoyé à la station 2	Request	Vérin d'extraction rentré	OutTop
		Vérin d'extraction sorti	OutBottom
		Vérin d'alimentation de la presse d'extraction sorti	PraExt
		Vérin d'alimentation de la presse d'extraction rentré	PraRet
		Signal départ du cycle	DCY
		Signal pignon sans palier	Pignon_sans_palier
		Signal du poste 4	Sigfrom4

Tableau 6.1. Liste des entrées et sorties du contrôleur

La spécification du fonctionnement de la station 3 est réalisée avec un grafcet, montré en figure 6.5.

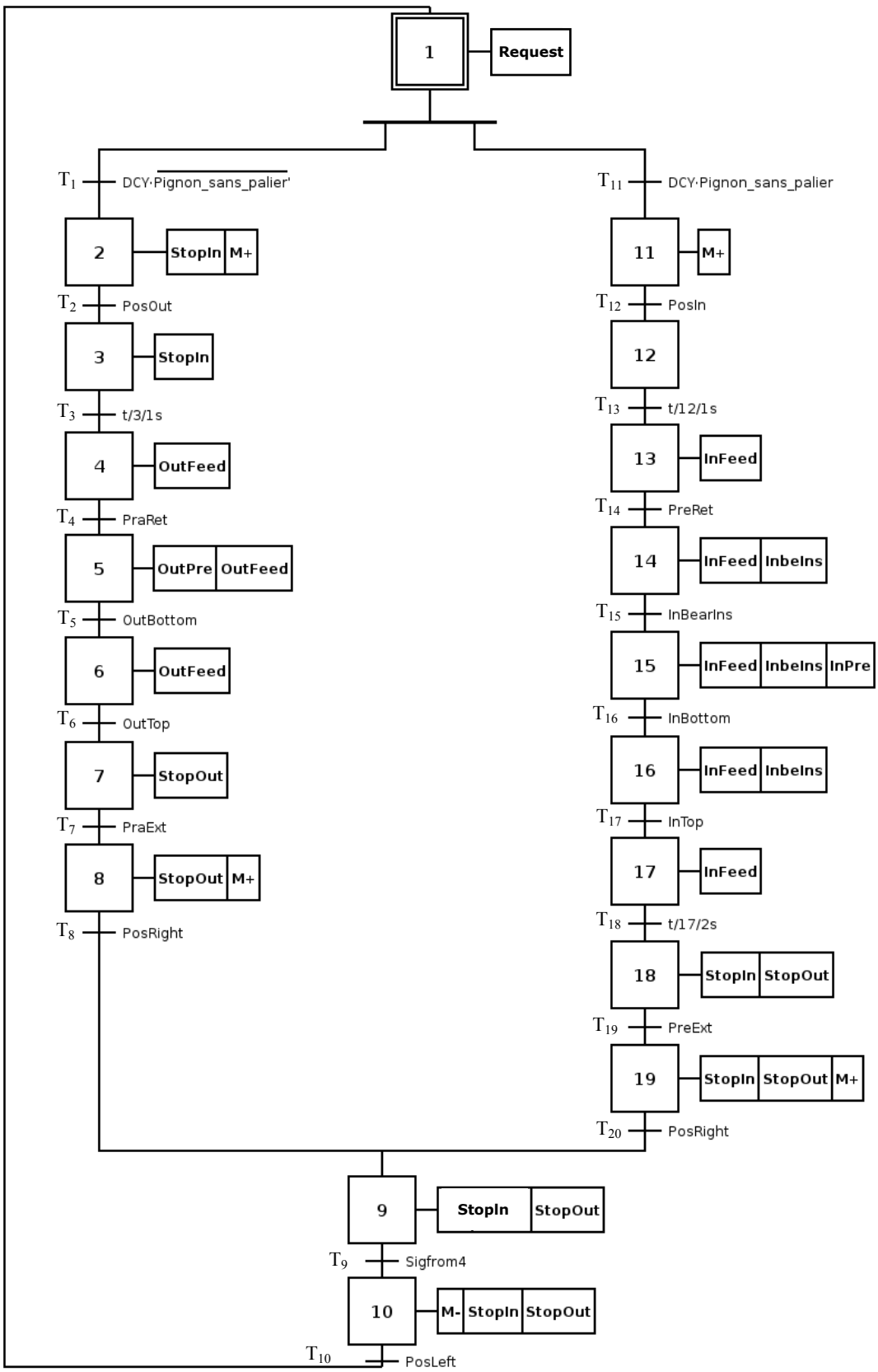


Figure 6.5. Grafcet de commande de la station 3

Le fonctionnement est le suivant. Le chariot étant vide et dans la position initiale (PosLeft), le contrôleur émet un signal (Request) à la station 2 pour lui demander un pignon. La présence ou absence de palier dans le pignon est indiquée par le signal Pignon_sans_palier, qui est envoyé par la station de travail 2. Ce signal vaut 1 (est VRAI) si le pignon ne contient pas de palier et vaut 0 si, dans le cas contraire, le pignon en contient un. L'arrivée d'un pignon sur le chariot démarre le processus. Le chariot transporteur avance de la position initiale à l'une des deux presses. Si le pignon porte un palier, alors il est envoyé à la presse d'extraction pour l'enlever (voir branche gauche du grafcet de la figure 6.5). Au contraire, s'il n'y a pas de palier dans le pignon, il est alors envoyé à la presse d'insertion (branche droite du grafcet). L'avance du chariot est indiquée par le signal de commande du moteur, ainsi M+ indique le mouvement vers la droite, et M- le mouvement vers la gauche.

S'il n'y a pas de palier dans le pignon (suivre la branche droite du grafcet), le chariot avance à droite jusqu'à ce qu'il atteigne la position située devant la presse d'insertion. Dans cette position il s'arrête. Lorsque le chariot est arrêté devant la presse depuis 1 seconde, le vérin d'alimentation (InFeed) déplace le pignon du chariot à l'intérieur de la presse. Ensuite, un autre vérin place un palier sur le pignon pour qu'il soit inséré, ce qui est repéré par le détecteur InBearIns. Le vérin d'insertion InPre peut alors sortir pour insérer le palier. Une fois le palier inséré dans le pignon (InBottom = 1), le vérin d'insertion rentre ce qui permet au vérin qui place les paliers de rentrer lorsque le vérin d'insertion est en position haute. Le vérin d'alimentation remet alors le pignon sur le chariot et celui-ci continue son avance à droite jusqu'à la position de transfert (signal PosRight =1). Dans cette position, le pignon est transféré à la station suivante par un préhenseur. Une fois que le chariot est libre (indiqué par le signal Sigfrom4 qui provient de la station suivante), il peut alors être renvoyé à la position initiale.

Dans le cas où il y a initialement un palier dans le pignon (Pignon_sans_palier = 0), le chariot est envoyé à la presse d'extraction pour démonter le palier. Le processus d'extraction est similaire à celui d'insertion. Seul le vérin qui positionne les paliers n'existe pas bien entendu dans cette presse.

2 Analyse prévisionnelle des fautes et obtention des propriétés

Dans cette section, nous allons effectuer l'analyse de trois événements indésirables qui peuvent avoir pour causes des fautes systématiques dues au contrôleur logique.

Ces trois fautes font partie d'un ensemble de neuf fautes que nous avons identifiées pour cette station. La liste complète de ces fautes est donnée dans l'annexe B, où chaque faute est accompagnée d'un chronogramme et d'un extrait de l'AdD correspondant. Nous avons déterminé ces neuf fautes en nous appuyant, d'une part sur les recommandations de sûreté et

de bon fonctionnement de la station 3 fournies par le fabricant, et d'autre part sur les critères de l'AMDEC listés dans le tableau 1.1 du premier chapitre.

Les trois fautes retenues nous paraissent représentatives de cas d'utilisation de portes temporelles et temporisées. Bien sûr, l'ensemble des neuf fautes peut être aussi développé mais, pour le propos illustratif de notre étude de cas, l'analyse des trois fautes nous paraît suffisante.

2.1 Première analyse : Un pignon avec palier est introduit dans la presse d'insertion

Le premier événement indésirable (EI) à considérer peut être ainsi énoncé : «Un pignon avec palier est introduit dans la presse d'insertion». Cette faute est formulée à partir des recommandations du fabricant, qui indique qu'il faut éviter cette situation car le vérin d'insertion peut être endommagé si on veut introduire un palier dans un pignon qui en contient déjà un. L'arbre construit pour cet EI est montré dans la figure 6.6.

Nous considérerons que cet EI ne peut pas être attribué à un composant physique mais qu'il est dû au contrôleur qui génère une commande InFeed erronée. On peut donc appliquer le modèle générique si on considère le contrôleur logique comme étant un composant, selon ce qui a été exposé au chapitre 3. Dans l'analyse prévisionnelle, on ne considère pas cependant de faute physique du contrôleur, ni de faute due à l'environnement. Deux causes possibles, reliées par une porte OU, peuvent être alors à l'origine de cet EI. Soit le signal indiquant la présence d'un pignon (Pignon_sans_palier = 0) est mal transmis au contrôleur, ce qui peut être dû à un mauvais fonctionnement du poste précédent qui délivre ce signal ; c'est alors une faute de commande classique. Comme elle sort du domaine du poste 3, cette faute décrite comme «Information erronée Pignon_sans_palier» ne sera pas développée dans l'analyse. Elle est donc contenue dans un losange.

La deuxième cause est : «Le contrôleur commande l'introduction du pignon avec palier dans la presse d'insertion même avec l'information Pignon_sans_palier = 0 correcte ». C'est une faute systématique, qui signifie que même si le signal qui indique la présence d'un palier est bien transmis et reçu par le contrôleur, ce dernier commande à tort l'introduction du pignon dans la presse d'insertion.

Pour représenter la faute systématique dans l'arbre, il faut utiliser une porte SI qui indique que la condition Pignon_sans_palier = 0 doit être présente pour que la faute se produise en réponse à la séquence ordonnée d'événements exprimée par la porte PAND (front montant de PosIn suivi d'un front montant de InFeed).

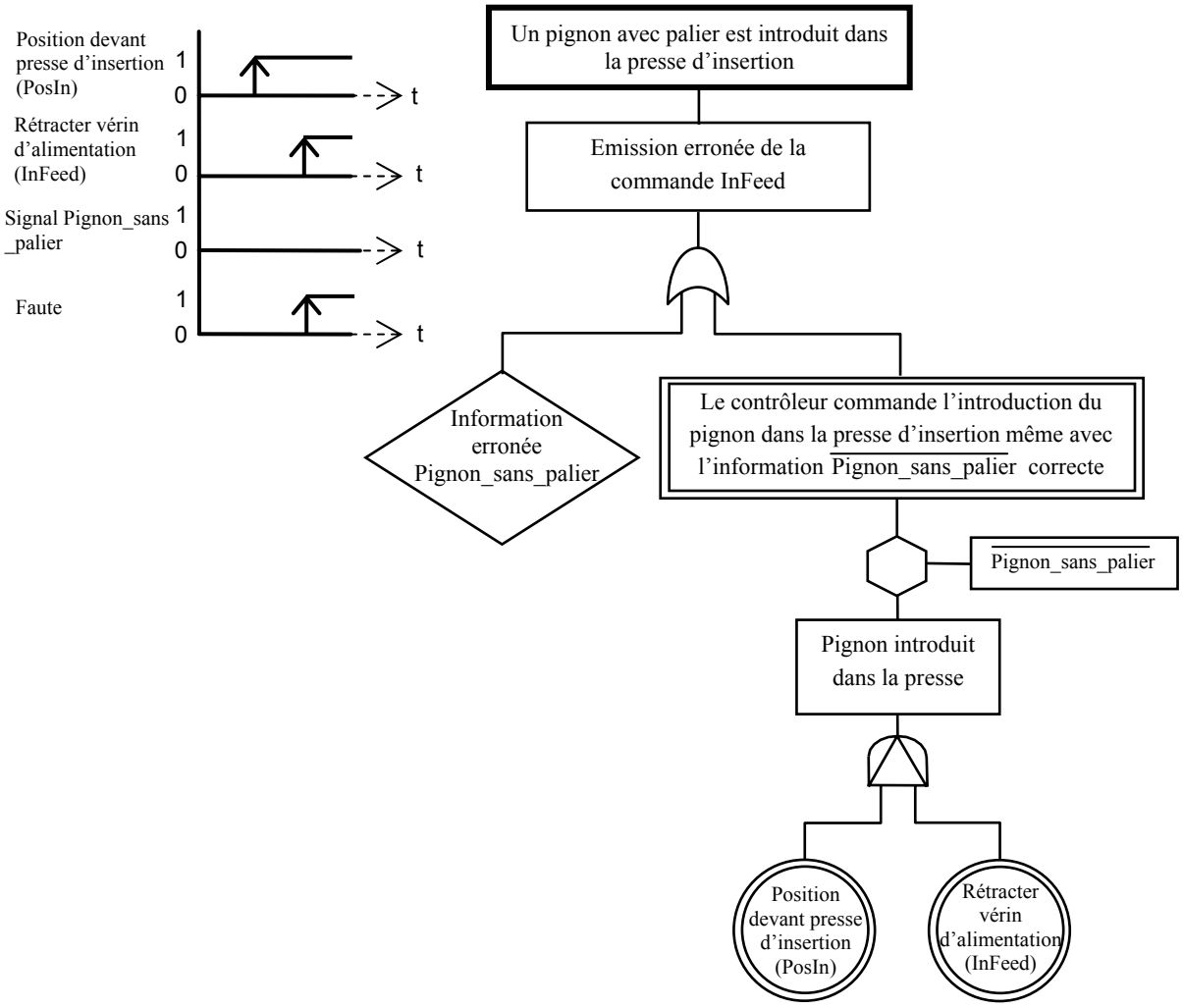


Figure 6.6. L'AdD pour le premier EI

Les modèles formels des propriétés déduits de cette analyse (figure 6.7) sont donc ceux obtenus au chapitre 4 pour la porte PAND auxquels il a été ajouté :

- une condition ¬Pignon_sans_palier dans chacune des propositions de la formule en logique temporelle ;
- une garde identique dans l'action de la première transition de l'automate,

afin de tenir compte de la condition portée par la porte SI située en sortie de la porte PAND.

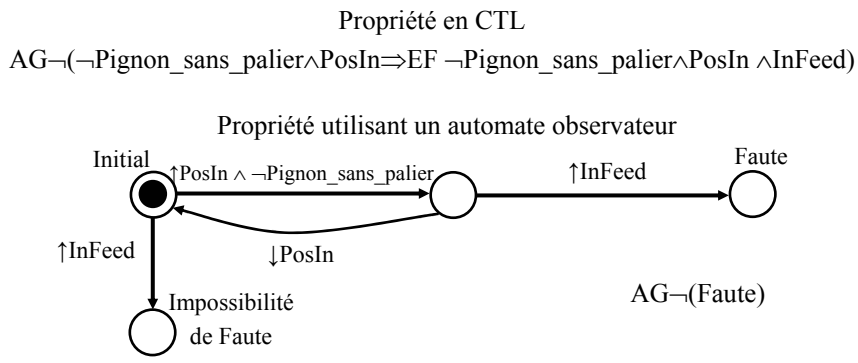


Figure 6.7. Propriété formelle déduite de la première analyse

2.2 Deuxième analyse : faute dans la commande du chariot

L'événement indésirable à analyser ici est : «le chariot est renvoyé à la position initiale avant d'arriver à la position de transfert (PosRight)». L'arbre correspondant est montré dans la figure 6.8.

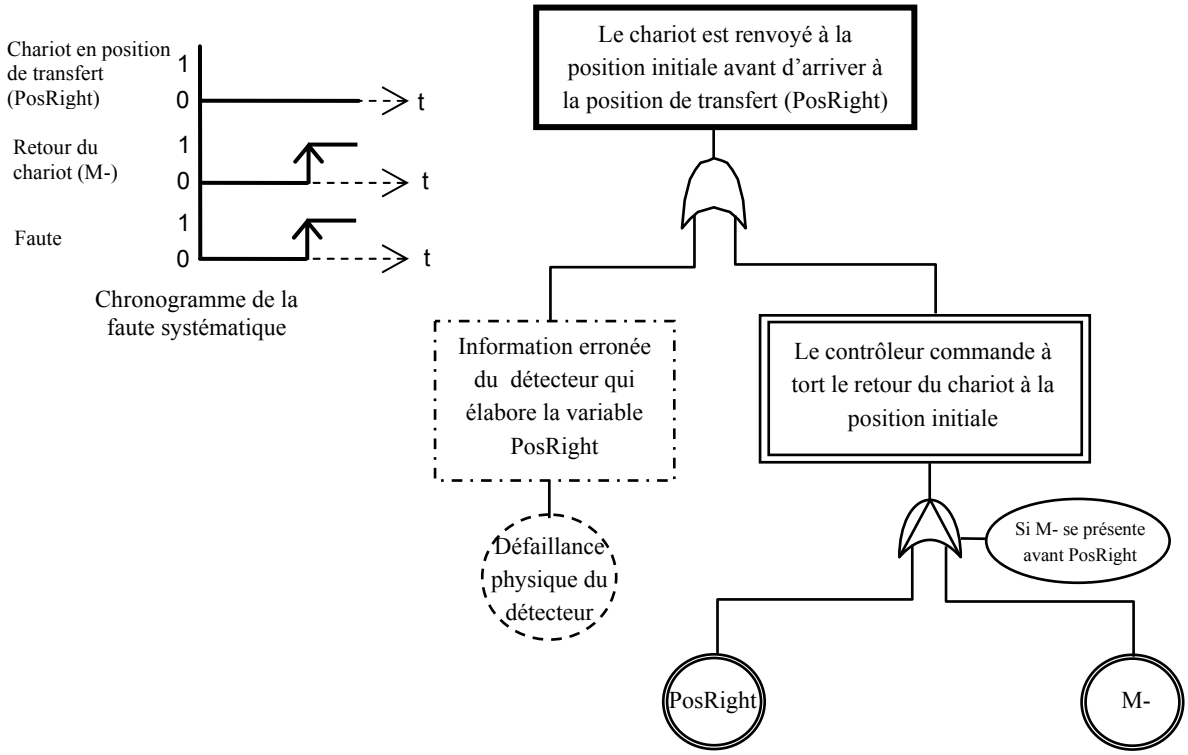


Figure 6.8. L'arbre pour le deuxième EI

On procède de la même manière que pour le cas précédent ; le nouveau modèle générique est une fois de plus appliqué au contrôleur en omettant les fautes primaires et secondaires. Une faute de commande classique ou une faute systématique sont les causes potentielles de l'EI.

La faute de commande classique est due à une défaillance physique du détecteur qui élabore le signal PosRight.

La faute systématique est exprimée à l'aide d'une porte POR à deux entrées dont l'entrée conditionnée est la commande de renvoi du chariot (M-) et l'autre entrée le signal de fin de course droit PosRight. La faute se produit en effet si l'événement ↑M- apparaît avant l'événement ↑PosRight, le fonctionnement normal correspondant à l'ordre inverse de ces événements.

La propriété formelle déduite de cette analyse (figure 6.9) est donc une instance du modèle générique de la porte POR présenté au chapitre 4.

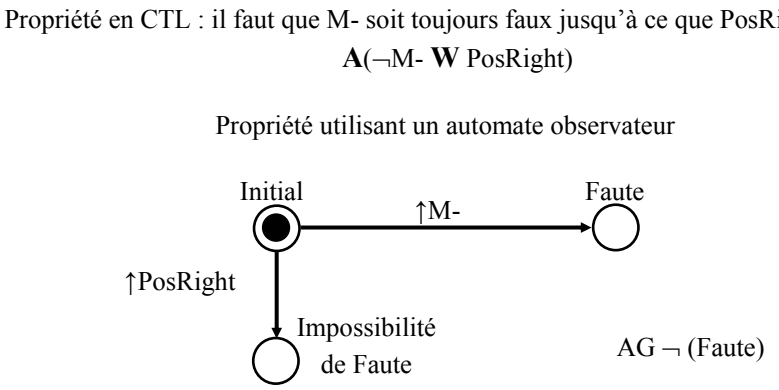


Figure 6.9. Propriété formelle déduite de la deuxième analyse

2.3 Troisième analyse : faute dans la commande du vérin d'alimentation en pignons de la presse d'extraction

Le dernier EI étudié concerne la presse d'extraction et permet de montrer un cas d'utilisation d'une association de portes temporisées. Cet événement indésirable peut être ainsi exprimé : «le contrôleur ne commande pas la rétraction du vérin d'alimentation dans la seconde qui suit l'arrivée du chariot devant la presse d'extraction». Cet événement indésirable n'est donc pas relatif à la sécurité mais traduit plutôt un manque de réactivité de la commande. La Figure 6.10 montre l'arbre détaillant les causes de cet EI.

Les sorties des portes temporisées FORNEXT 1 et WITHIN 1 de cet arbre sont respectivement : «Chariot en position devant la presse d'extraction depuis 1 sec» et « Omission de la commande OutFeed (rétraction du vérin d'alimentation de la presse)». Cette association étant une instance du modèle étudié au chapitre 5, la propriété à vérifier, sous la forme d'un automate temporisé, s'obtient aisément (figure 6.11).

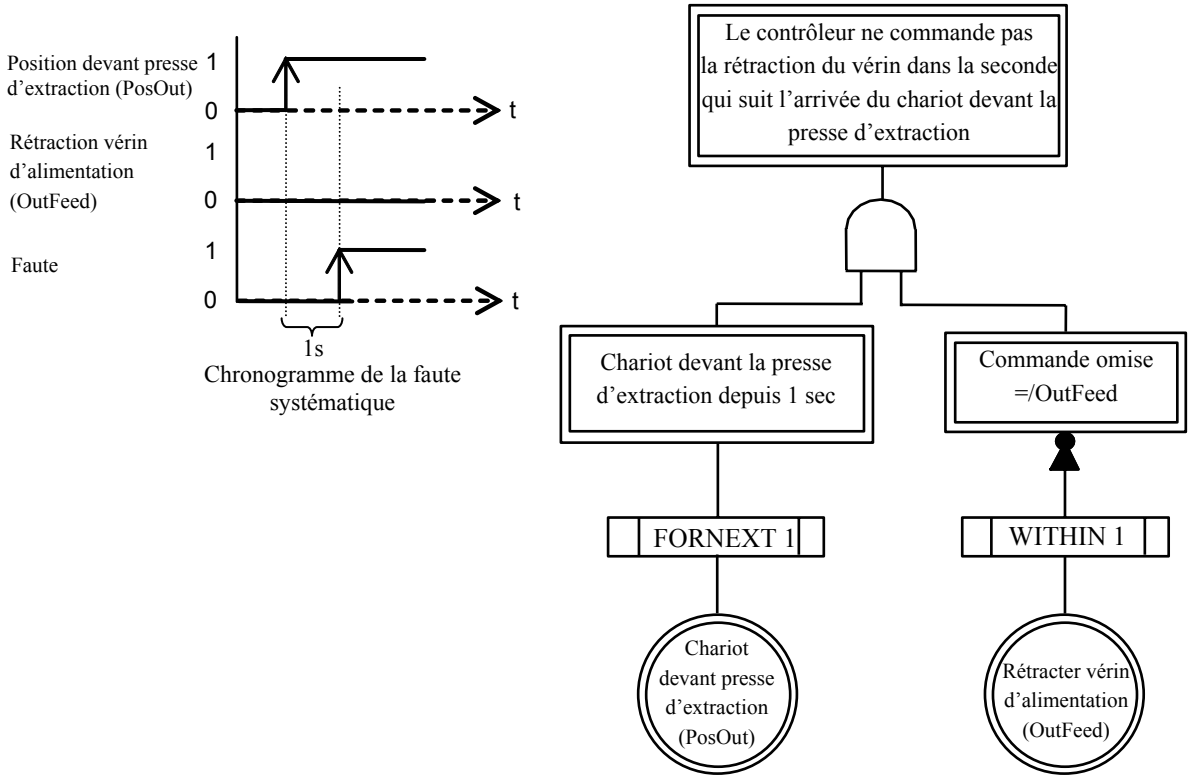


Figure 6.10. AdD pour le troisième EI

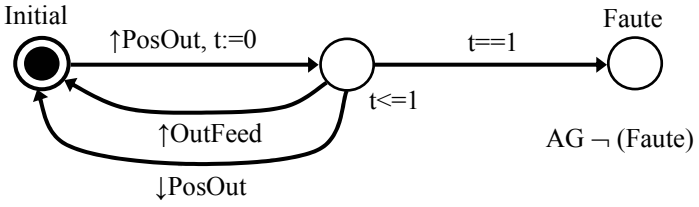


Figure 6.11. Propriété formelle déduite de la troisième analyse

3 Vérification formelle du contrôleur logique

Nous pouvons procéder maintenant à la vérification formelle du contrôleur logique. Pour ce faire, il faut tout d'abord construire un modèle formel de celui-ci. Il a été mentionné au chapitre un, sous-section 4.1, que la modélisation peut concerner, soit seulement le contrôleur logique, ce qui est connu comme une approche «non model-based» ou bien, d'autre part, considérer en plus du modèle du contrôleur, un modèle de la partie opérative, ce qui est connu comme une approche « model-based ». Dans notre cas, nous adoptons cette dernière approche et modéliserons aussi des éléments de la partie opérative, car nous considérons qu'il faut valider non seulement le programme du contrôleur pris isolément, mais aussi son intégration dans le système qu'il doit commander. Le model-checker utilisé est UPPAAL, qui est un outil

de vérification pour des systèmes temps réel et permet l'inclusion des contraintes du temps physique. Nous commençons alors cette section avec une présentation succincte de l'outil UPPAAL, pour ensuite expliquer en détail la construction du modèle de la station 3. La dernière partie présentera des exemples de vérification à l'aide des automates décrivant les propriétés élaborées à la section 2.

3.1 Le model-checker UPPAAL

UPPAAL (Bengtsson et Larsson, 1996 ; Behrmann et al. 2004), a été développé par les universités d'Uppsala (Suède) et d'Aalborg (Danemark). Cet outil peut vérifier et simuler des systèmes dont une modélisation en automates temporisés, synchronisés par des messages ou des variables partagées, est possible. Le formalisme sous-jacent des automates UPPAAL est celui dont la définition est donnée au chapitre 4 sous-section 1.3, et dont on illustre la forme générale dans la figure 6.12.

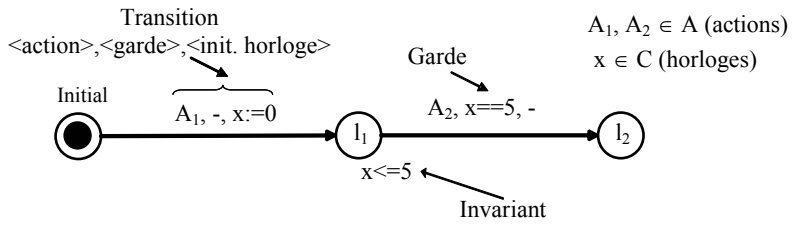


Figure 6.12. Forme générale d'un automate temporisé

UPPAAL permet en plus la communication entre automates temporisés. Par exemple, la figure 6.13 montre deux automates temporisés communicants qui sont synchronisés par 2 messages, *start!* qui désigne une action, et *start?* qui désigne une co-action. En UPPAAL, *message!* et *message?* sont aussi connus comme des canaux de communication. Ce concept d'automates communicants sera utilisé pour modéliser la communication entre contrôleur et partie opérative.

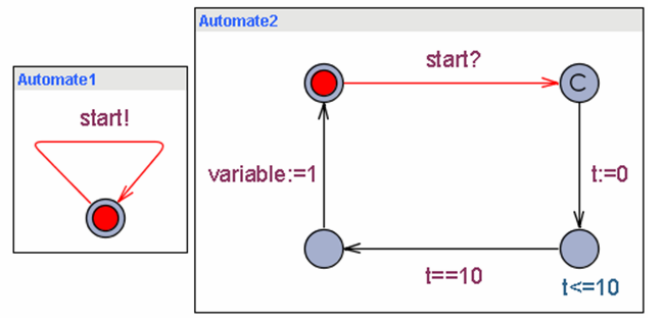


Figure 6.13. Automates temporisés communicants

L'automate qui envoie le message est un automate émetteur (automate 1, dans la figure). Celui qui reçoit le message est l'automate récepteur (automate 2). Il existe également la possibilité d'envoyer un message à plusieurs récepteurs. L'évolution des automates se déroule

de la façon suivante. Le cercle noir (marque) à l'intérieur d'un état indique que cet état est actif. Quand l'état devient inactif, la marque quitte l'état. L'automate 1 émet le signal *start!* quand il franchit la transition. Lorsque *start!* est émis, l'automate 2 peut franchir sa première transition au même instant. Les franchissements de transition portant sur le même canal sont donc synchrones.

Le deuxième état de l'automate 2 est alors actif. Cet état est un état *Committed*, c'est-à-dire un état sans durée. La transition suivante est donc tout de suite franchie. En franchissant une transition, un automate peut assigner des valeurs aux variables et aux horloges ; dans cette deuxième transition l'horloge *t* est initialisée à 0. À partir de ce moment la valeur de l'horloge est incrémentée automatiquement par UPPAAL.

Le prochain état de l'automate 2 porte l'invariant *t<=10*. On rappelle qu'un invariant, dans ce contexte, est une condition sur valeurs d'horloge qui doit être toujours satisfaite quand l'état auquel l'invariant est associé est actif. Dans ce cas précis, l'état devient inactif au plus tard après 10 unités de temps. La prochaine transition porte une garde *t==10*. La garde est la condition qui doit être satisfaite pour franchir la transition. La combinaison d'un invariant dans l'état précédent et d'une garde permet de déterminer le moment précis où la marque quitte l'état. Dans notre cas, c'est exactement 10 unités de temps après l'activation de l'état. Le prochain état ne porte pas d'invariant et un temps indéterminé se déroule jusqu'à ce que la prochaine transition soit franchie. En franchissant cette transition, la variable *variable* est assignée à la valeur 1. Le cycle de l'automate 2 peut alors recommencer.

L'outil UPPAAL comporte trois parties :

- l'éditeur, qui permet de construire les automates de manière graphique ou avec un langage textuel ;
- le simulateur qui permet de visualiser le comportement des automates et donne aussi la possibilité d'observer les valeurs des variables et des horloges ;
- le model-checker, qui peut utiliser des expressions dans une version simplifiée de CTL, qui comprend les opérateurs G (globally, représenté en UPPAAL par le symbole []) et F (futur, représenté en UPPAAL par le symbole <>), ainsi que les quantificateurs de chemins A (always) et E (exist). Les paires supportés par le model-checker sont :
    - A[] φ : φ est toujours vraie ;
    - E<> φ : il est possible (en suivant une des exécutions) d'avoir la proposition φ ;
    - A<> φ : φ sera forcement vraie dans le futur ;
    - E[] φ : il existe une exécution le long de laquelle φ reste toujours vraie.

Dans la figure 6.14 on peut voir représentées les combinaisons acceptables par le model-checker. Si une propriété n'est pas satisfaite par le modèle, le simulateur permet de visualiser un contre-exemple.

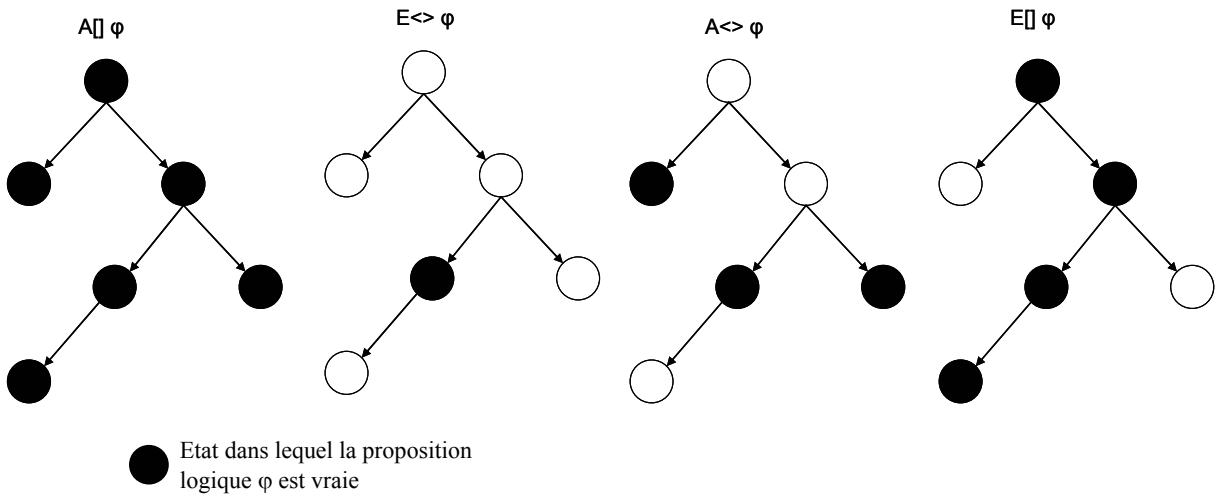


Figure 6.14. Expressions possibles en UPPAAL

3.2 Le modèle du système en UPPAAL

Dans le processus de modélisation, nous avons décidé de nous servir des valeurs et des changements de valeurs (fronts montants/descendants) des entrées/sorties du contrôleur logique. Ainsi, pour chaque signal d'entrée et pour chaque signal de sortie, nous repérons le front montant, le front descendant et la valeur du signal. Dans la figure 6.15, nous montrons cela pour un signal de sortie quelconque. La même démarche est faite sur les entrées.

On verra par la suite que l'utilité de cette démarche réside dans le fait qu'il est possible d'utiliser les fronts montants/descendants comme des canaux de communication, et la valeur de chaque entrée et de chaque sortie, comme une variable partagée, et de cette façon faciliter la synchronisation entre les automates modélisant le contrôleur et ceux modélisant les éléments de la partie opérative. Dans ce qui suit, nous abordons la procédure de modélisation de ces composants.

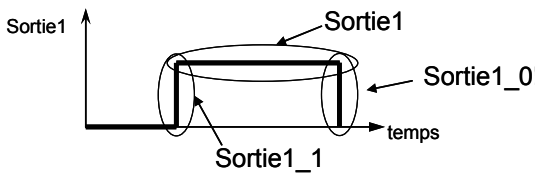
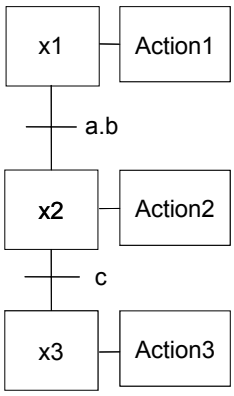


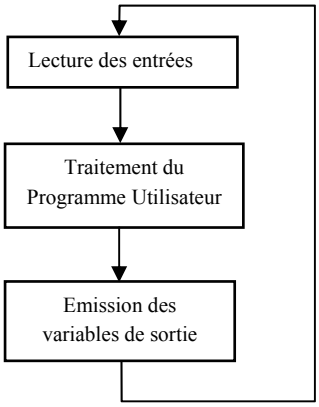
Figure 6.15.Les trois variables déduites d'un signal

3.2.1 Le modèle du contrôleur

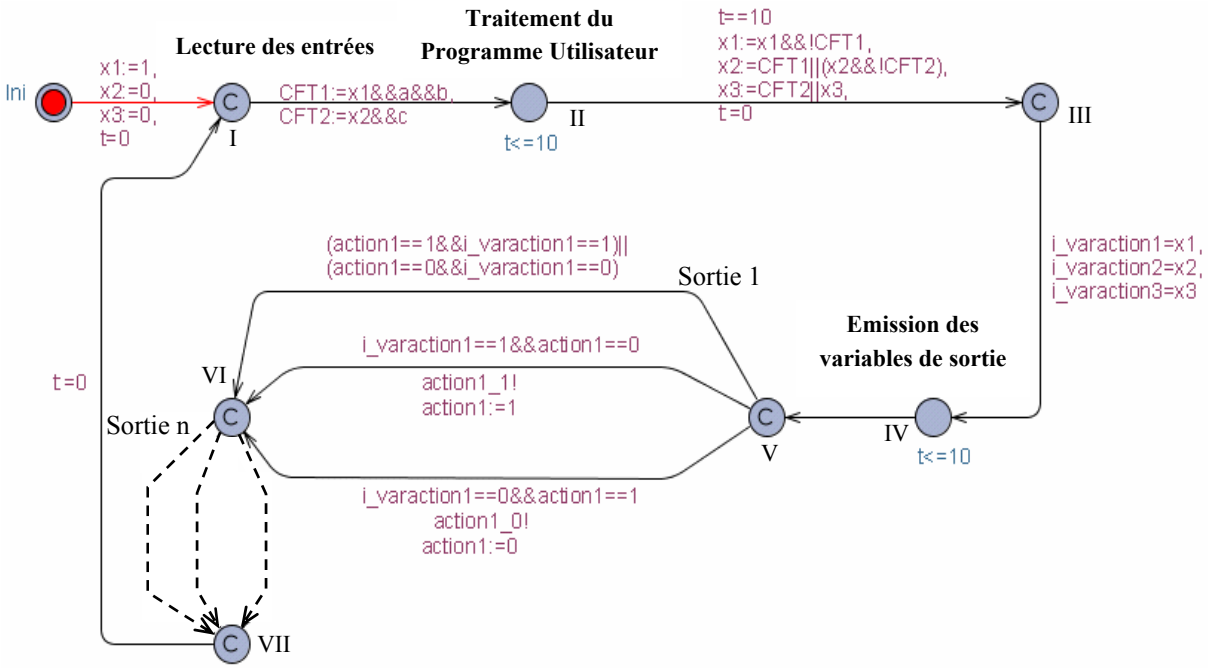
La modélisation du contrôleur logique de la station 3 de la machine Bosch est réalisée à partir du grafcet de spécification. Nous allons expliquer le principe de cette modélisation à l'aide d'un exemple simple illustré en figure 6.16, où on peut voir un grafcet à trois étapes.



(a) Grafcet de commande



(b) Cycle du moniteur temps réel



(c) Automate modélisant le contrôleur

Figure 6.16. Exemple de modélisation du contrôleur

Comme on l'a déjà expliqué, le programme de commande s'exécute sous le contrôle d'un moniteur temps réel, qui réalise un cycle à trois étapes (figure 6.16.b): lecture des entrées, traitement du programme utilisateur, émission des sorties. Ce cycle est modélisé en UPPAAL avec un automate comportant une boucle, qui utilise une horloge pour mesurer le temps de cycle (supposé ici fixe et égal à 10 unités de temps).

La boucle est composée de quatre phases reparties dans sept états (figure 6.16.c) :

1. Lecture des entrées et détermination des transitions franchissables du grafcet (transition entre les états I et II). Les conditions de franchissement des transitions (CFT) sont déterminées de façon algébrique à partir des entrées du contrôleur.
2. Détermination de la nouvelle situation du grafcet (transition entre les états II et III) et calcul des nouvelles valeurs des sorties (transition entre les états III et IV) notées i_varactioni (actioni désignant la valeur de la sortie i au cycle précédent).

3. Emission des sorties. Pour chaque sortie, on introduit 2 états (V et VI pour la sortie correspondant à Action1 par exemple) reliés par trois transitions qui correspondent aux trois cas de figure possibles (pas de changement de la valeur de la sortie, passage de la valeur 0 à la valeur 1 et passage de la valeur 1 à la valeur 0). Dans ces deux derniers cas, on actualise la valeur de la variable partagée et du canal de communication, qui représente un front montant ou descendant, correspondants. Les transitions en pointillés entre les états VI et VII indiquent qu'un traitement similaire doit être fait pour les sorties Action2 et Action3.

4. Réinitialisation de l'horloge qui mesure le temps du cycle (transition entre les états VII et I).

Suivant la démarche qu'on l'on vient de décrire, nous modélisons alors le contrôleur de la station 3 (figure 6.17). On peut noter que 11 sorties sont actualisées à chaque cycle.

3.2.2 Le modèle de la partie opérative

Les éléments de la partie opérative que nous avons modélisé sont :

- les vérins d'insertion de paliers, d'extraction de paliers et d'alimentation des pignons dans les presses (vérins pneumatiques),
- le convoyeur avec le chariot et les détecteurs de position.

On notera que ces modèles ne prennent pas en compte d'éventuelles fautes ou défaillances de ces éléments, l'analyse des fautes d'éléments physiques étant hors du propos.

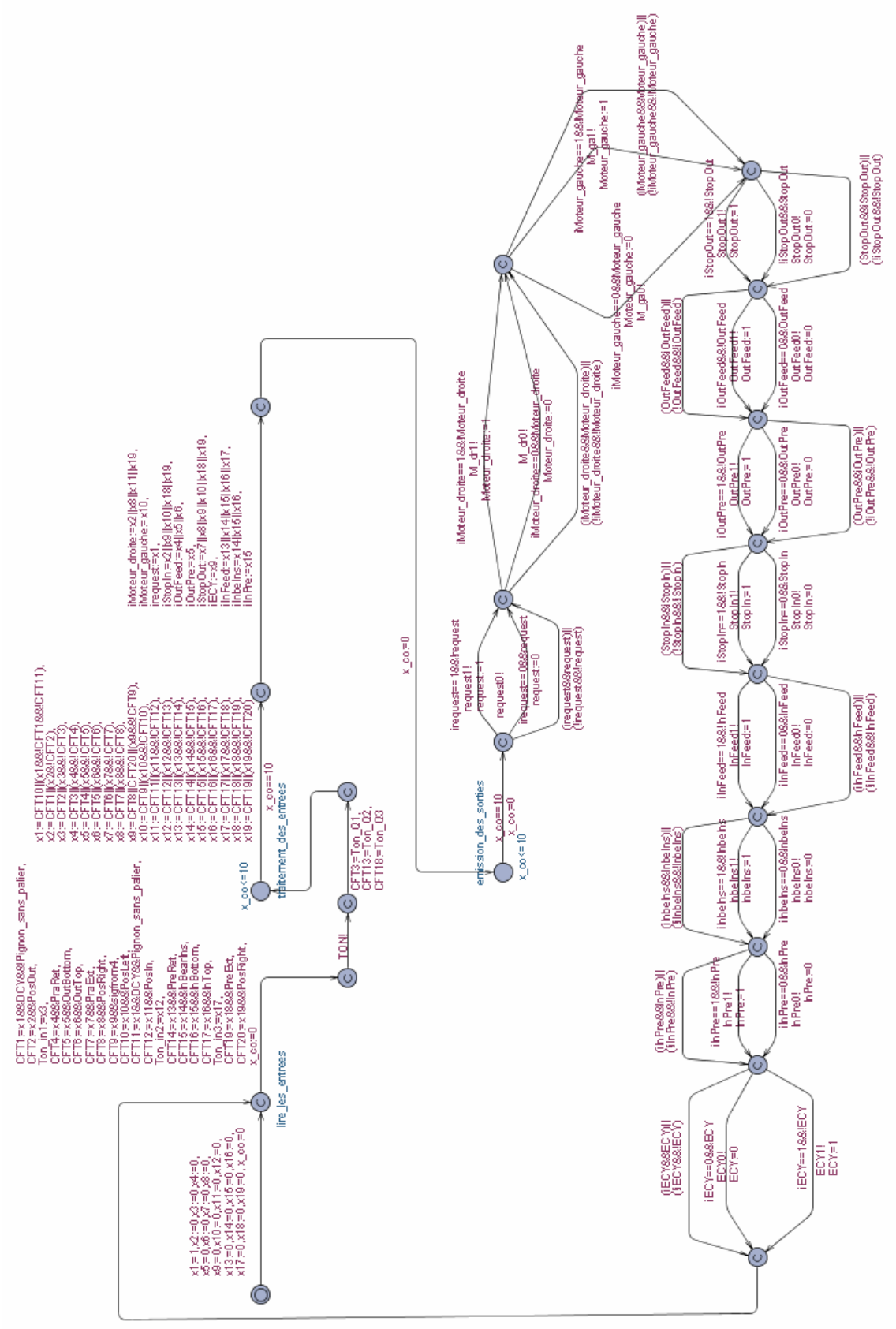


Figure 6.17. Le modèle en UPPAAL du contrôleur de la station 3

Les vérins

Ici, seul le modèle du vérin d'insertion de paliers, **muni de ses détecteurs**, est détaillé (figure 6.18). Les autres vérins sont modélisés de façon similaire.

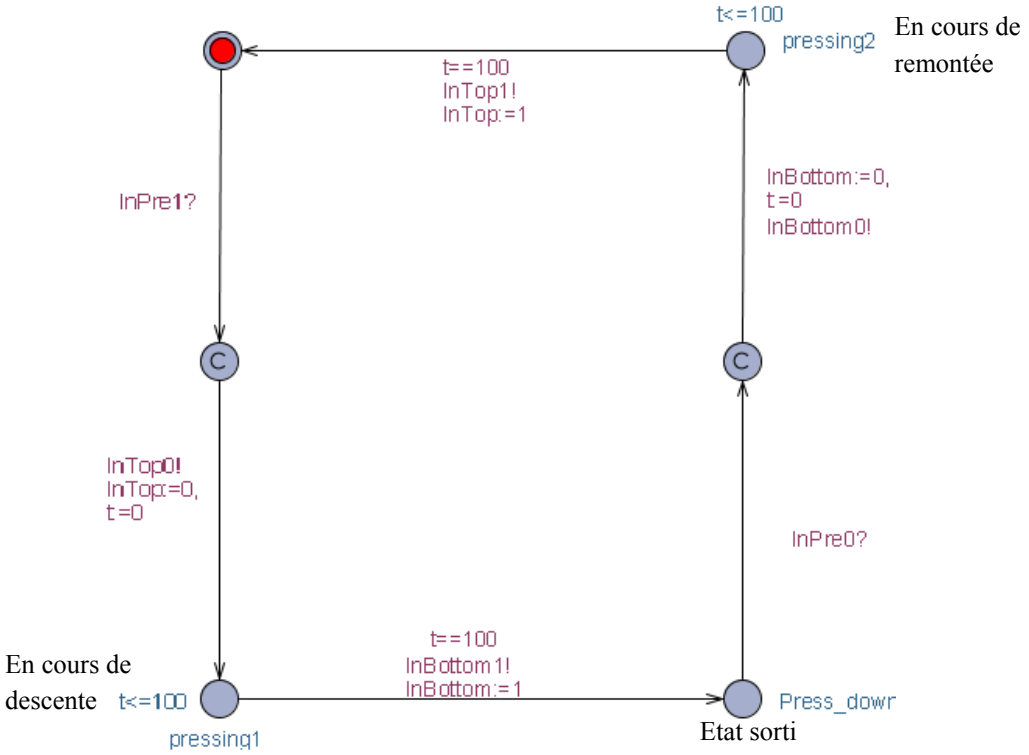


Figure 6.18. Modèle du vérin d'insertion

Dans l'état initial, le vérin est supposé être en position haute. Dès que la commande de sortir le vérin (InPre) est délivrée, l'automate peut évoluer ; la première transition est alors franchie et l'état initial devient inactif. Le prochain état porte l'étiquette «committed» et par conséquent la marque le quitte tout de suite. Cet état est nécessaire parce qu'on ne peut affecter qu'un canal de communication à une transition.

Dans la prochaine transition, le signal d'entrée InTop, qui représente le signal du détecteur de position haute du vérin, passe à 0 et InTop0 (front descendant de InTop) est émis. Lorsque l'état «En cours de descente» est actif, la marque est contrainte d'atteindre 100 unités de temps jusqu'à ce que la prochaine transition puisse être franchie grâce à l'invariant t<=100 et à la condition t==100. Cet état représente le temps qu'il faut au vérin pour être complètement sorti. A l'issue de ce temps, il est possible alors de franchir la prochaine transition qui mène à l'état Press_down. Dans cette transition, l'entrée InBottom, qui représente le signal du détecteur de position basse, passe à 1 et InBottom1 (front montant de InBottom) est émis. Maintenant l'automate attend que la sortie InPre, commande de sortie du vérin, passe de nouveau à 0 (attente de InPre0) et le vérin peut alors rentrer. Dès que cela est fait, la marque passe dans l'état étiqueté comme *committed* et ensuite dans l'état *pressing2*. Lors de la transition, le signal indiquant la position basse InBottom passe de nouveau à 0 et l'horloge est

réinitialisée. Dans le nouvel état, on compte 100 unités de temps pour modéliser la remontée du vérin (invariant t<=100), pour ensuite franchir la dernière transition et rejoindre de nouveau l'état initial.

Le convoyeur

L'automate correspondant au convoyeur est montré en figure 6.19. Nous avons modélisé :

- les positions stables du chariot. Ces positions correspondent à quatre états : cap_Posleft (position initiale à gauche), cap_PosIn (position devant la presse d'insertion), cap_PosOut (position devant la presse d'extraction), cap_posRight (position à droite) ;
- entre deux positions, le comportement du convoyeur est exprimé avec deux états, un état qui porte l'étiquette *committed* et l'autre portant un invariant qui représente le temps qui met le convoyeur pour parcourir la distance entre ces deux positions. Par exemple, pour aller de la position gauche (cap_Posleft) à la position devant la presse d'insertion (cap_PosIn), le convoyeur met 500ms.

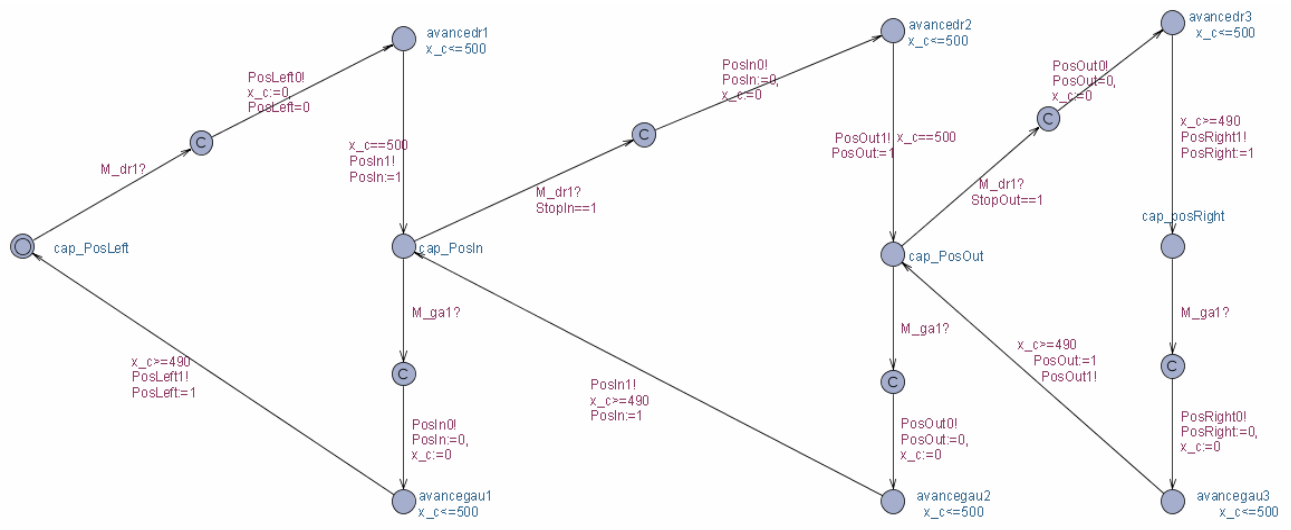


Figure 6.19. Modèle du convoyeur

3.3 Vérification avec UPPAAL

Dans ce qui suit, nous allons présenter la vérification en UPPAAL de chacune des trois propriétés élaborées en section 2 de ce chapitre. Dans tous les cas, nous avons ajouté au modèle du système (contrôleur + partie opérative), l'automate observateur de la faute en question selon la syntaxe du model-checker. La propriété à vérifier est alors que l'état (les états) de cet automate représentatif(s) de cette faute n'est pas (ne sont pas) atteint(s) et s'énonce :

A[] not (nom de l'automate).Faute

ce qui veut dire que l'on n'aura jamais (A[] not) l'automate observateur dans un état où *Faute* est vraie. Nous allons voir que pour chaque cas analysé, le model-checker indique que la propriété n'est pas vérifiée, ce qui veut dire qu'il existe la défaillance dans le modèle du système. Nous demandons au model-checker de fournir un contre-exemple et nous obtenons ainsi la trace qui mène l'automate observateur à *Error*. Pour chaque analyse nous allons discuter où se trouve l'origine de la défaillance et comment le corriger.

3.3.1 Première propriété à prouver : le contrôleur ne commande jamais l'introduction d'un pignon avec palier dans la presse d'insertion

L'automate observateur utilisé pour cette preuve est nommé *Verifdef1* et est représenté à la figure 6.20 en respectant la syntaxe UPPAAL. On rappelle en particulier que *Variable1* et *Variable0* désignent respectivement les fronts montant et descendant de la variable *Variable* ; le complément de cette variable est noté *!Variable*. Nous voulons vérifier que l'état *Error* de cet automate n'est jamais atteint ; la propriété à prouver est donc :

A[] not Verifdef1.Error

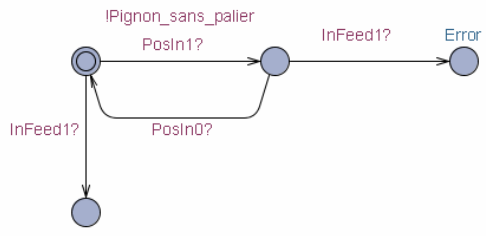


Figure 6.20. Automate observateur selon la syntaxe UPPAAL pour la première analyse

Pour s'assurer que cet automate permet la détection de la faute du contrôleur redoutée, nous introduisons une faute de conception dans le grafcet de la figure 6.5 en **permutant les réceptivités associées aux transitions** T_1 et T_{11} (figure 6.21). Cette faute est détectée par le model-checker (l'état *Error* de l'automate *Verifdef1* peut être atteint) et identifiable à partir du contre-exemple fourni. Il est alors possible de corriger cette faute en nous ramenant au grafcet initial ; le modèle UPPAAL de ce dernier grafcet satisfait par contre la propriété étudiée, ce qui prouve son exactitude vis-à-vis de cette spécification.

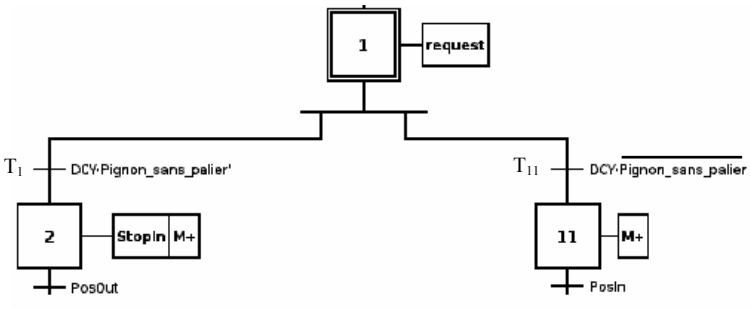


Figure 6.21. Extrait du grafcet erroné utilisé pour la première preuve

3.3.2 Deuxième propriété à prouver : le contrôleur ne commande jamais le retour du chariot en position initiale avant qu'il ne soit arrivé à la position de transfert

L'automate observateur utilisé pour cette preuve est nommé *Verifdef2* et est représenté à la figure 6.22 en respectant la syntaxe UPPAAL. La propriété à prouver est:

A[] not Verifdef2.Error

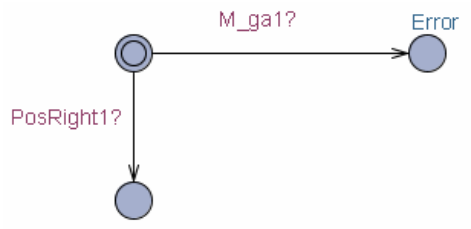


Figure 6.22. Automate observateur selon la syntaxe UPPAAL pour la seconde analyse

La pertinence de cet automate pour la détection de la faute du contrôleur redoutée est testée en introduisant dans le grafcet de la figure 6.5 une autre faute de conception (figure 6.22). Cette faute consiste à **associer une action supplémentaire** M- à l'étape 5 (figure 6.23). Cette faute est détectée par le model-checker (l'état *Error* de l'automate *Verifdef2* peut être atteint) et identifiable à partir du contre-exemple fourni. Le grafcet présenté à la figure 6.5 satisfait par contre la propriété étudiée.

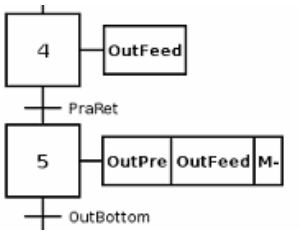


Figure 6.23. Extrait du grafcet erroné pour la seconde preuve

3.3.3 Troisième propriété à prouver : le contrôleur ne commande jamais la rétraction du vérin d'alimentation plus d'une seconde après l'arrivée du chariot devant la presse

L'automate observateur pour la troisième analyse (figure 6.24) est nommé *Verifdef3* et la propriété à prouver est :

A[] not Verifdef3.Error

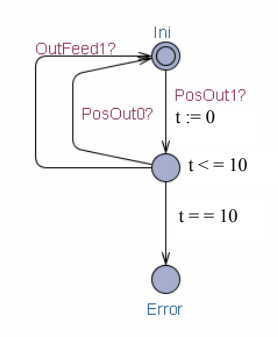


Figure 6.24. Automate observateur selon la syntaxe UPPAAL pour la troisième analyse

Pour évaluer la pertinence de cet automate à des fins de détection de la faute du contrôleur redoutée, nous introduisons dans le grafcet de la figure 6.5 une faute de conception (figure 6.24) qui consiste à **remplacer une action par une autre ;** OutFeed a été remplacée par OutPre à l'étape 4. Cette faute est détectée par le model-checker (l'état *Error* de l'automate *Verifdef3* peut être atteint) et identifiable à partir du contre-exemple fourni. Le grafcet original présentée à la figure 6.5 satisfait par contre la propriété étudiée.

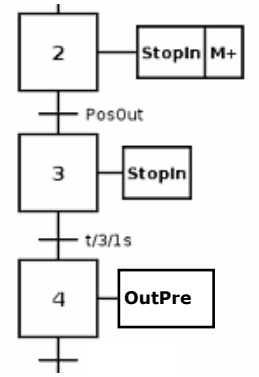


Figure 6.25. Extrait du grafcet erroné pour la troisième preuve

4 Conclusion

Cette étude de cas nous a permis de montrer l'intérêt de notre méthode qui consiste à intégrer les fautes systématiques dans l'analyse prévisionnelle des fautes et à élaborer les propriétés formelles à vérifier à partir de cette analyse. Trois fautes de conception, se situant tant au niveau des réceptivités qu'à celui des actions du grafcet, ont pu être détectées par notre approche.

Conclusion & Perspectives

L'objectif de ce travail de thèse était de faciliter l'obtention des propriétés formelles des contrôleurs logiques en vue de leur vérification formelle par model-checking. Pour ce faire, tout au long de ce mémoire, une méthode a été proposée et développée. L'intérêt et l'originalité de cette méthode résident dans l'utilisation de la technique d'analyse par Arbre des Défaillances (AdD), ce qui nous permet de faire un lien entre les méthodes de prévision de fautes et les méthodes formelles de détection de fautes. Ceci permet d'envisager une utilisation plus aisée des méthodes formelles de vérification en milieu industriel.

L'état de l'art des principaux travaux scientifiques portant sur les évolutions et améliorations de la méthode de l'AdD nous a permis de constater que, même s'il existait des résultats intéressants en matière de prise en compte de la dynamique des systèmes lors de cette analyse, ces travaux se focalisaient sur les défaillances de composants et systèmes physiques et non sur les fautes systématiques de contrôleur logiques.

Nous avons alors fait quatre propositions qui, de fait, constituent les principaux apports de cette thèse :

— deux propositions de nature méthodologique :
  - proposition d'un nouveau modèle générique de la faute de composant, qui intègre le concept de faute systématique ;
  - expression des fautes systématiques avec un vocabulaire de portes statiques, temporelles et temporisées ;

— deux de nature formelle :
  - modélisation formelle du comportement des portes du vocabulaire retenu, et obtention de propriétés formelles à vérifier à partir des modèles formels des portes statiques et temporelles ;
  - obtention des modèles formels des comportements d'associations cohérentes de portes et dérivation de propriétés formelles à partir de ces modèles.

Une étude de cas basée sur un système réel a permis de montrer l'intérêt de notre méthode, car plusieurs fautes de conception, se situant tant au niveau des réceptivités qu'à celui des actions d'un grafcet modélisant un contrôleur logique, ont pu être détectées à l'aide d'arbres de défaillances utilisant les trois types de portes définis.

Plusieurs perspectives peuvent être envisagées à partir de ce travail de thèse. En premier lieu, la convergence entre les résultats présentés dans ce mémoire et ceux obtenus au LURPA dans le domaine de la simplification d'un AdD dynamique à l'aide d'une représentation algébrique des fautes (Merle et Roussel, 2007) devrait être recherchée.

Il est d'autre part indispensable de traiter d'autres études de cas, afin d'évaluer la complétude (ou la non-complétude) du vocabulaire de portes choisi. Ces nouvelles études de cas permettront aussi de considérer d'autres types d'associations de portes ainsi que des associations plus importantes que celles que nous avons étudiées.

Il est également envisageable d'automatiser la méthode d'obtention des propriétés et de l'intégrer dans la mesure du possible dans un logiciel commercial d'édition d'arbre de défaillances qui permettrait l'utilisation de portes dynamiques. Ce travail aura besoin d'une collaboration industrielle bien entendu.

Nous avons basé notre formulation de propriétés sur l'analyse du comportement dysfonctionnel des contrôleurs logiques, ce qui découle de l'utilisation de l'analyse par AdD. Il serait utile de s'intéresser à l'obtention des propriétés relatives au comportement fonctionnel du contrôleur (propriétés de vivacité, d'équité) ; bien sûr, l'AdD ne sera probablement plus utilisé dans cette approche.

Enfin, les résultats obtenus en matière de formalisation du comportement des portes dynamiques et des associations de portes pourraient être utilisés pour l'AdD de systèmes physiques dont les défaillances sont fonction de l'ordre d'événements ou du temps physique. Les expressions formelles obtenues semblent en effet des éléments de base intéressants pour une approche formelle du diagnostic et de la maintenance des systèmes modélisables par des systèmes à événements discrets.

Bibliographie

Alur, R. et D.L. Dill (1994). A Theory of Timed Automata. *Theoretical Computer Science*, **Vol. 126**, pp. 183-235.

Andrews, J. (2002). Fault Tree Analysis – Common Misconceptions. *Proceedings of the 20th International System Safety Conference*, pp. 401-410, August 5-9, Denver, Colorado, USA.

Arboost Technologies (2006). http://www.arboost.com

Barragan, I. et J.M. Faure (2005). From fault-tree analysis to model-checking of controllers. *Proceedings of 16th IFAC World Congress*, CDROM paper n° 4596, Praha (CZ), July 4-8.

Barragan, I., M. Roth, et J.M. Faure (2006). Obtaining temporal and timed properties of logic controllers from fault tree analysis. *12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006*, Saint-Etienne, France, May, 17-19, pp. 243-248.

Barragan, I., J.M. Faure et Y. Papadopoulos (2006). Including systematic faults into fault-tree analysis. *Proceedings of the 6th IFAC Symposium on Fault Detection, Supervision et Safety of Technical Processes (SAFEPROCESS 2006)*, Aug 30-Sep 1, Beijing, P. R. China, pp. 811-816.

Behrmann, G., A. David et K.G. Larsen (2004). *A Tutorial on UPPAAL*. Department of Computer Science, Aalborg University, Denmark.

Bengtsson, J. et F. Larsson (1996). *UPPAAL a tool for automatic verification of real time systems*. Master of science thesis, Uppsala University, Department of Computer Science.

Bobbio, A. et D. Codetta (2004). Parametric Fault Trees with Dynamic Gates and Repair Boxes. *In: Proceedings of Reliability and Maintainability Symposium RAMS2004*, pp. 459-465.

Bozzano, M. et A. Villafiorita (2003). Integrating Fault Tree Analysis with Event Ordering Information. *In Proceedings of ESREL 2003*, pp. 247-254, June 15-18, Maastricht, The Netherlands.

Cabarbaye, A. (2001). Simulation dynamique des arbres d'événements. *Congrès Qualita 2001*, Annecy.

Cha, S., H. Son, J. Yoo, E. Jee, P.H. Seong (2003). Systematic evaluation of fault trees using real-time model checker UPPAAL. *Reliability Engineering and System Safety 82*, pp. 11-20.

Cimatti, A., E. Clarke et M. Roveri (2000). NuSMV: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4), pp. 410–425.

Commission Electrotechnique International CEI 61025 (1990). *Analyse par Arbre de Pannes*.

Commission Electrotechnique International CEI 60812 (2006). *Techniques d'analyse de la fiabilité du système* – Procédure d'analyse des modes de défaillance, de leurs effets et de leur criticité (AMDEC).

De Smet, O. et O. Rossi (2002). Verification of a controller for a flexible manufacturing line written in Ladder Diagram via model-checking. *21th American Control Conference*, ACC'02, Anchorage (USA), CDRom paper N°734, pp. 4147-4152, May 2002.

Dugan, J.B., B. Venkataraman et R. Gulati (1997). DIFtree : A software package for the analysis of dynamic fault tree models. *In Proc. Reliability and Maintainability Symposium*, January 1997.

Dutuit, Y., A. Rauzy, E. Niel et E. Craye (2002). *Maîtrise des risques et sûreté de fonctionnement des systèmes de production*. Chapitre 7. Approche événementielle : l'arbre des défaillance. Lavoisier. Hermes Sciences publications. ISBN 2-7462-0402-9.

Faure, J.M. et J.J. Lesage (2001). Methods for safe control systems design and implementation. *INCOM'2001*, CDRom paper, 6 pages, September 20-22, Vienna, Austria.

Filkorn, T. (1999). Formal verification of PLC-programs. *Proceedings of the 14th IFAC*, July 5-9, Beijing, P.R. China.

Frey, G. et L. Litz (2000). Formal methods in PLC programming. *Proceedings of the IEEE Conference on Systems, Man and Cybernetics*, Nashville, USA, Octobre 8-11, pp. 2431-2436.

Gaudré, T., M. Bouissou, P. Chaussis et P. Nonclercq (2001). Application de l'outil KB3-IFAST (ARALIA inside) de génération automatique d'Arbres de Défaillances à un système électronique embarqué automobile. *2ème Conférence Annuelle d'Ingénierie Système*. Toulouse, 26-28 juin 2001.

Gourcuff, V., O. De Smet, J.M. Faure (2006). Efficient representation for formal verification of PLC programs. *8th International Workshop On Discrete Event Systems (WODES\'06)*, Ann Arbor(USA), July 2006, pp. 182-187.

Henry, S. et J.M. Faure (2003). Elaboration of invariant safety properties from fault-tree analysis. *Proceedings of IMACS-IEEE "CESA'03*, 6 pages, July 9-12, Lille, France.

INERIS (2003). *Outils d'analyse des risques générés par une installation industrielle*. Direction des Risques Accidentels. Mai 2003.

International Electrotechnical Committee (1993). IEC 61131, *Programmable controllers, Programming languages*.

Isograph Ltd (1986). http://www.isograph-software.com/index.htm

Julliand, J. (2003). Automates finis et applications. Cours de DEA I.A.P. Obtenu dans http://lifc.univ-fcomte.fr/~julliand/CoursAutomates.pdf

Kehren, C. et C. Seguin (2003). Evaluation qualitative de systèmes physiques pour la sûreté de Fonctionnement. *FAC 03*. Toulouse.

Klein, S. (2001). *A case study in design and formal verification of control algorithms using Interpreted Petri Nets and SFC*. Mémoire de recherche de DEA de l'Ecole Normale Supérieure de Cachan.

Lampérière-Couffin, S., O. Rossi, J.M. Roussel et J.J. Lesage (1999). Formal validation of PLC programs : a survey. *European Control Conference 1999, ECC'99*, Karlsruhe (Germany), CD-ROM paper n°741, 6 pages, Sept. 1999.

Laprie, J.C., B. Courtois, M.C. Gaudel, D. Powell (1989). *Sûreté de fonctionnement des systèmes informatiques*. Bordas, Paris, 1989. ISBN 2-04-016942-3.

Laprie, J.C. (2004). Sûreté de fonctionnement des systèmes : concepts de base et terminologie. *Rapport LAAS N° 04520. Revue de l'Electricité et de l'Electronique*, N°11, pp.95-105, Décembre 2004.

Limnios, N. (2005). *Arbres de défaillances*. 2e édition. Lavoisier. Hermes Sciences publications. ISBN 2-7462-1067-3.

Machado, J.M., B. Denis, J.J. Lesage, J.M. Faure et J.C. Ferreira Da Silva (2003). Increasing the efficiency of PLC Program Verification using a plant model. *IEPM'03: 6th International Conference on Industrial Engineering and Production Management*, CDROM proceedings, paper ormhm-4, 10 pages, Mai 26-28, Porto, Portugal.

Manian, R., J.B. Dugan, D. Coppit et K.J. Sullivan (1998). Combining various solution techniques for dynamic Fault Tree Analysis of computer systems. *In Proceedings Third IEEE International High-Assurance Systems Engineering Symposium*, pages 21–28, Nov. 1998.

Manian, R., D. Coppit, K.J. Sullivan et J.B. Dugan (1999). Bridging the gap between Fault Tree Analysis Modeling Tools and the Systems being Modeled. *In Annual Reliability and Maintainability Symposium*, 1999.

Merle, G., J.M. Roussel (2007). Algebraic modelling of Fault Trees with Priority AND gates. 1st IFAC Workshop on Dependable Control of Discrete Systems, DCDS'07. Cachan, France. June 13-15.

Metge, S. (1996). *Connaissez-vous l'AEEL* ? Note technique CENA/NT96/SdF/S. www.tls.cena.fr/divisions/SDF/Public/DocsPS/Vulgarisation/Les_AEELs.ps

Palshikar, G.K. (2003). Temporal Fault Trees. *Information and Software Technology*, n° 44, p137-150.

Papadopoulos, Y., J. McDermid, R. Sasse et G. Heiner (2001). Analysis and synthesis of the behaviour of complex programmable electronic system in conditions of failure. *Reliability Engineering and System Safety 71*, pp. 229-247.

Papadopoulos, Y. et M. Maruhn (2001). Model-based automated synthesis of fault trees from Matlab-Simulink models. *DSN'01, Int'l Conf. on Dependable Systems and Networks*. pp. 77–82. Götenborg.

Point, G. et A. Rauzy (1999). AltaRica, constraint automata as a description language. *APII-JESA*. Volume 33, n° 8-9, pp. 1033-1052.

Rauzy, A. (1993). New Algorithms for Fault Trees Analysis. *Reliability Engineering and System Safety*, 40(3), pp. 203–211.

Rauzy, A. et Y. Dutuit (1997). Exact and Truncated Computations of Prime Implicants of Coherent and Non-Coherent Fault Trees within Aralia. *Reliability Engineering and System Safety*, 58(2), pp. 127–144.

Relex Software Corporation (1986). http://www.relex.com

Rossi, O. (2003). *Validation formelle de programmes Ladder Diagram pour Automates Programmables industriels*. Thèse de doctorat. Ecole Normale Supérieur de Cachan.

Roussel, J.M. et B. Denis (2002). Safety properties verification of ladder diagram programs. *Journal Européen des Systèmes Automatisés*, 36(7), pp. 905-917.

Rushby, J. (2000). Theorem proving for verification. *In Franck Cassez, editor, Modelling and Verification of Parallel Processes: MoVEP 2k*, Nantes, France, June 2000.

Schäfer, A. (2003). Combining real-time model-checking and fault-tree analysis, *LNCS 2805*, pp 522–541.

Schellhorn, G., A. Thums et W. Reif (2002). Formal Fault Tree Semantics. *In: Proceedings of The Sixth World Conference on Integrated Design & Process Technology*. Pasadena, CA, June.

Schnoebelen, P., B. Bérard, M. Bidoit, F. Laroussinie et A. Petit (1999). *Vérification de logiciels : Techniques et outils du model-checking*. Vuibert, Paris, 1999. ISBN 2-7117-8646-3.

Sullivan, K., J.B. Dugan, D. Coppit (1999). The Galileo Fault Tree Analysis Tool. *In Proc. Symposium on Fault-Tolerant Computing (FTCS 1999)*, pp. 232–235. IEEE, 1999.

Thums, A. et G. Schellhorn (2003). Model Checking FTA. *World Congress on Formal Methods FME, LNCS 2805*, pp. 739-757.

Timking Electronics, 2000. *Technical Testbed Documentation*. http://www.personal.engin.umich.edu/~tilbury/testbed/

US Nuclear Regulatory Commission (1981). *Fault Tree Handbook*. Technical Report NUREG-0492, Washington, DC.

Vesely, W., J. Dugan, J. Fragola, J. Minarick et J. Railsback (2002). *Fault Tree Handbook with Aerospace Applications*. NASA Office of Safety and Mission Assurance. Washington, D.C.

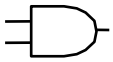
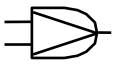
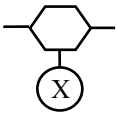
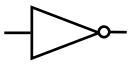
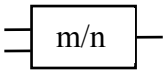
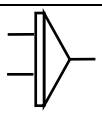
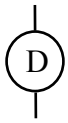
Walker, M. et Y. Papadopoulos (2006). PANDORA : The Time Of Priority-AND Gates. *12th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 2006*, Saint-Etienne, France, May, 17-19

Xiang, J (2005). *FTA and Formal Methods for requirements engineering*. Japan Advanced Institute of Science and Technologie. PhD Thesis. Sept 2005.

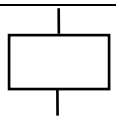
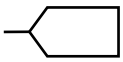
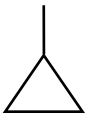
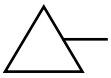
Annexe A.

Conventions de présentation de l'AdD couramment utilisées

1 Symboles des portes

Symbole	Porte	Description
	ET	L'événement de sortie se produit si tous les événements d'entrée surviennent.
	OU	L'événement de sortie se produit si un des événements d'entrée survient.
	ET priorité	L'événement de sortie se produit seulement si ses entrées se produisent dans un ordre spécifique, de gauche à droite normalement.
	OU exclusif	L'événement de sortie se produit si une entrée et une seule existe.
	SI (Conditionnelle)	L'événement de sortie est généré par un seul événement d'entrée, qui survient tandis que la condition X est présente.
	NON	L'événement de sortie est généré lorsque l'événement d'entrée ne se produit pas.
	COMBINAISON	L'événement de sortie survient si m parmi les n événements d'entrée surviennent.
	MATRICIELLE	L'événement de sortie est généré à partir de certaines combinaisons des entrées qui ne sont pas pour le moment explicitées.
	DELAI	L'événement de sortie est générée après un délai sur l'entrée, qui doit être présente pendant ce délai.

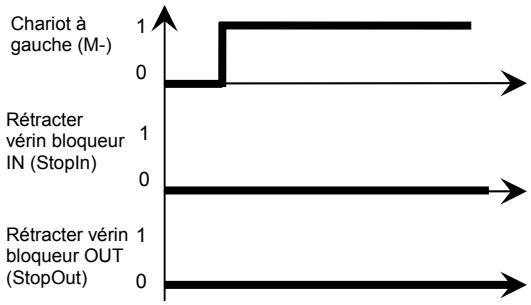
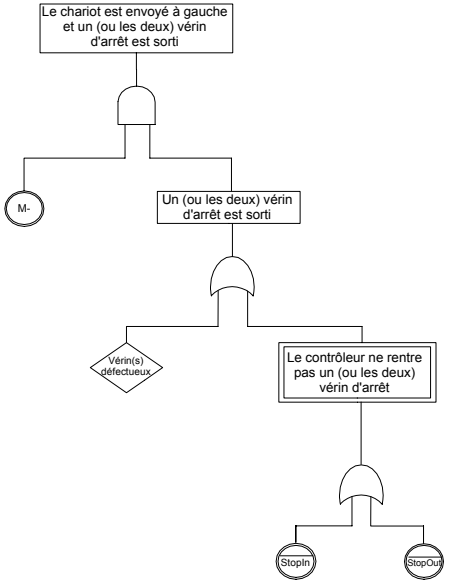
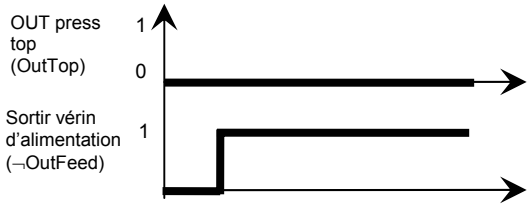
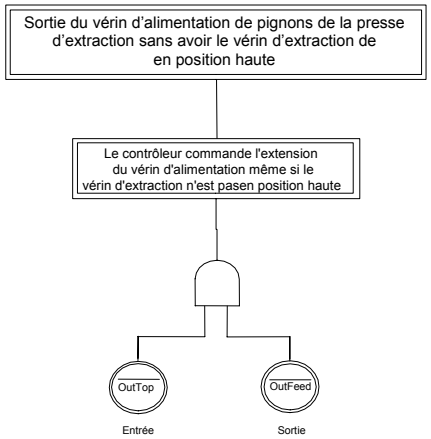
2 Symboles d'événements

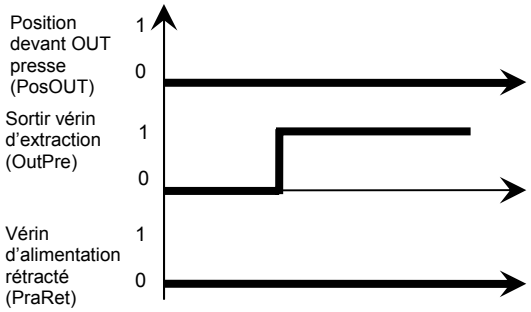
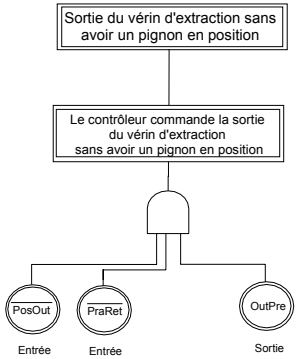
Symbole	Fonction	Description
	Evénement intermédiaire ou sommet	Symbole pour introduire le nom ou la description d'un événement intermédiaire ou sommet.
	Evénement élémentaire	Evénement qui ne peut pas être divisé en nouveaux éléments.
	Evénement non développé	Evénement de base non élémentaire, mais qui n'est plus développé.
	Evénement de condition	La condition n'est pas une défaillance ou un événement externe, mais quelque chose à faire avec l'environnement.
	Maison	Evénement qui est normalement attendu.
	Renvoi IN	La partie de l'arbre qui devrait suivre est développée dans une partie séparée de l'arbre.
	Renvoi OUT	Indique que la portion de l'arbre doit être attachée à son correspondant renvoi IN.

Annexe B.

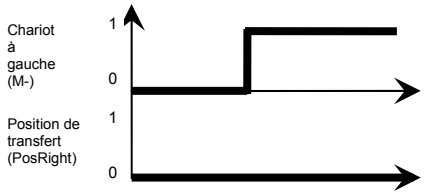
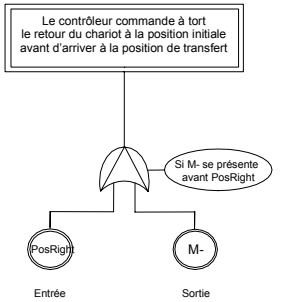
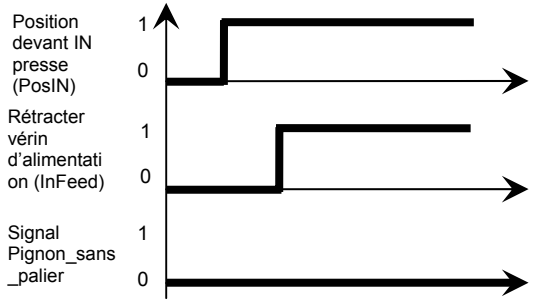
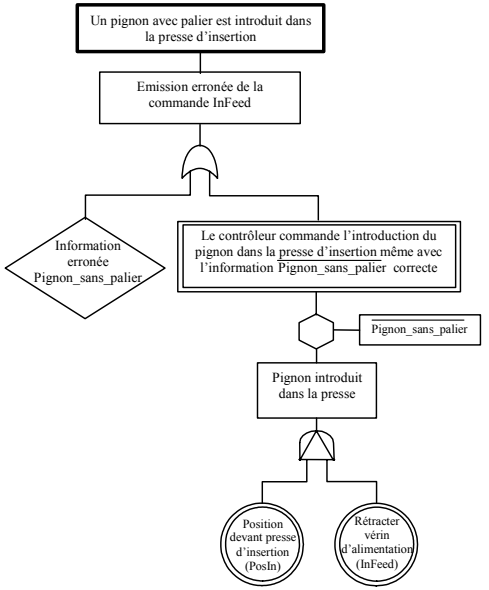
Liste des fautes identifiées de la station 3 de l'ensemble mécatronique Bosch

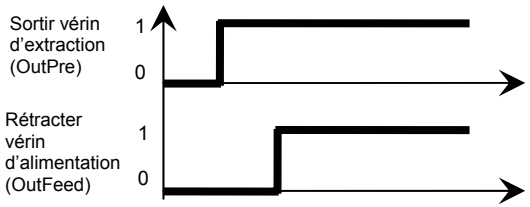
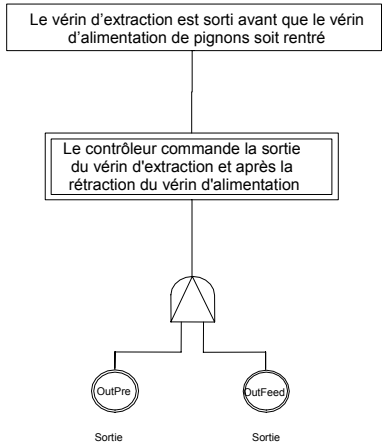
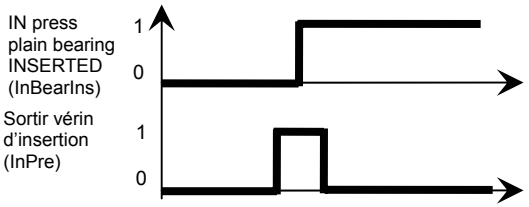
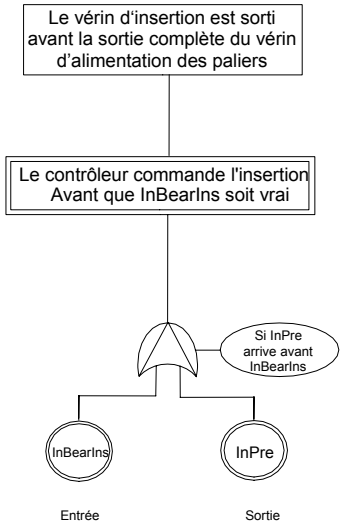
1 Fautes ne faisant intervenir aucune notion de temps

Description de la faute	Chronogramme	Extrait de l'AdD
Le chariot est envoyé à gauche et un (ou les deux) vérin d'arrêt est sorti		
Sortie du vérin d'alimentation de pignons de la presse d'extraction sans avoir le vérin d'extraction de en position haute		

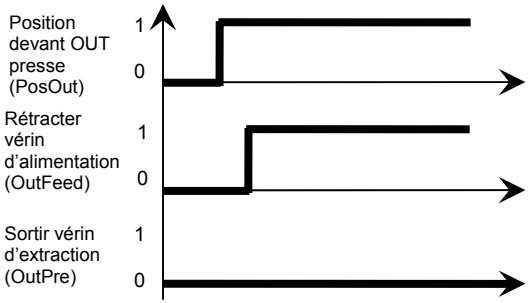
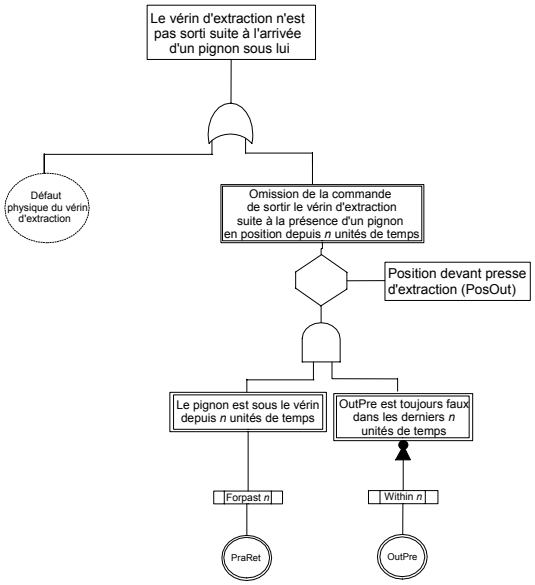
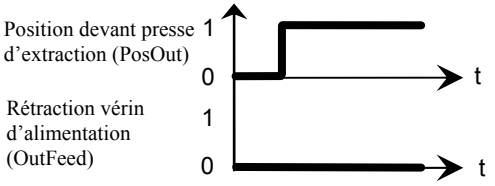
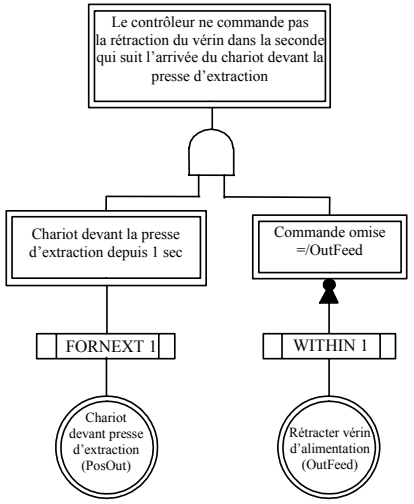
Description de la faute	Chronogramme	Extrait de l'AdD
Sortie du vérin d'extraction sans avoir un pignon en position		

2 Fautes faisant intervenir le temps logique

Description de la faute	Chronogramme	Extrait de l'AdD
Le contrôleur commande à tort le retour du chariot à la position initiale avant d'arriver à la position de transfert.		
Un pignon avec palier est introduit dans la presse d'insertion		

Description de la faute	Chronogramme	Extrait de l'AdD
Le vérin d'extraction est sorti avant que le vérin d'alimentation de pignons soit rentré		
Le vérin d'insertion est sorti avant la sortie complète du vérin d'alimentation des paliers		

3 Fautes faisant intervenir le temps physique

Description de la faute	Chronogramme	Extrait de l'AdD
Le vérin d'extraction n'est pas sorti suite à l'arrivée d'un pignon en position.		
Le contrôleur ne commande pas la rétraction du vérin dans la seconde qui suit l'arrivée du chariot devant la presse d'extraction		

Résumé • La difficulté d'exprimer les propriétés formelles d'un contrôleur logique en vue de sa vérification est un des obstacles majeurs à la diffusion de ce type de techniques. L'objectif de cette thèse est donc de faciliter l'élaboration de ces propriétés formelles en proposant une méthode basée sur l'analyse prévisionnelle par Arbre des Défaillances (AdD). Ainsi, une propriété sera la non réalisation d'une faute.

Quatre contributions sont alors développées pour mettre au point cette méthode : deux contributions de nature méthodologique et deux autres de nature formelle. Les contributions de la première catégorie sont, d'une part, l'intégration, dans la structure de l'AdD, des fautes du logiciel de commande du contrôleur logique (dites fautes systématiques car reproductibles) et, d'autre part, la représentation de ces fautes systématiques avec un vocabulaire de portes prenant en compte les temps logique et physique.

Les deux contributions méthodologiques proposent une sémantique formelle, en premier lieu, des portes adoptées dans le travail, et deuxièmement, d'associations de portes. Enfin, un exemple permet de montrer l'intérêt de ces quatre propositions pour l'amélioration de la sûreté des contrôleurs logiques.

Mots Clés • Vérification formelle, contrôleurs logiques, arbre des défaillances, propriétés formelles, analyse prévisionnelle, model-checking, logique temporelle, automates observateurs

Abstract • Expressing the formal properties of logic controllers is one of the main obstacles to the diffusion of formal verification techniques. The objective of this thesis is to facilitate formal properties elaboration, by proposing a method based on Fault Tree Analysis (FTA). Thus, a property will be the non realization of a fault.

Four contributions are then proposed to develop the method: two methodological ones and two formal ones. The contributions of the first category are, on one hand, integration, in the FTA structure, of software faults of the logic controller (known as systematic faults because they are reproducible) and, on the other hand, the representation of these systematic faults with a vocabulary of gates which enables logical and physical times representation.

The aim of the two methodological contributions is to propose a formal semantics, first, for the gates that are adopted in this work, and second, for associations of gates. Lastly, a case study shows the interest of these four proposals for improvement of dependability of the logical controllers.

Keywords • Formal verification, logic controllers, fault tree analysis, formal properties, fault forecasting, model-checking, temporal logic, observer automata