THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ BORDEAUX 1

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Fabien KUNTZ**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Une approche basée modèle pour l'optimisation du monitoring de systèmes avioniques relativement à leurs performances de diagnostic

---

Soutenue le :	10 juillet 2013	
Après avis de :	M. Eric FABRE	**Rapporteurs**
	Mme Louise TRAVÉ-MASSUYÈS	

Devant la commission d'examen formée de :

M. Eric FABRE	Directeur de Recherche (INRIA IRISA)	**Rapporteur**
Mme Stéphanie GAUDAN	Docteur Ingénieur (Thales Avionics)	**Co-directeur de thèse**
M. Alain GRIFFAULT	Maître de Conférences (LaBRI)	**Co-directeur de thèse**
M. Mohamed MOSBAH	Professeur des Universités (LaBRI)	**Président du jury**
Mme Anca MUSCHOLL	Professeur des Universités (LaBRI)	**Directeur de thèse**
M. Christian SANNINO	Ingénieur (Thales Avionics)	**Invité**
Mme Louise TRAVÉ-MASSUYÈS	Directeur de Recherche (LAAS-CNRS)	**Rapporteur**

– 2013 –

Une approche basée modèle pour l'optimisation du monitoring de systèmes avioniques relativement à leurs performances de diagnostic

Résumé

Les systèmes avioniques s'étoffent et se complexifient de plus en plus. Avec l'augmentation des capacités de calcul, de nouvelles architectures basées sur le partage de ressources émergent. Effectuer le diagnostic d'un système n'est désormais plus une opération anodine. L'enjeu actuel est donc de mettre en place des techniques de diagnostic performantes tout en optimisant les capacités de monitoring nécessaires. Ce mémoire donne une caractérisation basée modèle d'un système sous diagnostic, puis propose des techniques pour en évaluer les performances de diagnostic, ainsi que celles de son monitoring (relativement à ces performances). Le contexte industriel dans lequel s'inscrit cette thèse amène d'autres contraintes, notamment la prise en compte de la taille des systèmes avioniques à analyser. Cette thèse étudie alors l'applicabilité des techniques introduites à ce contexte et en propose une adaptation.

Mots-clés : diagnostic, avionique, monitoring, performances, modèle, modélisation, AltaRica.

A model-based approach for avionics systems monitoring optimization with respect to diagnostic performances

Summary

Avionics systems become more and more complex. With the improvment of computing possibilities, new architectures based on resources sharing are growing. Performing the diagnosis of a system is no longer a trivial operation. The challenge is to develop efficient techniques of diagnosis while optimizing required capabilities of monitoring. This thesis gives a model-based characterization of a system under diagnosis, and proposes techniques to assess both diagnostic and monitoring performances. The industrial context of this thesis brings other constraints, and in particular the need to handle the size of avionics systems to analyze. This thesis then examines the applicability of the introduced techniques to this particular context, and proposes an adaptation.

Keywords : diagnosis, avionics, monitoring, performances, model, model-based, AltaRica.

Remerciements

Je tiens tout d'abord à remercier Mohamed Mosbah d'avoir accepté de présider mon jury de thèse. Merci à Éric Fabre et Louise Travé-Massuyès qui ont rapporté ma thèse. C'est un réel honneur d'avoir eu vos retours sur mon travail de plusieurs années. Un autre honneur est de voir apparaître sur la première page de mon mémoire les noms de tels membres de la communauté qui n'apparaissaient avant que sur les dernières.

J'ai à cœur aussi de remercier Anca Muscholl et Alain Griffault, mes directeurs de thèse au LaBRI. Je vous suis très reconnaissant de l'ouverture d'esprit dont vous avez su faire preuve et qui a permis à cette thèse CIFRE de se dérouler d'une très bonne manière. Merci pour votre disponibilité et vos conseils qui se sont toujours révélés motivants et justes. Merci finalement pour votre gentillesse et votre bienveillance.

Merci à l'équipe MVTsi qui m'a accueilli au sein du laboratoire et en particulier à Gérald Point et Aymeric Vincent, mes compagnons AltaRica, qui ont été présents jusqu'au bout pour m'aider et avec qui cela a été un réel plaisir de travailler.

Je ne saurai jamais assez remercier Stéphanie Gaudan et Christian Sannino, mes encadrants à Thales Avionics Toulouse. D'un point de vue professionnel, vous vous êtes battus pour que je sois dans des conditions optimales pour effectuer mon travail de thèse, et votre abnégation dans cette tâche est aussi appréciable qu'elle est rare. D'un point de vue personnel, vous avez toujours veillé sur moi malgré le travail ou les événements ponctuels qui vous occupaient ; c'est une chose très appréciable dans le cadre d'un travail de tous les jours. Je suis conscient de la chance qui m'a été donnée de travailler aux côtés de personnes autant passionnées que passionnantes et aussi bienveillantes que vous.

Merci aux rencontres de Thales Avionics Toulouse, et en particulier à ces personnes qui m'ont aidé et avec qui j'ai échangé sur mes travaux sur le diagnostic au long de mon parcours de thèse : Eric H. Sébastien, Quentin, Jonathan, et tous les autres. Des mercis et félicitations particulières à Eric L. qui s'est toujours précipité pour m'aider et qui me supporte dans son bureau depuis très longtemps.

Merci à Ronan qui a volé et marché beaucoup de kilomètres pour me voir parler. Merci à Romain et Alex qui sont aussi venus me soutenir comme ils le font depuis très longtemps. Merci à tous mes amis qui m'ont évité la folie en me divertissant et me remuant.

Merci à ma famille qui a toujours su paraître passionnée lorsque je parlais de mon travail et en particulier à mes parents qui m'ont toujours accordé une pleine confiance et dont la fierté égale ma reconnaissance envers eux.

Merci enfin à Elodie, mon équilibre.

Table des matières

Table des figures

Liste des tableaux

Introduction

Le but de cette thèse a été principalement de définir des notions de performances de diagnostic pour des systèmes à événements discrets et ainsi de permettre l'optimisation du monitoring de ces systèmes vis-à-vis de leurs performances de diagnostic. Ce travail s'inscrivant dans un contexte industriel, il a du être applicable en particulier au cas de diagnostic des systèmes de type avionique pour la maintenance.

Les motivations de ce travail viennent avec l'évolution du transport aérien, des technologies à bord des avions et en particulier des systèmes avioniques. Thales Avionics, l'entreprise dans laquelle a été effectué ce travail de thèse, est un acteur majeur du domaine de l'aéronautique. De par cette position, Thales Avionics a pu participer et se confronter aux évolutions du domaine.

Les nécessités de changement et d'évolution dans lesquelles notre travail trouve son origine, viennent de la complexification accrue des architectures des systèmes avioniques ces dernières années. Jusqu'à il y a encore peu de temps, les systèmes avioniques étaient basés presque exclusivement sur les ressources matérielles. Nous signifions par là que chaque fonction avionique était assignée à un composant matériel qui s'occupait alors exclusivement de cette action. La miniaturisation des systèmes et l'amélioration des capacités de calcul ont, depuis quelques années, changé la tendance. Aujourd'hui, nous trouvons dans les systèmes des composants fournissant une capacité de calcul telle que plusieurs fonctions avioniques se retrouvent alors hébergées sur une même ressource de calcul. Cela constitue une des raisons de la complexification des systèmes.

Le spécialité du domaine de l'avionique dont il est question dans cette thèse est le diagnostic des systèmes avioniques pour la maintenance. Le terme de *complexification* prend tout son sens pour cette spécialité car alors qu'il était relativement peu compliqué avant de déduire de manière précise quel équipement matériel était la cause d'un dysfonctionnement du système, c'est beaucoup moins évident, et plus complexe, de résoudre un tel problème dès lors que les ressources matérielles sont partagées. Ceci est une illustration de la complexification des systèmes qui montre à elle seule en quoi il est plus difficile d'effectuer un diagnostic précis. Mais ce n'est pas la seule illustration de cela. D'autres évolutions l'accompagnent :

- L'augmentation des capacités de communication (augmentation du nombre et du type de données transitantes), des protocoles et des types de réseaux.
- Le fait qu'une fonction peut se voir hébergée de manière distribuée sur plusieurs équipements matériels.
- ...

L'amélioration des technologies et méthodes de diagnostic est inévitable et indispensable pour conserver de bonnes performances malgré l'évolution des systèmes.

Durant ces années de thèse, nous sommes passés par différentes étapes qui ont allié un travail pratique et un travail théorique. Après une recherche d'état de l'art sur les méthodes de diagnostic, nous avons attaqué le problème d'un point de vue pratique pour aborder la notion de faisabilité qui est une caractéristique difficile et importante dans le cadre d'une thèse en entreprise. Nous avons commencé par modéliser un système avionique sous un angle maintenance, en considérant le comportement du système d'un point de vue dysfonctionnel. Nous avons ainsi pu dimensionner le type de problème que nous allions avoir à traiter.

Après avoir réussi à obtenir le modèle d'un système avionique, nous avons essayé de l'exploiter afin d'obtenir des résultats utilisables dans le cadre d'un travail de diagnostic du système. Nous

avons pu trouver une solution à cette étape difficile grâce à une collaboration, en parallèle de la thèse, entre Thales Avionics et le LaBRI. Celle-ci a mené à l'obtention de relations causales, appelées règles de diagnostic. Nous avons ainsi pu expérimenter l'applicabilité pratique d'une solution de diagnostic.

Toujours avec l'objectif d'améliorer le monitoring des systèmes vis-à-vis des performances de diagnostic, notre travail s'est ensuite porté sur les façons possibles de caractériser les performances de diagnostic d'un système. Nous avons ainsi abordé les notions de détectabilité et de diagnosticabilité pour des systèmes sous diagnostic, qui permettent d'en définir la qualité pour le diagnostic. Ces notions ont pu ensuite être appliquées à des systèmes industriels, tels que de réelles suites avioniques ou des sous-ensembles de celles-ci, et ont pu être intégrées dans leur évaluation en pratique pour le diagnostic.

La dernière étape pour pouvoir améliorer le monitoring de systèmes sous diagnostic pour améliorer leurs performances a consisté à s'intéresser à la comparaison de systèmes. En s'appuyant sur les qualités de performances précédemment définies, nous avons pu construire des relations permettant de comparer et d'ordonner les systèmes selon leur qualité pour le diagnostic. Cela permet de travailler sur la modification du monitoring des systèmes et de comparer ces derniers afin d'obtenir le système sous diagnostic le meilleur.

En parallèle de ces étapes, nous nous sommes concentrés sur un autre aspect du travail, amené par le contexte industriel dans lequel il s'inscrivait. À chaque étape d'un travail théorique nous avions en tête comment il s'appliquerait sur des systèmes industriels. Mais plus que cela, pour que la solution que nous proposons soit totalement réaliste, il fallait qu'elle soit applicable au contexte industriel. Nous avons pour cela travaillé sur l'industriabilité de notre solution en cherchant à automatiser ses différentes étapes, et notamment celle de modélisation qui est une étape cruciale et coûteuse d'un point de vue industriel.

L'ensemble des travaux de thèse que nous venons de présenter va être décrit dans ce document. Nous avons décidé de présenter les aspects théoriques et pratiques des travaux dans des parties différentes afin d'en faciliter la compréhension et de faciliter la lecture de ce rapport. Cela ne reflète pas la façon dont a été traité le sujet qui est plus proche en réalité de ce qui a été décrit dans les paragraphes précédents.

Pour présenter le travail effectué pendant cette thèse, nous organisons le document en trois parties divisées chacune en chapitres.

Dans la première partie, nous exposons le contexte des travaux. Le chapitre 1 présente le contexte industriel (diagnostic et maintenance) dans lequel se sont déroulés les travaux et le chapitre 2 fait un état de l'art de différentes approches utilisées pour faire du diagnostic.

Dans la deuxième partie, nous nous intéressons à la caractérisation du monitoring et des performances de diagnostic. Le chapitre 3 formalise le problème de diagnostic ainsi que le type de systèmes considérés et certaines de leurs propriétés particulières. Le chapitre 4 décrit les performances de diagnostic d'un système à travers des définitions et métriques caractérisant la détectabilité et la diagnosticabilité des systèmes. Le chapitre 5 montre comment s'appuyer sur les performances définies en chapitre 4 pour comparer les systèmes et optimiser leur monitoring.

Dans la troisième partie, nous abordons les aspects pratiques du travail à travers des expérimentations mais aussi en montrant comment le travail s'est intégré au contexte industriel. Le chapitre 6 présente le langage AltaRica utilisé pour modéliser les systèmes, explique ensuite comment évaluer les performances de diagnostic définies dans la deuxième partie sur ces modèles, et présente finalement un exemple de cela. Le chapitre 7 s'intéresse lui à l'intégration industrielle des travaux en présentant le groupe d'études dans lequel ces travaux ont été menés et en décrivant comment les travaux du groupe d'études ont évolué à travers le travail de thèse.

Première partie

Diagnostic – Contexte et Méthodes

Chapitre 1

Diagnostic et maintenance

1.1 Définition

Lorsque nous recherchons dans la littérature des informations concernant le mot "diagnostic", nous nous rendons rapidement compte que c'est un terme ambigu et qu'il n'a pas toujours la même définition. Parfois, nous pouvons lire que diagnostiquer un système consiste à détecter lorsqu'une défaillance se produit, d'autres fois, le diagnostic est défini comme étant la localisation des éléments à l'origine de la défaillance. D'autres fois encore, le diagnostic est l'ensemble de ces deux phases, d'abord la détection d'une erreur, puis dans un deuxième temps l'identification de la ou les causes de cette défaillance.

Loïc Hélouët et ses collègues ont d'ailleurs identifié une autre partie de ce problème dans [HGG06]. Ils distinguent ce qu'ils appellent "*fault diagnosis*" qui a pour but de savoir si l'occurrence d'une défaillance peut être finalement détectée après un nombre fini d'observations et "*history diagnosis*" qui correspond à la recherche de causes. Leur nommage n'est pas universel, mais il dénote bien de la nécessité de préciser quel est le diagnostic dont il est question.

La définition officielle du mot diagnostic (*diagnosis*) que donne le dictionnaire Oxford est la suivante : *"the identification of the nature of an illness or other problem by examination of the symptoms"*. On peut traduire cela par *"l'identification de la nature d'une maladie ou d'un autre problème à partir de l'examen de symptômes"*. Dans le domaine du diagnostic de systèmes, cela revient à expliquer la situation problématique observée.

Notre vision des choses, qui correspond à la définition donnée par le dictionnaire Oxford, est que diagnostiquer revient à trouver/localiser les causes d'un problème observé. Nous considérons que la phase de détection est en amont et tient plus de la surveillance que du diagnostic lui-même.

Le diagnostic est utilisé dans plein de domaines différents. Le premier que l'on associe est le domaine de la médecine. Dans ce contexte, l'idée est à partir d'un ensemble de symptômes d'un patient, trouver l'état dans lequel il est (et ce qui a pu le rendre malade).

On retrouve aussi la même notion dans le domaine de l'enquête policière. Ce que l'on souhaite ici c'est trouver les coupables à partir des indices qu'ils ont laissés. Ainsi nous avons des suspects, qui sont disculpés à l'aide d'indices et d'alibis. Nous retrouvons tous ces termes lorsque nous parlons de diagnostic.

Et bien sûr nous avons aussi le diagnostic de système, où le but est de localiser ou identifier les différentes causes qui ont pu provoquer un ensemble d'effets observés.

Par la suite nous utiliserons donc indifféremment les termes symptômes et effets observés, ainsi que les termes causes et suspects/coupables.

1.2 Maintenance avionique

La maintenance et la réparation des avions sont de la responsabilité des compagnies aériennes. Les différents constructeurs fournissent avec leurs produits des informations et des méthodes de

maintenance de ces produits. En outre, certains acteurs, parmi lesquels THALES AVIONICS, vendent des services et des garanties lorsqu'ils vendent leur produit.

La maintenance que nous considérons porte sur le système avionique et plus précisément sur les équipements qui le composent.

Les intérêts d'une maintenance efficace sont multiples :

- Les pannes sont réparées plus rapidement ce qui permet à l'avion de passer moins de temps au sol. C'est un gain pour les compagnies aériennes qui gagnent ainsi en temps d'immobilisation et diminuent les coûts de possession.
- Les pannes sont mieux localisées ce qui évite les *NFF* (*No Fault Found)* et ainsi les fausses déposes. En effet, lorsqu'un équipement est identifié comme étant en panne, l'agent de maintenance le remplace simplement et renvoie l'équipement incriminé aux responsables de sa maintenance. L'équipement est ensuite analysé et réparé. Dans les cas de fausses déposes, ce sont des équipements fonctionnels qui sont incriminés, changés et renvoyés pour réparation.

La chaîne de maintenance fait intervenir différents acteurs, parmi lesquels :

- L'équipage de l'avion qui va noter les dysfonctionnements qu'il observe durant le vol.
- L'équipe de maintenance en ligne vient effectuer le diagnostic de l'avion et identifier les équipements (dits *LRU* pour *Line Replaceable Unit*) défaillants.
- Les agents de maintenance remplacent les LRU incriminés par des LRU fonctionnels.
- À côté de cela, le Centre de Maintenance Centrale (CMC) s'occupe de gérer la logistique d'acheminement des équipements et la planification des interventions de maintenance.

La phase qui nous intéresse dans la maintenance est la phase de localisation de pannes. Sa qualité est indispensable pour assurer une maintenance efficace. Le travail de localisation de pannes est principalement basé sur la chaîne de diagnostic.

1.3 Diagnostic pour la maintenance avionique

1.3.1 Chaîne de diagnostic

La chaîne de diagnostic pour la maintenance a pour rôle de localiser les pannes qui arrivent dans le système avionique afin de permettre à l'agent de maintenance de le remettre en état.

La chaîne de diagnostic pour la maintenance telle que nous la considérons est représentée en figure 1.1.

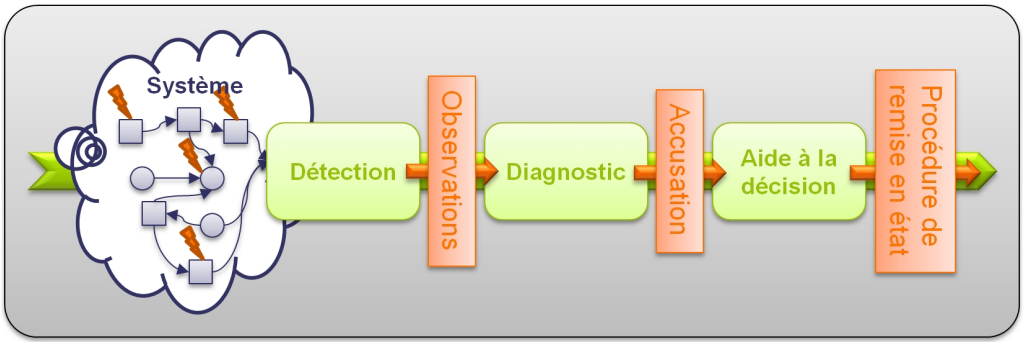


FIGURE 1.1 – Chaîne de diagnostic pour la maintenance

Elle est composée de trois étapes principales :

1. La détection.
2. Le diagnostic.
3. L'aide à la décision.

La première étape a pour rôle de détecter lorsque le système est défaillant. Elle utilise l'ensemble des informations fournies par les capteurs du système (ou *Monitoring*) pour détecter les anomalies et donc la présence de défaillances. Une fois une anomalie détectée, des messages de monitoring caractérisant l'état du système sont envoyés aux composants chargés du diagnostic. Cet ensemble de message va représenter une observation du système. Le système déclenche alors l'étape suivante.

L'étape de diagnostic a pour rôle d'expliquer l'observation qui lui est donnée. Il s'agit alors de localiser les défaillances qui sont arrivées. Les résultats de l'étape de diagnostic sont formulés sous la forme d'une *accusation* (appelée aussi *diagnostic*) qui est fournie pour la dernière étape de la chaîne.

La dernière étape de la chaîne de diagnostic consiste à organiser les éléments de l'accusation fournie par l'étape de diagnostic pour les présenter ensuite à l'agent de maintenance. L'idée de cette étape intitulée "Aide à la décision" est de classer les résultats de diagnostic selon des préférences qui peuvent porter sur la probabilité d'occurrence, la disponibilité des équipements, le coût de réparation, etc. Ainsi, certaines défaillances pourront être présentées en priorité à l'agent de maintenance par exemple parce que leur occurrence est plus fréquente. Le résultat de cette étape consiste en une procédure de remise en état du système, qui peut être simplement sous la forme d'une liste d'équipements à remplacer, ou à travers une procédure matérialisée par un arbre de décision qui donnera le déroulement des interventions et vérifications à effectuer par l'agent de maintenance, ou bien peut prendre encore d'autres formes.

1.3.2 Topologie du diagnostic

Le point de départ du diagnostic est l'observation du système. Ce sont les valeurs relevées par les capteurs et les messages transmis par l'ensemble des équipements de monitoring qui définissent les données d'entrée de l'opération de diagnostic.

Les opérations de diagnostic s'effectuent ensuite en s'appuyant sur un ensemble d'équipements spécifiques. Ces équipements sont de différents types selon le périmètre dont ils sont responsables. Une topologie que l'on retrouve souvent pour le diagnostic des systèmes avioniques est la présence de diagnostiqueurs locaux et d'un ou de plusieurs diagnostiqueurs plus globaux.

Les diagnostiqueurs locaux ont la responsabilité d'un périmètre réduit et ont souvent un rôle génériques qui consiste en plusieurs étapes :

- Récupérer les données de capteurs dans leur périmètre.
- Ingérer ces données et les communiquer au diagnostiqueur central.

Ces diagnostiqueurs locaux de manière générale ne mettent pas en place des procédés de corrélation de messages et se concentrent sur le relais des informations de monitoring. Les opérations qu'ils effectuent sont principalement la traduction et le filtrage des données issues des capteurs. Les équipements appelés *BITE*[1] jouent par exemple un rôle de cet ordre.

Ensuite les messages émis par les composants locaux sont reçus par le diagnostiqueur global qui a pour rôle de les corréler et d'en déduire un diagnostic.

Chaque fournisseur de systèmes a sa propre topologie avec un nombre de filtres-relais plus ou moins important, mais il répond bien souvent à la topologie que nous venons de décrire. Aussi, les composants que nous avons caractérisés par le terme "diagnostiqueurs locaux" sont parfois considérés comme des composants de surveillance/monitoring plutôt que comme des diagnostiqueurs purs. Néanmoins, au vu de la description que nous avons faite dans la section précédente, nous pouvons affirmer qu'ils font entièrement partie de la chaîne de diagnostic.

1. ***B**uilt-**I**n **T**est **E**quipment*

Chapitre 2

Approches pour le diagnostic

Il est apparu durant nos recherches sur le diagnostic que dans la littérature, de nombreux "types de diagnostic" sont nommés. Voici quelques-uns uns des termes rencontrés les caractérisant : “rule-based”, “case-based”, “model-based”, mais aussi “image-based”, “evidence-based”, “parent-based”, “syndromic-based”, “laboratory-based”, “questionnaire-based”, “conflict-based”, “software-based”, “intention-based”, “knowledge-based”, “web-based”, “molecular-based”, etc. Après analyse, il ressort que certains de ces qualificatifs représentent des types de diagnostic établis, qui sont étayés de nombreux articles, sont appliqués dans le monde de l'industrie, sont supportés par des communautés et ont un historique conséquent. C'est le cas des diagnostics à base de règles, à base de cas et à base de modèle, qui sont trois raisonnements intéressants et différents.

Dans ce chapitre nous allons donc exposer les trois types de raisonnement majeurs pour le diagnostic que sont le diagnostic à base de règles (rule-based), le diagnostic à base de cas (case-based) et le diagnostic à base de modèle (model-based). Nous allons présenter chacun de ces raisonnements avec une première partie consacrée à la description de l'approche, une deuxième qui proposera une illustration du raisonnement sur un exemple avant d'en faire la critique, et nous finirons en donnant des exemples d'outils rencontrés durant les recherches menés sur ces approches. Nous profiterons en outre de la section sur le raisonnement à base de règles pour présenter la méthode la plus répandue chez les industriels, et notamment dans l'aéronautique et à Thales Avionics, pour faire du diagnostic.

2.1 Raisonnement à base de règles (Rule-Based Reasoning ou RBR)

2.1.1 Description

2.1.1.1 Introduction au raisonnement à base de règles

Le raisonnement à base de règles est une des toutes premières méthodes de raisonnement utilisées pour le diagnostic. Il utilise des règles prédéfinies de type déduction (elles seront décrites plus tard) ou des arbres de décision pour localiser l'origine d'une panne.

Le RBR est apparu dans les années 40 et continue d'être utilisé aujourd'hui dans le milieu industriel.

Nous allons dans cette partie donner un bref historique du raisonnement à base de règles, puis expliquer comment fonctionne un système basé sur ce raisonnement.

2.1.1.2 Historique du raisonnement à base de règles

Charles M. Higgins, Jr., dès 1993, s'intéresse dans sa thèse [Jr.93] aux systèmes à base de règles et en propose un historique. Nous allons présenter les grands points de l'histoire des systèmes à

base de règles ; on se référera à cette thèse ou à la thèse de Padhraic Smyth [Smy88] pour plus de détails.

L'apparition des systèmes à base de règles s'est déroulée dans la première moitié des années 40. Elle a commencé avec Emil L. Post [Pos43] et ses systèmes de production, une forme générale de système d'inférence. Les systèmes de production ont très rapidement attiré les chercheurs des sciences cognitives du fait de leur proximité avec la façon de penser humaine. Dans les années 60, Allen Newell et Herbert A. Simon [NS65, New68] ont réfléchi à l'application de ces systèmes pour la modélisation psychologique, puis pour la modélisation cognitive humaine en général dans les années 70. Nous pouvons citer PROLOG [CKPR72], un des langages de programmation à base de règles les plus connus, qui est inventé en 1972 par un groupe autour d'Alain Colmerauer. En 1986, John H. Holland [HHNT86] écrit un livre sur l'utilisation des systèmes de production pour les modèles cognitifs.

C'est dans les systèmes experts que les systèmes à base de règles ont été grandement utilisés. L'idée des systèmes à base de règles est de représenter l'ensemble des connaissances d'un expert par rapport à un problème particulier sous forme de règles, et ainsi de pouvoir résoudre des problèmes de décision complexes. On se rend vite compte que pour un gros problème, il devient difficile de construire un ensemble de règles suffisamment complet et d'éviter les contradictions. Cette difficulté est devenue connue sous le nom de "knowledge acquisition bottleneck" que l'on pourrait traduire par "goulot d'étranglement de l'acquisition de connaissances".

Malgré cela, il y a eu de nombreux exemples de systèmes experts construits de cette manière. Parmi les plus connus, on trouve MYCIN [SAB+73] qui donne des conseils sur le diagnostic et la thérapie de maladies infectieuses, DENDRAL [LBFL80] qui analyse les spectrogrammes de masse pour déterminer la structure moléculaire, et XCON/R1 [McD80] qui configure les systèmes VAX pour Digital Equipment Corporation. Les systèmes experts ont été utilisés dans de nombreuses disciplines, comme par exemple l'ingénierie civile [TTE91], la chimie [MWM91, HBZ92], ou encore les sciences du nucléaire [CCC93].

Le raisonnement à base de règles est la méthode de diagnostic la plus utilisée dans le monde industriel depuis plusieurs années déjà, même si aujourd'hui, les recherches se tournent plutôt vers d'autres approches comme celle à base de cas ou à base de modèle que nous traiterons par la suite dans ce document.

2.1.1.3 Système à base de règles

Les systèmes experts à base de règles essaient de traduire les associations heuristiques utilisées par les experts humains. Ils ont été conçus à l'origine pour aider des utilisateurs à résoudre un problème dans un domaine spécifique nécessitant des connaissances expertes.

Une règle est exprimée sous la forme d'expressions "si-alors" :

- SI (une condition particulière apparaît) ALORS (une action associée est effectuée).

Les conditions sont exprimées sous la forme d'expressions logiques telles que des conjonctions ou même des disjonctions de prédicats. Elles peuvent correspondre par exemple à des données de capteurs.

Un système à base de règles typique comprend quatre composants :

- Une base de règles.
- Un moteur d'inférence.
- Une mémoire à court terme.
- Une interface d'utilisation ou un autre moyen de connexion au monde extérieur au moyen de signaux d'entrées et de sorties reçus et envoyés.

Toutes les règles sont donc stockées dans une base de règles. Comme on peut le voir sur la table 2.1, cette base est typiquement composée d'un ensemble de règles où chacune est décomposée en deux parties : expression booléenne et action.

Expression booléenne	Action
Expression booléenne A	Action A
Expression booléenne B	Action B
Expression booléenne C	Action C
.	.
.	.
.	.
.	.
.	.

TABLE 2.1 – Table de règles (de [B05TATEM])

La méthode de résolution typique d'un problème est la suivante : on parcourt la base pour trouver l'expression booléenne qui correspond au problème, et une fois que cela est fait, on exécute l'action associée à l'expression trouvée.

Un moteur d'inférence est un logiciel correspondant à un algorithme de simulation basé sur des raisonnements déductifs. Il permet, via des raisonnements logiques, de décider des actions à suivre en fonction d'une situation, à partir de la base de règles. Plusieurs méthodes d'implémentation existent, dont notamment celles basées sur une logique formelle d'ordre 0 (logique des propositions), 0+, 1 (logique des prédicats) ou d'ordre 2 avec :

- Une gestion d'hypothèses monotone ou non monotone.
- Un chaînage avant, arrière ou mixte.
- Une complétude déductive ou non.

2.1.2 Illustration du diagnostic à base de règles

Pour illustrer les différentes approches, nous avons décidé de prendre un exemple de système simple (il sera dénommé l'Exemple). Il sera utilisé pour l'illustration de chacune des approches présentées. Le système a été simplifié sur certains points et des hypothèses ont été posées afin de permettre de faire comprendre simplement comment peuvent fonctionner les différents types de diagnostic. Nous allons donc dans un premier temps exposer le problème considéré tout en justifiant les différentes simplifications et hypothèses, puis nous allons commencer par illustrer la méthode de diagnostic à base de règles sur cet exemple.

2.1.2.1 Présentation de l'Exemple

Le système à diagnostiquer sera un système simple composé de six composants : Deux sondes, un comparateur, deux fonctions et un diagnostiqueur. Le système considéré est représenté sur la figure 2.1.

Nous allons maintenant décrire les composants du système plus particulièrement.

Une sonde est un composant qui va envoyer la valeur d'une grandeur qu'elle mesure (dans la réalité, cela peut être la pression, la température, la vitesse...). Cette valeur est comprise dans un intervalle que nous noterons $[a, b]$. Nous considérons que la sonde peut se trouver dans trois états différents :

- SAFE : la sonde fonctionne parfaitement et produit donc une donnée dans son intervalle de valeurs correctes.
- ERR : la sonde a un fonctionnement dégradé et envoie une valeur incorrecte appartenant à l'intervalle de valeurs correctes.
- LOST : la sonde est défaillante de manière visible : elle n'envoie plus de données ou produit une valeur hors de l'intervalle attendu.

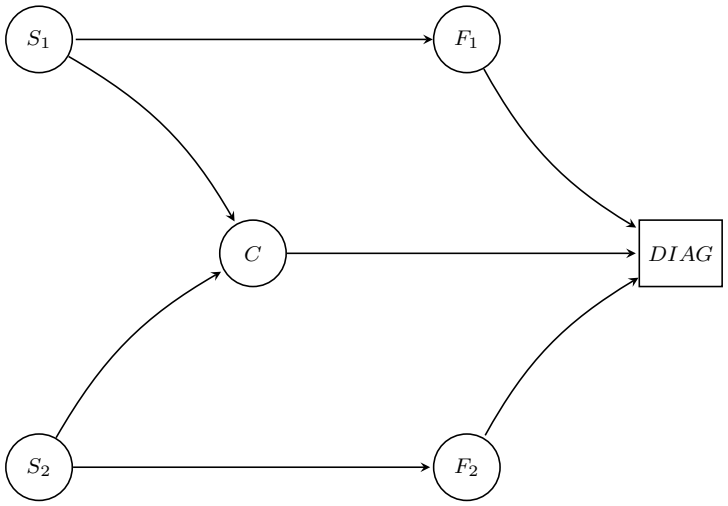


FIGURE 2.1 – Système Exemple

Un comparateur est un composant qui va recevoir deux données et qui va dire si elles sont égales ou différentes. Nous considérons que le comparateur n'envoie jamais de réponse erronée, autrement dit qu'il ne dira jamais que les deux données d'entrée sont différentes si elles sont égales et vice et versa. Ce composant a donc deux états :

- SAFE : le comparateur fonctionne parfaitement.
- LOST : le comparateur n'envoie plus aucune valeur.

Une fonction a pour rôle de recevoir une donnée et de dire si elle appartient ou non à un intervalle particulier (cet intervalle de valeurs correctes sera noté $[x, y]$). Si elle ne reçoit aucune donnée, elle va aussi le dire. Pour des raisons de simplicité, nous allons supposer que la fonction n'est jamais perdue ou erronée. Elle n'a donc qu'un état :

- SAFE : la fonction fonctionne parfaitement.

Enfin, le diagnostiqueur prend en entrée plusieurs messages, et son rôle est de fournir un diagnostic à partir d'eux. Il représente dans notre exemple le domaine d'observabilité du système en récupérant les messages des composants C, F_1 et F_2 qui sont observés. Nous considérons qu'il n'est soumis à aucune défaillance, et il n'a donc aussi qu'un état :

- SAFE : le diagnostiqueur fonctionne parfaitement.

La présence des comparateur et fonctions a pour but ici de prévenir les erreurs sur la valeur qu'envoient les sondes.

Pour ne pas rendre complexe, alourdir inutilement les illustrations des différentes approches, quelques simplifications et hypothèses supplémentaires ont été faites :

- Tout d'abord, les transferts de données entre les différents composants sont supposés parfaits, c'est-à-dire qu'il n'y a aucune perte possible entre l'envoi et la réception d'une donnée, alors qu'en réalité, les câbles par lesquels transitent les informations sont aussi soumis à défaillance.
- Ensuite, les deux sondes du système sont supposées ne pas pouvoir défaillir exactement de la même façon, les versions des sondes étant différentes.

On peut maintenant décrire l'Exemple de manière plus formelle.

Soient S_1 et S_2 représentant les deux sondes, C le comparateur, F_1 et F_2 les fonctions, et $DIAG$ le diagnostiqueur.

Soit $Syst = \{S_1, S_2, C, F_1, F_2, DIAG\}$ le système à diagnostiquer.

Introduisons la fonction σ qui associe à un composant l'ensemble des états dans lesquels il peut se trouver.

$$\text{Pour } q \in Syst, \sigma(q) = \begin{cases} \{SAFE, ERR, LOST\} & \text{si } q \in \{S_1, S_2\} \\ \{SAFE, LOST\} & \text{si } q = C \\ \{SAFE\} & \text{si } q \in \{F_1, F_2, DIAG\} \end{cases}$$

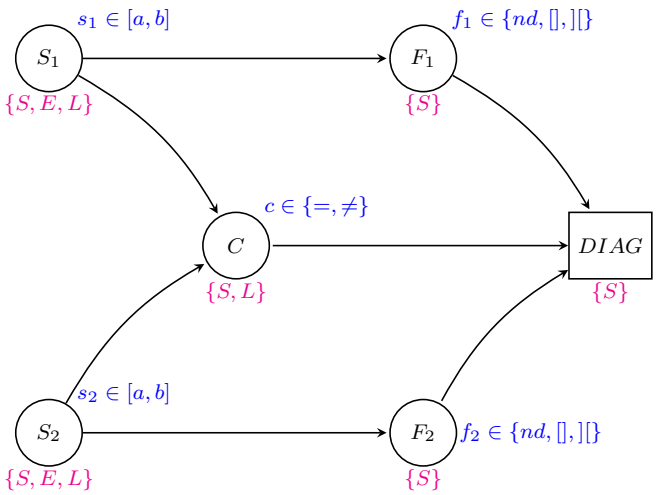
Introduisons finalement la fonction μ qui associe à un composant l'ensemble des valeurs qu'il peut émettre.

$$\text{Pour } q \in Syst, \mu(q) = \begin{cases} [a, b] & \text{si } q \in \{S_1, S_2\} \\ \{=, \neq\} & \text{si } q = C \\ \{[],][, nd\} & \text{si } q \in \{F_1, F_2\} \end{cases}$$

où $[a, b]$ représente l'ensemble des valeurs possibles des sondes.

où [] signifie que la valeur d'entrée est dans l'intervalle attendu par la fonction (l'intervalle $[x, y]$),][hors intervalle, et nd signifie qu'aucune valeur n'est reçue.

On peut alors représenter le système par la figure suivante :



– Valeur transférée
– États

FIGURE 2.2 – Système Exemple – valeurs et états

2.1.2.2 Illustration

Dans cette partie nous allons donc montrer l'utilisation du diagnostic à base de règles sur l'Exemple.

Nous allons présenter différents types de règles en mettant en place deux sous-exemples. Le premier représentera le type de base de règles qui est majoritairement utilisé aujourd'hui, le deuxième décrira une base de règles permettant un diagnostic plus précis.

2.1.2.2.1 Base "messages ↔ coupables"

Le premier type de règles que nous considérons est donc celui qui est le plus utilisé aujourd'hui dans notre domaine lorsque l'on veut faire du diagnostic à base de règles. Ces règles associent des coupables à des messages.

Rappelons le système considéré dans notre Exemple (figure 2.1).

Les messages sont en fait les valeurs que va recevoir le diagnostiqueur, ou plus exactement, ce qu'il va pouvoir observer à partir de ces valeurs. Autrement dit, dans notre cas, chaque valeur

différente transmise par un composant au diagnostiqueur correspond à un message différent, mais à cela s'ajoute le fait que le comparateur peut tomber en panne et n'envoyer aucune donnée au diagnostiqueur, ce qui doit être aussi considéré comme une information.

Les coupables sont les composants qui sont suspectés d'être défaillants. Ils sont exprimés sous forme logique de "et" et de "ou". Par exemple, "A et B ou A et C" signifie "soit A et B sont coupables, soit A et C le sont", soit les deux.

Considérons la table de règles 2.2 reposant sur ce type de règles "messages ↔ coupables".

	Messages	Coupables
1	$F_1.nd$	S_1
2	$F_1.[]$	$-$
3	$F_1.][$	S_1
4	$F_2.nd$	S_2
5	$F_2.[]$	$-$
6	$F_2.][$	S_2
7	$C.=$	$-$
8	$C.\neq$	$S_1 \vee S_2$
9	$C.none$	C

TABLE 2.2 – Table de règles messages ↔ coupables

On peut remarquer que le comportement nominal du système est la réception des messages suivants au niveau du diagnostiqueur : $F_1.[]$ et $F_2.[]$ et $C.=$. Cela représente le fait que les données sont égales et dans le bon intervalle.

Regardons maintenant un cas où le système a besoin d'être diagnostiqué. Considérons l'arrivée des messages suivants : $F_1.[]$ et $F_2.][$ et $C.\neq$. Pour diagnostiquer le système, le moteur d'inférence va corréler les accusations de chacun des messages. Autrement dit, en l'exprimant de manière logique, le diagnostic sera la conjonction des messages, ce qui donne : $- \wedge S_2 \wedge (S_1 \vee S_2) \iff (S_1 \wedge S_2) \vee S_2$. Le diagnostic est donc clair, on sait que S_2 est défectueuse et il est possible que S_1 le soit aussi.

Considérons un autre exemple, le cas où le diagnostiqueur interprète les messages suivants : $F_1.[]$ et $F_2.][$ et $C.none$. Grâce au moteur d'inférence, on obtient l'accusation suivante : $S_2 \wedge C$. Cela signifie que les deux composants S_2 et C sont défaillants.

Nous pouvons remarquer que ce type d'accusation par coupables matériels n'est pas très précise car elle ne distingue pas par exemple un composant en panne (à remplacer) d'un composant juste défaillant (peut-être uniquement un problème de réglages). Intéressons-nous donc à un deuxième type de règles.

2.1.2.2.2 Base "observations ↔ événements"

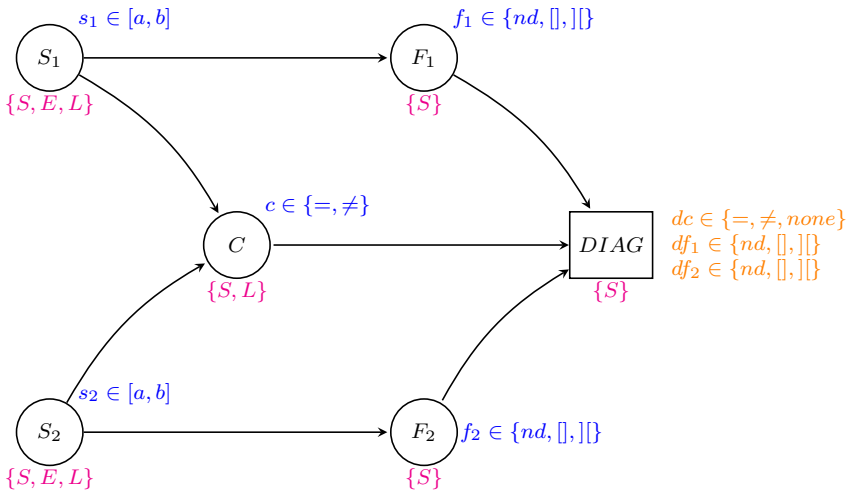
Le deuxième type de règles que nous voyons sont les règles qui associent des événements à une observation.

Rappelons le système considéré dans notre Exemple (figure 2.1). Les observations correspondent à l'état des variables d'observation du système. Ici, une observation est un triplet contenant les valeurs transférées des composants C, F_1 et F_2 interprétées par le diagnostiqueur. Par exemple, l'observation (=, [], []) décrit le système à diagnostiquer dans son comportement nominal.

Sur la figure 2.3, les variables d'observation notées dc, df_1 et df_2 sont représentées au niveau du diagnostiqueur.

Un triplet correspond à une observation est donc de type (dc, df_1, df_2).

Ce qui est appelé observation correspond à une formule logique sur les états des composants observables : C, F_1 et F_2. Par exemple, dans le cas nominal, tous les composants sont en état de fonctionnement, la formule est donc : $S_1.SAFE \wedge S_2.SAFE \wedge C.SAFE \wedge F_1.SAFE \wedge F_2.SAFE \wedge$



– Valeur transférée
– États
– Variable d'observation

FIGURE 2.3 – Système Exemple avec variables observables

$DIAG.SAFE$. Étant donné que les composants F_1, F_2 et $DIAG$ ne peuvent pas être dans un état autre que $SAFE$, nous exprimerons les événements uniquement selon S_1, S_2 et C.

Considérons la table 2.3 reposant cette fois-ci sur des règles du type "observations ↔ événements" :

	Observations	Événements
1	$(none, nd, nd)$	$C.L \wedge S_1.L \wedge S_2.L$
2	$(\neq, nd, nd)$	$C.S \wedge S_1.L \wedge S_2.L$
3	$(none, nd, --)$	$C.L \wedge S_1.L \wedge S_2.SE$
4	$(\neq, nd, --)$	$C.S \wedge S_1.L \wedge S_2.SE$
5	$(none, --, nd)$	$C.L \wedge S_1.SE \wedge S_2.L$
6	$(\neq, --, nd)$	$C.S \wedge S_1.SE \wedge S_2.L$
7	$(none, [], [])$	$C.L \wedge S_1.SE \wedge S_2.SE$
8	$(\neq, [], [])$	$C.S \wedge [(S_1.S \wedge S_2.E) \vee (S_1.E \wedge S_2.S) \vee (S_1.E \wedge S_2.E)]$
9	$(=, [], [])$	$C.S \wedge S_1.S \wedge S_2.S$
10	$(none, [],][)$	$C.L \wedge S_1.SE \wedge S_2.E$
11	$(\neq, [],][)$	$C.S \wedge S_1.SE \wedge S_2.E$
12	$(none,][, [])$	$C.L \wedge S_1.E \wedge S_2.E$
13	$(\neq,][, [])$	$C.S \wedge S_1.E \wedge S_2.SE$
14	$(\neq, --, --)$	$C.S \wedge [(S_1.S \wedge S_2.EL) \vee (S_1.EL \wedge S_2.S) \vee (S_1.EL \wedge S_2.EL)]$
15	$(none,][,][)$	$C.L \wedge S_1.E \wedge S_2.E$
16	$(\neq,][,][)$	$C.S \wedge S_1.E \wedge S_2.E$

où $S = SAFE, E = ERR, L = LOST, SE = SAFE \vee LOST, EL = ERR \wedge LOST$.

TABLE 2.3 – Table de règles observations ↔ événements

Ici, le comportement nominal est représenté dans le tableau par la ligne 9.

Nous remarquons que dans ce type de table, beaucoup d'informations doivent être représentées.

Dans la table que nous utilisons ici, l'ordre des règles est important. En effet, pour évaluer la situation dans laquelle se trouve le système, la table sera parcourue à partir de sa première entrée jusqu'à sa dernière, en s'arrêtant à la première règle qui convient.

Cet ordre apporte une facilité d'expression. Prenons l'exemple de la ligne 5 : $(none, -, nd)$. Elle représente les observations $(none, nd, nd)$, $(none, [], nd)$ et $(none,][, nd)$. Néanmoins, on voit au niveau des événements que le fait que le composant S_1 soit dans l'état LOST n'est pas considéré, alors que cela est clairement le cas lors de l'observation (none, nd, nd). Ce qui explique qu'il n'est pas considéré est le fait que le composant S_1 puisse être dans l'état LOST et que la seule observation qui l'induirait, (none, nd, nd), est déjà capturée à la première ligne de la table. En résumé, une observation capturée à une certaine ligne de la table ne le sera plus par la suite. Cette caractéristique permet de diminuer significativement la taille de la table de règles, mais cela ajoute des possibilités d'imprécision et d'erreurs.

Intéressons-nous à un cas où nous rencontrons de l'imprécision. Considérons l'observation $(\neq,][,][)$. Nous nous rendons compte en parcourant la table comme le ferait un moteur d'inférence, que nous nous arrêtons à l'observation $(\neq -, -)$ de la ligne 14 dont la formule associée est la suivante : $C.S \wedge [(S_1.S \wedge S_2.EL) \vee (S_1.EL \wedge S_2.S) \vee (S_1.EL \wedge S_2.EL)]$. Cet événement est clairement moins précis que celui de la ligne 16, dont l'observation correspond exactement à l'observation considérée dans notre cas.

Sur cet exemple simple, il est relativement facile de remarquer ces imprécisions. Il faut se dire que dans des systèmes bien plus complexes, il est impossible pour l'esprit humain de détecter des imprécisions de ce type, ou même parfois des erreurs dues à l'ordre. Les erreurs ne sont relevées qu'en étant confronté à la situation correspondante. Il peut aussi y avoir simplement des oublis ou des erreurs lors de la création de la table.

2.1.3 Avantages et limites

L'approche à base de règles est simple et intuitive puisqu'elle est basée sur un raisonnement humain. Elle peut s'appliquer efficacement à des systèmes simples. Les systèmes experts fonctionnent mieux pour des domaines assez simples et bien structurés, où les expressions booléennes sont adaptées.

Cette méthode est aussi plutôt facile à valider. Elle peut l'être par une simple simulation. Cela permet par exemple d'utiliser des techniques de raffinement par validation.

Cependant, il peut parfois s'avérer compliqué de réussir à traduire tous les problèmes d'un domaine sous forme d'expression booléenne, ce qui implique une perte d'information pour les connaissances ne pouvant être exprimées. En outre, cela demande beaucoup de temps pour les experts de convertir leur savoir en règles quand cela est possible. On peut aussi percevoir quelques difficultés concernant directement les règles :

- Formuler une règle sous la forme "si-alors" est obligatoire.
- Gérer la maintenance de la base de règles.
- Déterminer si une situation entre dans le périmètre d'une règle.
- Trouver une règle claire couvrant la situation.
- Gérer les conflits entre règles.
- Gérer les exceptions aux règles.
- La totalité des règles doit être stockée.
- Un trop grand nombre de règles dans le cas de systèmes complexes. La couverture d'un domaine complexe ou de n'importe quel domaine mal structuré peut exiger une quantité énorme de règles.

À cela s'ajoute le fait que ce type de système expert ne peut pas traiter le cas d'une faute qui n'a pas été prévue à la base. Le système n'a en effet aucune capacité d'apprentissage. Les systèmes s'appuyant sur des raisonnements à base de règles sont des systèmes statiques. Ils n'apprennent

pas de l'expérience, ils ne génèrent pas de nouvelles règles et n'apprennent pas des erreurs non plus.

Plus généralement, la base de règles sera spécifique au diagnostic et non réutilisable pour d'autres tâches.

2.1.4 Outils

Les principaux outils pour le raisonnement à base de règles correspondent aux différents moteurs d'inférence. En voici quelques exemples :

- CLIPS : Chaînage avant [1], Contrôle irrévocable, Moteur d'ordre 1 [2], Logique non monotone, Monde fermé [3].
- Moteur de MYCIN : système de diagnostic médical (voir [BS84]).
- GOSSEYN : moteur d'ordre 1^2 en chaînage avant, développé en Fortran par Jean-Marc Fouet.
- PROLOG II : moteur d'ordre 1^2 en chaînage arrière[1], développé par Alain Colmerauer (voir [vC86] [LK89]).
- Kadviser : moteur d'ordre 1^2 à propagation de contraintes, créé en 1988 par la société KADETECH et développé aujourd'hui par la société NIMTOTH [4].
- MACISTE moteur d'ordre 2^2, développé par Jacques Pitrat (voir [Pit96]).
- SMECI : développé en LISP par la société ILOG (voir [DL94]).
- SNARK : développé par Jean-Louis Laurière (voir [Via85]).

2.1.5 Diagnostic industriel

Historiquement, le raisonnement utilisé en majorité pour faire du diagnostic dans le milieu industriel est le raisonnement à base de règles. Le milieu de l'avionique, et Thales Avionics en particulier, ne déroge pas à cette règle. Prenons le cas des systèmes avioniques et de leur évolution.

Utiliser une approche à base de règles était particulièrement simple et efficace il y a encore quelques années lorsque les systèmes avioniques étaient simples. À l'époque par exemple, une fonction avionique était hébergée sur un composant matériel et cette relation d'association bijective était valable pour la quasi-totalité des fonctions. D'un point de vue maintenance et diagnostic, exprimer les règles était assez simple, du fait de cette bijection entre fonction et composant matériel. Ainsi, une fonction était défaillante si le matériel l'hébergeant l'était, etc.

Avec l'évolution des systèmes, avec la multiplication du nombre de fonctions et avec l'évolution de la technologie en général, les systèmes se sont vus complexifiés. Nous pouvons rencontrer désormais plusieurs fonctions hébergées sur un même matériel, ou même une fonction répartie sur plusieurs. Il y a même des plate-formes de calcul spécifiques dédiées à l'hébergement de fonctions. C'est le cas de ce que l'on appelle la plate-forme *IMA*, signifiant ***I**ntegrated **M**odular **A**vionics*. Malgré cette augmentation de complexité, et par historique, le raisonnement à base de règles reste la principale approche utilisée pour le diagnostic des systèmes avioniques.

Il y a une autre caractéristique à prendre en compte quand on s'intéresse aux méthodes actuellement appliquées en majorité pour le diagnostic industriel. Les systèmes réels, et avioniques en particulier, ont des spécificités qui sont exploitées dans les méthodes de diagnostic actuelles. C'est

1. Le chaînage avant va de la prémisse vers la conclusion alors que le chaînage arrière fait l'inverse.
2. La logique d'ordre 2 permet d'utiliser des variables dans les expressions logiques contrairement à la logique d'ordre 1.
3. voir `http://clipsrules.sourceforge.net`
4. voir `http://www.kadviser.com`

le cas notamment des propriétés de déterminisme, d'indépendance des défaillances et de profondeur de système.

La première caractéristique est le déterminisme. Le non déterminisme n'intervient pas aujourd'hui de manière générale dans la description du comportement des systèmes industriels. Ces propriétés de déterminisme de système sont aujourd'hui utilisées par les techniques de diagnostic industrielles. Néanmoins, s'il n'est pas nécessaire de faire usage de non déterminisme pour décrire le comportement des systèmes actuels, ces derniers s'étoffant, il faut être conscient que cela peut être amené à changer. En effet, la complexité grandissante des systèmes pourrait par exemple nécessiter l'utilisation d'abstractions plus fortes.

Une hypothèse qui est souvent prise est celle d'indépendance des défaillances. En effet, on considère que les défaillances ont des effets indépendants les uns des autres, ce qui fait que quel que soit l'ordre dans lequel arrive les pannes, on obtient les mêmes effets. On considère aussi à travers cela que les défaillances ont des signatures uniques, autrement dit qu'elles n'engendrent qu'un ensemble d'effets. Les effets correspondant souvent aux résultats des envois de messages de monitoring, il est pertinent de considérer que pour une panne donnée, l'ensemble des messages envoyé est toujours le même. Mais on peut très bien imaginer qu'il soit possible qu'une défaillance ait des effets différents quand elle arrive dans un contexte particulier plutôt qu'un autre. En outre, avec la mise en commun des ressources dans les systèmes, cette hypothèse est plus difficile à tenir et la complexification des systèmes risque de rendre invalide les résultats issus des méthodes actuelles, qui utilisent cette hypothèse.

Enfin les systèmes sur lesquels sont faites les analyses de diagnostic sont bien souvent réduits en profondeur et les seuls événements qui sont considérés sont les défaillances du système. Cela signifie que l'on ne considère pas les séquences de défaillances dépassant une certaine longueur. Cette décision n'est pas arbitraire et correspond à une réalité. En effet, la séquence de trois défaillances par exemple représente ce que l'on appelle une *triple panne*. Cela signifie que trois défaillances (indépendantes) sont arrivées dans le système. La décision de ne considérer l'occurrence des défaillances que pour une certaine profondeur maximale du système est cohérente car elle repose sur le fait que plus de pannes sont considérées, moins cela est probable. On considère donc qu'à partir d'une certaine profondeur, la petite probabilité fait que la situation est négligeable.

Nous avons caractérisé dans la partie concernant la topologie du diagnostic (1.3.2), un ensemble des composants qui jouent un rôle dans le diagnostic des systèmes avioniques aujourd'hui. La logique qui régit le comportement de ces composants diagnostic est basée sur des règles.

Les composants qui jouent un rôle de "diagnostiqueurs" locaux ont des tables de règles exprimant les messages émis en fonction des messages reçus telles que la table 2.4 présentée en exemple.

Messages reçus	Diagnostic
Émission défaillante pour $Comp1$	Pas de résultat pour $Comp1$
Valeur hors intervalle pour $Comp1$	
Mauvais format de valeur pour $Comp1$	
$Alim1$ défaillante pour $Comp1$	
$Cable1$ (en sortie de $Comp1$) rompu	
...	...

TABLE 2.4 – Exemple de règles pour diagnostiqueurs locaux (BITE)

Nous avons dans cette table l'exemple d'un composant diagnostiqueur local qui consolide plusieurs messages reçus localement pour n'en envoyer qu'un seul. Ce dernier représentera à l'étage supérieur l'ensemble des messages reçus auquel il est associé.

Le diagnostiqueur global dans nos systèmes avioniques a un fonctionnement proche des diagnostiqueurs locaux et est lui aussi basé sur une table de règles. La différence est que lui fournit un diagnostic de la situation en sortie. La table 2.5 illustre à quoi ressemble les règles pour un diagnostiqueur global.

Cette table est une table messages $\leftrightarrow$ coupables semblable à celle présentée en table 2.2 page 30. La particularité que l'on rencontre est que l'accusation aujourd'hui est restreinte à une disjonction

Messages reçus	Accusation
Pas de résultat pour $Comp1$	$Comp1 \vee Alim1 \vee Cable1$
...	...

TABLE 2.5 – Exemple de règles pour diagnostiqueur global

de trois composants au maximum. C'est ainsi à cause de restrictions au niveau de la communication (nombre de bits alloués pour les messages).

Les limites de l'approche actuelle sont apparues avec la complexification des systèmes. Elles concernent principalement deux points : la construction des tables et la limitation de la taille de l'accusation.

Aujourd'hui, pour construire les tables de règles de diagnostic, il y a une procédure mise en place où chaque expert joue son rôle. L'expert d'un équipement va fournir des tables qui donnent les défaillances possibles de son équipement associées aux messages alors envoyés. L'expert diagnostic va ensuite consolider ces tables et construire la table de règles donnant l'accusation correspondant à chaque message reçu. Mais cette procédure n'est pas suffisante pour rendre le travail relativement facile. La complexité actuelle des systèmes est telle que trouver toutes les causes associées à un message est d'une trop grande difficulté pour l'humain. Ce n'est pas le propre de l'humain de raisonner ainsi, mais celui de l'ordinateur. Il est donc extrêmement difficile de s'assurer que la table de règles est correcte et complète.

S'ajoute à cela les limitations du nombre d'accusés associés à un message. Les accusés qui n'apparaissent pas ne pourront pas faire partie d'un diagnostic. Ainsi, il apparaît que les tables sont forcément incomplètes à cause de cela.

Nous venons d'exposer dans cette section le raisonnement à base de règles qui est l'approche majoritairement utilisée aujourd'hui en industrie pour faire du diagnostic, et avons vu ses limites. Regardons maintenant une autre approche, le raisonnement à base de cas.

2.2 Raisonnement à base de cas (Case-Based Reasoning ou CBR)

2.2.1 Description

2.2.1.1 Introduction au raisonnement à base de cas

Le raisonnement à base de cas est une méthode de résolution de problème qui se démarque des autres approches d'Intelligence Artificielle. Le CBR est capable d'utiliser et de réutiliser les connaissances découlant d'une situation problématique précédente alors que les approches habituelles s'appuient uniquement sur les connaissances générales du domaine du problème ou concluent à partir de sa description.

Le CBR est apparu en réponse aux limites des raisonnements à base de règles. Il apporte une nouvelle base où l'acquisition des connaissances est plus économique et où la mise à jour est facilitée.

Les approches à base de cas sont passées sur le devant de la scène ces dernières années. Le nombre d'articles traitant de ce sujet dans les grandes conférences s'est clairement accru. Des outils basés sur le CBR sont aussi apparus dans le commerce et sont appliqués dans la vie quotidienne (qui ne se rapporte pas à l'industrie).

Si on devait se poser la question "Qu'est-ce que le raisonnement à base de cas ?", on pourrait répondre simplement que c'est résoudre un problème nouveau en se rappelant un cas précédent similaire et en réutilisant l'expérience tirée de cette situation (solution, méthode de résolution, erreurs, etc). Prenons quelques exemples simples de problèmes pour illustrer ces dires :

- Un médecin prend un patient et se rend compte en l'examinant qu'il lui rappelle le cas d'un patient de l'avant-veille. Les symptômes communs aux deux cas étant importants (non pas la couleur des yeux, des cheveux, de la chemise), le médecin va réutiliser le diagnostic et le traitement du patient précédent pour déterminer ce qu'a le patient auquel il est confronté.
- Un joueur de football arrive seul devant le gardien de but adverse lors d'une contre-attaque. La situation dans laquelle il se trouve, celle du gardien et de ces coéquipiers lui rappellent clairement une action du dernier match de l'équipe adverse qu'il a visionné la veille pour préparer la rencontre du jour. Il va donc prendre la même décision qui avait permis à l'attaquant sur la vidéo visionnée de marquer le but.
- Un humoriste en représentation arrive à un endroit précis de son spectacle. Là, une action inattendue d'une personne dans le public le trouble et laisse un blanc dans son spectacle. Il se souvient d'une situation similaire dans une de ses représentations précédentes où il avait alors improvisé et où le public avait réagi positivement. Il décide alors de réutiliser cette improvisation.

Comme ces exemples l'indiquent, le raisonnement à base de cas est puissant et régulièrement utilisé par les humains pour résoudre des problèmes qui se posent, et ce dans de nombreux domaines très différents.

Nous allons dans cette partie, après un bref historique du raisonnement à base de cas, aborder les points de base de ce raisonnement qui sont la résolution de problèmes et l'apprentissage par l'expérience, puis nous allons voir que cette approche fonctionne selon un cycle bien défini, pour finir par citer quelques unes de ses applications.

2.2.1.2 Historique du raisonnement à base de cas

Ian Watson et Fahri Marir s'intéressent dans leur article [WM94] au raisonnement à base de cas et en proposent un historique. Nous allons voir l'apparition du CBR et son évolution tout au long des années passées de manière résumée, pour plus de détails, on se référera à l'article de Ian Watson et Fahri Marir ou encore à l'état de l'art du projet européen SMMART [Mat05].

Le raisonnement à base de cas apparaît vers la fin des années 70 et le début des années 80 à travers des travaux en Intelligence Artificielle et notamment le travail de Roger Schank sur la mémoire dynamique et sur la place centrale de l'expérience des anciennes situations dans la résolution et l'apprentissage de problèmes [Sch82].

D'autres pistes ayant mené au CBR viennent de l'étude du raisonnement analogique [Gen83], et s'appuient aussi sur des théories dans les domaines de la philosophie et de la psychologie, sur la résolution de problèmes et l'apprentissage de l'expérience (e.g. [Wit53, Tul77, SM81]).

Le premier système CBR fut le système CYRUS, développé par Janet Kolodner [Kol83], à l'université de Yale (dans le groupe de travail de Schank). CYRUS était implémenté sur la base du modèle mémoire dynamique de Schank. Ce système a servi de base à plusieurs autres systèmes à base de cas (dont MEDIATOR [Sim85] et PERSUADER [Syc87], deux systèmes de résolution de conflits par médiation, CHEF [Ham86] et JULIA [Hin92], deux systèmes s'intéressant à la planification de processus, et CASEY [Kot89], une application du CBR dans le domaine médical pour le diagnostic de défaillances cardiaques).

Une autre approche et un autre type de modèles ont été développés par Bruce Porter et son groupe de travail à l'université du Texas à Austin [PB86]. Cela a mené au développement du système PROTOS [Bar89] qui avait la particularité de mêler les connaissances générales d'un domaine et les connaissances d'un cas spécifique dans une même structure. Deux ans plus tard, GREBE [Bra91], un système appliqué au domaine de la loi, voyait le jour.

Il n'est pas étonnant, étant donné l'habitude qu'ont les personnes de loi de se référer au passé, que l'on se soit intéressé au CBR dans ce domaine. C'est le cas d'Edwina Rissland et de son groupe de travail à l'université du Massachusetts qui se sont intéressés au rôle du passé dans les jugements juridiques [Ris83]. Les cas (précédents) ne sont pas utilisés pour donner une réponse précise, mais pour interpréter la situation en cours et pour trouver et appuyer des arguments pour les deux

parties. Il en résulta le système HYPO [Ash91] et plus tard le système à la fois à base de règles et à base de cas, CABARET [SR92].

En Europe, les recherches sur le CBR ont mis un peu plus de temps à débuter qu'aux États-Unis. Parmi les premiers résultats il y eut le système MOLTKE, traitant du diagnostic de défaillances, créé par Michael Richter en collaboration avec Klaus Dieter Althoff et quelques autres personnes de l'université de Kaiserslautern [Alt89]. Puis vinrent ensuite d'autres systèmes et méthodes [AW91], ou encore le système PATDEX [RW91] avec Stefan Wess en tant que développeur principal s'intéressant, tout comme le système MOLTKE, à trouver les causes de défaillances.

A la même époque, à l'IIIA de Blanes, Enric Plaza et Ramon Lopez de Mantaras développaient un système d'apprentissage à base de cas pour le diagnostic médical [PdM90]. A Aberdeen, le groupe de travail de Derek Sleeman étudiait l'utilisation de cas pour le raffinement de base de connaissance. On découvrit alors le système REFINER d'aide à l'acquisition de connaissances, développé par Sunil Sharma [SS88], puis le système IULIAN traitant de la planification de processus de raisonnement [Oeh92].

A l'université de Trondheim, Agnar Aamodt et ses collègues de Sintef étudièrent l'aspect apprentissage du CBR dans le contexte de l'acquisition de connaissances en général, et du savoir dans la maintenance en particulier. Pour la résolution de problèmes, l'utilisation combinée des cas et des connaissances générales a été étudiée [Aam89]. Cela a mené au développement du système CREEK [Aam91]. Du côté du Royaume-Uni, le CBR est surtout utilisé dans l'ingénierie civile. Par exemple, un groupe de l'université de Stanford utilise des techniques à base de cas pour le diagnostic de pannes, la réparation et restauration d'immeubles [WA94].

Depuis les années 90, les activités et le nombre d'articles sur le CBR aussi bien aux États-Unis qu'en Europe [Pro91], [IEE92], [Fir93], mais aussi en Israël [Oxm93], en Inde [VKR93] et au Japon [Kit93] ont explosé, et l'intérêt qui est aujourd'hui suscité par le raisonnement à base de cas n'a jamais été aussi important.

2.2.1.3 Résolution de problèmes

Comme nous l'avons remarqué précédemment, raisonner en utilisant les cas du passé est un moyen qui s'avère efficace pour résoudre des problèmes pour les humains. Cette revendication est aussi appuyée par les recherches autour des sciences cognitives. D'ailleurs, différentes études ont révélé le rôle prépondérant des situations du passé, spécifiques, dans la résolution des problèmes psychologiques humains.

Dans la terminologie du CBR, un "cas" représente généralement une situation problématique. Une expérience précédente qui a été enregistrée et apprise de manière à pouvoir être réutilisée pour résoudre des problèmes futurs est appelée cas passé, cas précédent, cas enregistré ou encore cas mémorisé.

De la même façon, un nouveau cas ou cas non résolu désigne un nouveau problème à résoudre.

Le raisonnement à base de cas peut être considéré comme un processus cyclique pour résoudre un problème, apprenant d'une expérience, résolvant un problème... De ce fait, la résolution de problèmes ne désigne pas nécessairement le fait de trouver une solution exacte à un problème d'application, cela peut aussi concerner par exemple le fait de justifier ou infirmer une solution proposée par un utilisateur, le fait d'interpréter une situation problématique ou encore le fait de générer un ensemble de solutions possibles à un problème donné.

2.2.1.4 Apprentissage par l'expérience

L'apprentissage par l'expérience est essentiel au raisonnement à base de cas. La méthode de résolution, seule, indifférente de la façon dont elle apprend de ses cas, n'aurait aucun sens. C'est pourquoi le CBR comprend aussi une machine d'apprentissage qui permet de tenir à jour la base de cas pendant et après la résolution d'un problème. Une fois qu'un problème est résolu, il est appris de manière à pouvoir résoudre un problème similaire à l'avenir. Aussi, lorsque la tentative pour résoudre un problème échoue, la raison de cet échec est identifiée et mémorisée pour éviter le même type d'erreurs dans le futur.

Le CBR favorise l'apprentissage par l'expérience, puisqu'il est généralement plus facile d'apprendre en mémorisant l'expérience d'un problème résolu plutôt qu'en généralisant à partir d'elle. La partie apprentissage du raisonnement nécessite un ensemble de méthodes bien travaillées pour extraire les connaissances pertinentes de l'expérience, pour intégrer un cas dans une structure de connaissance existante, et pour indexer le cas pour pouvoir faire la correspondance avec les cas futurs similaires.

2.2.1.5 Cycle de fonctionnement du raisonnement à base de cas

2.2.1.5.1 Cycle CBR

Un cycle CBR décrit les quatre processus suivants :

- Récupérer le ou les cas passés les plus similaires.
- Réutiliser les informations et la connaissance pour résoudre le problème.
- Réviser la solution proposée.
- Apprendre les parties de l'expérience utiles pour la résolution de problèmes futurs.

La méthode de résolution d'un nouveau problème est donc la suivante : après avoir récupéré un cas similaire, on le réutilise d'une manière ou d'une autre, puis on vérifie la solution déduite, pour finalement l'apprendre sous forme d'expérience dans la base de cas. Les quatre phases ont chacune différentes étapes spécifiques qui vont être décrites.

On peut voir une représentation de ce cycle sur la figure 2.4.

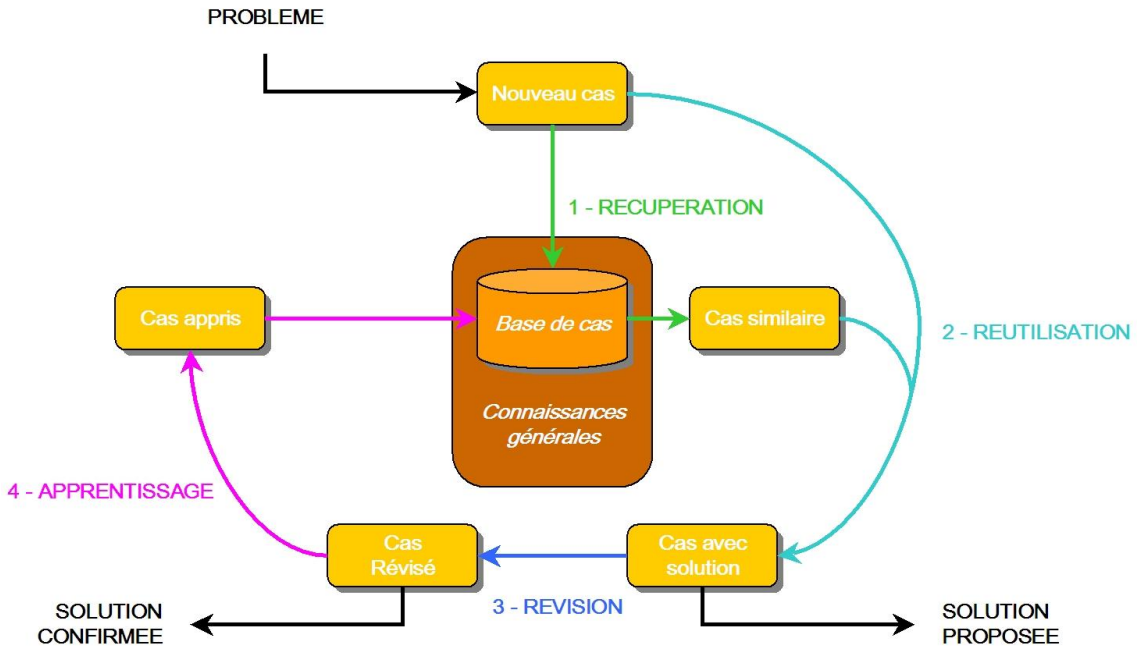


FIGURE 2.4 – Cycle CBR

On commence avec une description initiale du problème qui définit un nouveau cas. À partir de là, on récupère le cas le plus similaire de la base de cas. On réutilise alors le cas similaire en le combinant avec le nouveau cas pour obtenir un cas avec solution. La solution proposée est testée pendant la phase de révision, en l'appliquant au monde réel et en la réparant s'il le faut. Finalement, avec l'expérience du nouveau cas, la base de cas est mise à jour par l'ajout du cas appris ou par modification de cas déjà existants.

Les phases du cycle les plus compliquées sont probablement les phases de récupération et de révision. Récupérer un cas similaire à un nouveau problème implique le fait d'avoir une fonction de

similarité pour comparer la description du nouveau cas avec les anciens cas, qui n'est pas toujours évidente à trouver. Les caractéristiques du cas passé et du cas actuel n'étant pas forcément les mêmes, il peut s'avérer difficile de trouver comment vérifier la solution.

2.2.1.5.2 Représentation des cas

Savoir représenter les cas est une question essentielle pour le raisonnement à base de cas. Il y a un réel besoin d'efficacité pour la récupération et la réutilisation de cas, c'est pourquoi il est important de décider ce qui doit être stocké dans un cas, de trouver une structure adéquate pour décrire le contenu des cas, et de décider comment la mémoire de cas doit être organisée et indexée pour assurer une récupération et une réutilisation efficaces.

On peut distinguer trois parties dans la représentation de cas :

- La description du cas, qui est la définition des caractéristiques du problème.
- La solution du cas, qui est la description de la manière dont le problème représenté par la description du cas a été résolu dans le passé.
- Le résultat du cas, qui exprime quel type de changements a provoqué la solution sur l'environnement du problème.

Prenons l'exemple d'un cas résolu à représenter : Le remplacement du processeur d'un PC par un plus puissant a permis de gérer la réalisation de gros calculs pour la résolution de grilles de sudoku, augmentant la puissance du PC. Notre cas peut être représenté de la manière suivante selon les trois parties :

- "Gérer la réalisation de gros calculs pour la résolution de grilles de sudoku" correspond aux caractéristiques du problème.
- "Remplacer le processeur d'un PC par un plus puissant" correspond à la solution du problème.
- "Augmenter la puissance de calcul du PC" correspond au résultat du cas.

Le cas est souvent symbolisé par une liste de couples attribut/valeur. Choisir les attributs importants pour représenter le cas est sans doute l'activité la plus difficile et la plus coûteuse en temps, étant donné que les attributs sont la base de la comparaison des cas. En outre, il est possible que la difficulté augmente dans le cas où l'ensemble des attributs importants change en fonction de l'état de l'environnement du problème.

2.2.1.5.3 Description des phases du cycle

2.2.1.5.3.1 La récupération de cas

La récupération de cas commence avec une description du problème et se termine quand un cas précédent qui correspond bien est trouvé. Les tâches la composant sont l'identification du problème, la recherche de correspondances et la sélection d'un cas précédent, exécutées dans cet ordre.

La première tâche est donc l'identification du problème. Identifier un problème revient simplement à donner ses caractéristiques.

La deuxième tâche a pour but de récupérer un ensemble de candidats possibles à la correspondance. Pour ce faire, on utilise les descripteurs du problème (ou caractéristiques du problème) comme index à la mémoire de cas. Il y a en principe trois façons de récupérer un cas ou un ensemble de cas : soit en suivant directement les pointeurs des index depuis les caractéristiques du problème, soit en cherchant dans une structure d'index, soit en cherchant dans un modèle des connaissances générales du domaine. On peut aussi pondérer les descripteurs du problème selon leur aptitude à caractériser le problème durant la phase d'apprentissage.

La troisième et dernière tâche consiste à choisir le meilleur cas parmi ceux sélectionnés (il est rare que cela ait été fait par la tâche précédente). Le cas qui correspond le mieux est souvent déterminé en évaluant plus précisément le degré de correspondance initial. Si une correspondance s'avère ne pas être assez forte, une tentative est faite pour en trouver une meilleure, en se tournant

vers les cas qui lui sont proches. Le processus de sélection génère les conséquences et les attentes de chaque cas récupéré et essaie d'évaluer ces conséquences et de justifier ces attentes. Cela peut être fait au moyen du modèle du système représentant les connaissances générales du domaine, ou bien en le demandant à l'utilisateur. Ensuite, les cas peuvent être éventuellement classés d'après certains critères.

2.2.1.5.3.2 La réutilisation de cas

Pour réutiliser le cas récupéré, on regarde deux aspects :

1. Quelles sont les différences entre le cas passé et le cas présent ?
2. Quelle partie du cas récupéré peut être utilisée pour le nouveau cas ?

Dans le cas de classification simple, les différences sont très résumées (elles sont considérées comme non pertinentes tant que les similarités, elles, le sont) et la solution du cas récupéré devient celle du nouveau cas. C'est une façon triviale de faire de la réutilisation. Cependant, d'autres systèmes sont obligés de prendre en compte les différences 1., et par conséquent, la partie réutilisée 2. ne peut pas être directement transférée au nouveau cas mais nécessite une phase d'adaptation qui prend en compte ces différences.

Il y a principalement deux façons de réutiliser les cas passés :

- Réutiliser la solution du cas passé (réutilisation transformationnelle).
- Réutiliser la méthode qui a construit la solution du cas passé (réutilisation dérivationnelle).

Dans la réutilisation transformationnelle, la solution du cas passé n'est pas directement la solution pour le nouveau cas, mais il existe des connaissances sous forme d'opérateurs transformationnels telles qu'appliquées sur la solution passée, elles la transforment en une solution pour le nouveau cas. Une manière d'organiser ces opérateurs est de les classer autour des différences détectées entre le cas récupéré et le cas courant.

La réutilisation dérivationnelle, elle, regarde comment le problème a été résolu dans le cas récupéré. Ce dernier garde des informations sur la méthode employée pour résoudre le problème qui lui est associé, incluant une justification des opérateurs utilisés, des sous-objectifs considérés, des alternatives générées, des chemins de recherches ayant échoué, etc. La méthode récupérée est alors réutilisée pour établir un plan de résolution du nouveau cas et l'appliquer dans le nouveau contexte.

Finalement, la réutilisation propose une solution.

2.2.1.5.3.3 La révision de cas

La solution créée par la phase de réutilisation doit maintenant être testée dans l'environnement du cas. Cette phase est appelée révision de cas et consiste en deux tâches :

- On évalue la solution réutilisée. Si cela fonctionne, on apprend de ce succès (et on passe dans la phase d'apprentissage de cas).
- Sinon, on la répare en utilisant les connaissances spécifiques du domaine.

La tâche d'évaluation se sert du résultat de l'application de la solution dans l'environnement réel du nouveau cas. C'est habituellement une étape extérieure au système CBR, car elle implique l'application d'une solution proposée à un problème réel. Une solution peut aussi être appliquée à un programme de simulation capable de reproduire l'environnement réel du problème. Dans un système de support de décision, le succès ou l'échec d'un traitement peut aller de quelques secondes à plusieurs minutes, mais le cas peut toujours être appris et être disponible dans la base de cas pendant ce temps, il suffit qu'il soit marqué comme non-évalué.

Pour réparer un cas il est nécessaire de pouvoir détecter les erreurs sur la solution courante et de pouvoir récupérer ou générer des explications pour ces erreurs. Le meilleur exemple est le système où les connaissances causales sont utilisées pour générer une explication de la raison pour laquelle certains buts du plan de résolution n'ont pu être atteints (ce sont des échecs). Grâce à une technique d'apprentissage basée sur l'explication, les situations qui causent des erreurs sont mémorisées et sont ajoutées à une mémoire d'échecs utilisée pour prévoir les échecs éventuels durant la phase de réutilisation.

Si la solution ne passe pas l'évaluation, on passe dans la seconde partie de la phase de révision, la réparation de la solution. En utilisant les explications fournies, les échecs rencontrés durant la tâche d'évaluation vont pouvoir être évités. Par exemple, joindre un module de réparation qui ajoute des étapes au plan de résolution tout en s'assurant que les causes des échecs n'interviendront plus, peut être une réparation. Le module de réparation possède des connaissances générales causales et des connaissances du domaine qui expliquent comment éviter ou compenser les causes des erreurs dans le domaine. Le plan revu sera directement appris si la phase de révision assure qu'il est correct, ou bien sera peut-être évalué et réparé à nouveau.

2.2.1.5.3.4 L'apprentissage de cas

L'apprentissage est la phase terminale de chaque problème. Une fois le problème résolu, ce processus incorpore ce qu'il est utile de conserver de la solution du nouveau problème pour la faire devenir connaissance. L'apprentissage à partir de l'échec ou du succès de la solution proposée est déclenché durant la phase de révision soit par le résultat de l'évaluation, soit suite à la réparation. Il faut donc savoir quelles informations il faut retenir du cas, sous quelle forme, comment indexer le cas pour le retrouver plus tard à partir de problèmes similaires, et comment intégrer le nouveau cas dans la structure mémoire. Ce sont des points essentiels dans ce raisonnement puisque tout repose sur la base de connaissances.

Dans le CBR, la base de cas est mise à jour qu'importe la manière utilisée pour résoudre le problème. S'il a été résolu en utilisant un cas précédent, le nouveau cas peut être construit et ajouté à la base, ou alors le cas passé peut aussi bien être généralisé pour englober le cas courant. Le problème peut encore être résolu par d'autres moyens, comme par exemple par l'utilisateur, et cela doit apparaître dans la base, sous la forme d'un nouveau cas par exemple. En tout cas, il est nécessaire d'identifier ce qui va être la source de l'apprentissage. Des candidats évidents sont les descripteurs et les solutions du problème, mais une explication ou une autre forme de justification de la solution peut aussi être intégrée lors de l'ajout d'un nouveau cas.

Comme nous venons de l'expliquer, des explications peuvent être incluses dans les cas appris, et réutilisées plus tard pour modifier la solution. En effet, lorsque nous avons besoin d'explications pour un nouveau cas, il est donc possible de réutiliser les explications apprises d'anciens problèmes plutôt que de les chercher dans le modèle entier du domaine, ce qui serait beaucoup plus long. En plus des explications, on peut aussi retenir la méthode de résolution du problème, ce qui rend le système approprié pour la réutilisation dérivationnelle.

Les échecs, c'est-à-dire les informations issues de la tâche de révision, peuvent aussi être extraits et appris, soit en tant que cas séparés (cas d'échec), soit intégrés à d'autres cas problématiques. Lorsqu'un échec est rencontré, le système peut alors s'appuyer sur un précédent échec de nature similaire et l'utiliser pour améliorer sa compréhension de l'échec actuel (et le corriger).

Le problème d'indexation est un problème clé du raisonnement à base de cas. Cela consiste à choisir quel est le type d'index qui va être utilisé pour de futures récupérations, et comment structurer l'espace de recherche de ces index. Ce problème d'acquisition devrait être analysé comme une partie de l'étape d'analyse des connaissances du domaine et de modélisation. Une solution intuitive à ce problème est d'utiliser comme indices les caractéristiques d'entrée du problème. Dans la littérature, trois catégories d'indexation se démarquent :

- L'indexation formelle : elle insiste sur l'exactitude et la conformité de la solution du cas à utiliser. Cette approche se base sur le fait que la correspondance entre le cas récupéré et le cas source ne peut se réaliser de façon efficace et sans ambiguïté que si le cas est représenté en utilisant un symbolisme mathématique stricte.

- L'indexation par mots clés : à la différence de l'indexation formelle, l'indexation par mots clés utilise une représentation textuelle pour la description du cas. Les cas sont indexés et classés dans la base de cas suivant les mots clés contenus dans leur description. Pour effectuer une recherche, un analyseur de texte extrait les mots clés contenus dans la description du problème faite par l'utilisateur en langage naturel et ces mots sont utilisés pour localiser les cas dont la description contient ces mots.
- L'indexation ontologique : elle est actuellement la plus utilisée et la plus prometteuse pour l'indexation des cas [CCYB95]. Au lieu de baser la similarité entre cas sur une correspondance syntaxique, l'approche ontologique met en exergue la correspondance sémantique entre les différents concepts sélectionnés par l'utilisateur et ceux contenus dans la description du cas. Cette ontologie doit définir les concepts importants du domaine et les relations entre ces concepts. Les concepts issus de l'ontologie sont utilisés pour indexer les différents cas stockés.

L'intégration est l'étape finale de la mise à jour de la base de connaissances avec un nouveau cas. Et si aucun nouveau cas et aucun ensemble d'index n'a été construit, c'est même la principale étape de l'apprentissage. Cette étape a pour but de mettre à jour les index des cas existants et ainsi affiner l'évaluation de similarité. La qualité et l'importance d'un index pour un cas particulier ou une solution sont ajustées par rapport au succès ou à l'échec de l'utilisation du cas pour résoudre le problème de départ. Concernant les caractéristiques qui se sont révélées utiles pour récupérer un cas fructueux, leur association avec le cas est renforcée, tandis qu'il est affaibli pour les caractéristiques qui ont fait échouer la récupération de cas. De ce fait, la structure d'index a pour rôle d'adapter la mémoire de cas à son utilisation.

2.2.1.6 Applications du CBR

Le CBR est appliqué dans des domaines variés comme par exemple :

- L'analyse financière pour la gestion des risques.
- La maintenance technique.
- L'interprétation de photographies.
- Les sciences de l'information géographique pour prédire la pollution de l'air, pour la sélection d'images satellites, pour les prédictions météorologiques ...

Le CBR est aussi implanté (développement ou utilisation) dans de nombreuses entreprises dont par exemple :

- Black & Decker pour le service client hotline.
- Blue Cross pour le diagnostic médical.
- British Airways pour la maintenance des avions.
- Microsoft pour son assistant utilisateur pour Microsoft Office.
- L'institut Français du Pétrole (IFP) pour la sélection de lubrifiants.
- La NASA pour l'aide à la décision pour l'atterrissage de navettes.

2.2.2 Illustration du diagnostic à base de cas

Dans cette partie nous allons montrer l'utilisation du diagnostic à base de cas sur l'Exemple introduit dans la section 2.1.2.

Nous nous référerons donc au système en figure 2.1 dans un premier temps pour désigner le système considéré.

Nous allons voir ici le diagnostic d'un problème par réutilisation de la méthode de résolution d'un cas similaire et par réutilisation de la solution d'un cas similaire, puis nous verrons comment la base de cas peut être à la fois avantageuse et défavorable.

id	Nb_cas	Observations	Accusation	Méthodes utilisées	Solutions
1	4	$(\neq, [],][)$	$S_2.E$	F_2 montre $][$ donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
2	5	$(\neq, nd, [])$	$S_1.L$	F_1 montre nd donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
3	3	$(\neq, [], nd)$	$S_2.L$	F_2 montre nd donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
4	6	$(none, [], [])$	$C.L$	Test de C. Réparation C et re-test $\rightarrow$ OK	Changer C
5	9	$(\neq, [], [])$	$S_1.E$	C montre $\neq$ alors test avec autre sonde que S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1

TABLE 2.6 – Base de cas 1

Introduisons la base de cas telle que représentée en table 2.6.
Présentons les différentes colonnes :

- La colonne "Nb cas" référence le nombre de fois où le cas a été rencontré.
- Les observations correspondent à ce que nous avons vu dans la section 2.1.2.2.2.
- Les accusations désignent les composants responsables du problème, ainsi que l'état dans lequel ils sont. C'est une formule logique.
- Les méthodes utilisées sont les méthodes employées pour résoudre le problème.

Considérons une première situation. Nous avons l'observation suivante : $(\neq, [], nd)$. Il se trouve que cette situation a déjà été observée à la ligne dont l'identifiant est 3. On va donc réutiliser la solution du cas passé et changer S_2. Le cas maintenant résolu est appris. Nous obtenons désormais une nouvelle table (table 2.7) intégrant les résultats du diagnostic que nous venons d'effectuer.

id	Nb_cas	Observations	Accusation	Méthodes utilisées	Solutions
1	4	$(\neq, [],][)$	$S_2.E$	F_2 montre $][$ donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
2	5	$(\neq, nd, [])$	$S_1.L$	F_1 montre nd donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
3	4	$(\neq, [], nd)$	$S_2.L$	F_2 montre nd donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
4	6	$(none, [], [])$	$C.L$	Test de C. Réparation C et re-test $\rightarrow$ OK	Changer C
5	9	$(\neq, [], [])$	$S_1.E$	C montre $\neq$ alors test avec autre sonde que S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1

TABLE 2.7 – Base de cas 2

Maintenant une deuxième situation dans laquelle on observe : $(\neq,][, [])$. Considérons que la fonction de similarité que nous avons définie auparavant nous renvoie le cas d'identifiant 1. On réutilise alors la méthode de résolution du cas 1 : nous avons F_1 qui nous montre $][$, donc nous réparons S_1 et comparons sa nouvelle valeur avec S_2. Elles concordent donc le problème est résolu et le cas est appris. Nous obtenons avec l'apprentissage une nouvelle base (table 2.8) qui intègre le cas que nous venons de rencontrer pour la première fois.

Voyons le cas d'une base de cas défavorable. Considérons le cas où S_2 est défaillant et où on observe $(\neq, [], [])$. Etant donné que seul le cas où S_1 était défaillant est apparu, le diagnostic à base de cas va déduire de la base qu'il faut réparer S_1. Après l'avoir réparé, le problème sera identique et l'ancienne sonde aura été remplacée pour rien. Cet exemple, où l'utilisation de la base de cas

id	Nb_cas	Observations	Accusation	Méthodes utilisées	Solutions
1	4	$(\neq, [],][)$	$S_2.E$	F_2 montre $][$ donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
2	5	$(\neq, nd, [])$	$S_1.L$	F_1 montre nd donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
3	4	$(\neq, [], nd)$	$S_2.L$	F_2 montre nd donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
4	6	$(none, [], [])$	$C.L$	Test de C. Réparation C et re-test $\rightarrow$ OK	Changer C
5	9	$(\neq, [], [])$	$S_1.E$	C montre $\neq$ alors test avec autre sonde que S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
6	1	$(\neq,][, [])$	$S_1.E$	F_1 montre $][$ donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1

TABLE 2.8 – Base de cas 3

est mauvaise, va nous obliger à diagnostiquer le problème manuellement. Après résolution de ce problème, le cas est tout de même appris et la base est mise à jour. Nous pouvons voir cela en table 2.9.

id	Nb_cas	Observations	Accusation	Méthodes utilisées	Solutions
1	4	$(\neq, [],][)$	$S_2.E$	F_2 montre $][$ donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
2	5	$(\neq, nd, [])$	$S_1.L$	F_1 montre nd donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
3	4	$(\neq, [], nd)$	$S_2.L$	F_2 montre nd donc réparation S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2
4	6	$(none, [], [])$	$C.L$	Test de C. Réparation C et re-test $\rightarrow$ OK	Changer C
5	9	$(\neq, [], [])$	$S_1.E$	C montre $\neq$ alors test avec autre sonde que S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
6	1	$(\neq,][, [])$	$S_1.E$	F_1 montre $][$ donc réparation S_1 et comparaison avec $S_2 \rightarrow$ OK	Changer S_1
7	1	$(\neq, [], [])$	$S_2.E$	C montre $\neq$ alors test avec autre sonde que S_2 et comparaison avec $S_1 \rightarrow$ OK	Changer S_2

TABLE 2.9 – Base de cas 4

Néanmoins, maintenant que le cas a été rencontré, on peut observer un avantage de la base de cas. En effet, si l'on rencontre à nouveau le cas d'observation $(\neq, [], [])$, nous allons avoir les deux cas possibles expliquant la panne répertoriée, mais nous allons avoir en plus l'information comme quoi la solution la plus probable est celle où S_1 est dans l'état ERR puisqu'elle a été rencontrée plus de fois que celle où c'est S_2 qui est dans l'état ERR.

Sur cet exemple simple, nous avons déjà pu voir plusieurs utilisations de la base de cas pour le diagnostic d'une situation problématique. On perçoit aussi comment le fait de ne pas avoir une base bien construite peut amener des erreurs lors des diagnostics, et comment le contraire peut apporter indéniablement un avantage.

2.2.3 Avantages et limites

Les techniques de raisonnement à base de cas ne nécessitent aucun modèle et sont capables d'acquérir des nouvelles connaissances dans l'optique d'avoir des diagnostics de défaillances plus précis. Le CBR permet de résoudre des problèmes même si les connaissances du domaine sont

incomplètes. La chose la plus importante est de savoir comment comparer deux cas. On peut lister quelques avantages :

- Le CBR permet de prendre des raccourcis dans le raisonnement. En effet, si un cas adapté est trouvé, une solution sera proposée rapidement.
- Le CBR peut amener à une capacité d'explication améliorée dans des situations où les explications les plus compréhensibles sont celles qui impliquent des exemples spécifiques.
- Le CBR peut aider aussi à éviter les erreurs du passé et apprendre des erreurs et des succès. En effet, les systèmes peuvent garder une trace de chaque situation qui apparaît pour permettre de s'y rapporter dans le futur.
- Le CBR est particulièrement utile lorsque les règles du domaine sont difficiles et coûteuses à exprimer. La plupart des domaines du monde réel sont si complexes qu'il n'est pas possible ou pas pratique de spécifier complètement toutes les règles impliquées.
- La création d'une base de connaissances est généralement plus rapide que la création d'une base de règles.
- Les systèmes à base de cas sont facilement maintenus et augmentés par l'ajout de cas.

D'un autre côté, le raisonnement à base de cas connaît aussi des limites :

- La validation et la vérification de systèmes à base de cas est un processus difficile.
- La base de connaissances est vide au départ. La méthode n'utilise pas beaucoup les systèmes d'informations.
- Le CBR récupère le cas le plus similaire et essaie de réutiliser la solution qui lui est associée. Le CBR ne fournit pas de solutions précises, exactes, ou optimales.
- La caractérisation des critères est une étape primordiale du raisonnement à base de cas et est très ardue.

Une autre limitation du CBR est qu'il devient de plus en plus difficile de trouver des cas similaires lorsque le niveau de précision des correspondances augmente.

2.2.4 Outils

Nous avons cité dans l'historique du raisonnement à base de cas, de nombreux outils qui ont été développés. On peut citer quelques autres exemples d'outils :

- Caspian (U. Wales) [5]
- CBR Shell (AIAI, Stuart Aitken) [6]
- CBR*Tools (INRIA) [7]
- FreeCBR, open source [8]
- jCOLIBRI (U. Madrid) [9]
- myCBR (German Research Center for Artificial Intelligence DFKI GmbH) [10]
- Shyster : Un système expert juridique à base de cas (James Popple, 3/99) [11]
- SMILA [12]

5. voir `http://www.aber.ac.uk/~dcswww/Research/mbsg/cbrprojects/getting_caspian.shtml`
6. voir `http://www.aiai.ed.ac.uk/project/cbr/CBRDistrib`
7. voir `http://www-sop.inria.fr/axis/cbrtools/manual/`
8. voir `http://freecbr.sourceforge.net`
9. voir `http://gaia.fdi.ucm.es/projects/jcolibri`
10. voir `http://mycbr-project.net`
11. voir `http://cs.anu.edu.au/software/shyster`
12. voir `http://www.eclipse.org/smila`

2.3 Raisonnement à base de modèle (Model-Based Reasoning ou MBR)

2.3.1 Description

2.3.1.1 Introduction au raisonnement à base de modèle

Le raisonnement à base de modèle est une méthode de résolution de problème qui s'appuie sur un modèle représentant le système à diagnostiquer. En effet, le modèle est une description du système réel et est utilisé pour détecter et localiser les pannes.

Le MBR a fait son apparition durant les 70 et le début des années 80, soit à peu près à la même époque que le raisonnement à base de cas.

Si on devait résumer simplement la façon la plus répandue de raisonner en se basant sur un modèle, on pourrait dire qu'à partir d'un modèle représentant le système à diagnostiquer et d'un ensemble d'observations démontrant un comportement non attendu du système (déviant du comportement nominal), le but du diagnostic à base de modèle est d'identifier les causes de ces problèmes en utilisant un modèle du système.

Nous allons dans cette partie présenter le raisonnement à base de modèle en commençant par un historique de cette approche, puis nous allons en donner le principe général, décrire quelques-unes des façons de l'utiliser et traiter un peu plus de la phase importante qu'est la modélisation. Nous continuerons en vous proposant une méthode de diagnostic puis en illustrant cette technique sur un exemple afin d'avoir une idée de comment cette approche de diagnostic peut être utilisée. Finalement, nous ferons une critique de cette approche.

2.3.1.2 Historique du raisonnement à base de modèle

Cet historique a été rédigé à partir des documents [CDL+04, Rob03, Moz92, CT91, CB05, LA97, Con94, Sab06, Fra90, Che08]. On pourra les consulter pour avoir plus de détails.

Il y a deux communautés différentes à la source du diagnostic à base de modèles actuel : la communauté Automatique (Contrôle) et la communauté Intelligence Artificielle (IA). En effet, des chercheurs des deux domaines différents se sont intéressés au diagnostic de fautes (ou défaillances), mais ce, séparément. Voyons donc en bref comment ils ont évolué.

En ce qui concerne le monde de l'automatique, cela a débuté au début des années 70 avec le développement et l'application du contrôle, de la sécurité et de la sûreté de systèmes. Par exemple, en 1971, Dr Beard [Bea71] du MIT s'est intéressé à l'idée d'apporter une alternative à la redondance matérielle coûteuse utilisée alors pour augmenter la sûreté des systèmes. Il introduisit alors la redondance analytique, largement utilisée aujourd'hui. La première revue de cet article de Beard est publiée en 1976 dans le magazine Automatica par Alan S. Willsky [Wil76].

A cette époque, quelques algorithmes simples ont été proposés, mais la théorie sur le diagnostic de fautes n'était pas mûre et les cas d'application étaient rares. En effet, les recherches industrielles de l'époque étaient quasi inexistantes. Les rares intérêts de l'époque se trouvaient dans le milieu de l'aéronautique. On peut citer le travail de J. Deckert et ses collègues (dont Alan S. Willsky) qui publient en 1977 un article sur leurs travaux concernant le diagnostic de défaillances de capteurs à l'aide de redondance analytique sur le F-8 DFBW (Digital Fly-By-Wire) de la NASA [DDDW77], que l'on peut apercevoir en figure 2.5.

La période des années 70-80 fut donc dédiée au mûrissement de la théorie sur le diagnostic de fautes. De nouvelles théories apparaissent dont notamment : le filtre de détection [Bea71] [Jon73], les tests utilisant un unique filtre de Kalman [MP71], ou bien utilisant plusieurs filtres de Kalman ou observateurs de Luenberger [CFW75] [MC74], l'approche espace de parité [DDDW77], la technique d'estimation de paramètres [Kit80], etc. Pour trouver plus de détails ou plus d'informations sur les théories citées, on pourra aussi se référer à l'article de Paul M. Franck de 1990 [Fra90] ou à celui d'Alan S. Willsky de 1976 [Wil76].

FIGURE 2.5 – F-8 DFBW (de `http://www.allstar.fiu.edu/aero/gallery4.htm`)

Ce sont ces approches qui fondent la théorie du domaine du contrôle automatique sur le diagnostic de fautes.

Les chercheurs du domaine de l'intelligence artificielle ont commencé à s'intéresser au diagnostic à base de modèles un peu plus tard que leurs collègues du contrôle automatique. En effet, ils ne s'y sont intéressés qu'à partir du moment où ils se sont rendus compte que le raisonnement simple à base de règles comportait trop de limites pour les systèmes actuels évoluant.

Les recherches commencent donc dans le début des années 80, et si on analyse les formalisations logiques du diagnostic à base de modèle, on distingue immédiatement deux façons de voir les choses : d'un côté on a un diagnostic basé sur la cohérence (raisonnement consistant à vérifier la cohérence entre les solutions et l'état du système) et de l'autre un diagnostic basé sur l'abduction (raisonnement consistant à supprimer les solutions improbables) ; d'un côté on a des modèles de bon comportement du système, et de l'autre des modèles de dysfonctionnement.

Les approches basées sur la cohérence ont été originellement proposées en association avec le comportement correct des systèmes à diagnostiquer. Des chercheurs comme Davis [Dav84] ou Genesereth [Gen84] en 1984, et de Kleer et Williams [dKW87] ou encore Reiter [Rei87] en 1987, font partie de ceux qui ont proposé ce lien entre cohérence et bon comportement. De ce point de vue, le diagnostic désignera un ensemble de composants suspectés d'être défaillants de telle manière que les observations soient cohérentes avec les composants non défaillants.

Les approches abductives, elles, ont été associées à l'origine avec le comportement défaillant des systèmes à diagnostiquer. Cette association a été faite entre autres par Cox et Pietrzykowski en 1986 [CP86], Poole et ses collègues de l'Université de Waterloo au Canada en 1987 [PGA87], ou encore Console et ses collègues de l'Université de Turin en 1989 [CePT89a, CePT89b]. De ce point de vue, le diagnostic désignera un ensemble de composants suspectés d'être défaillants de telle manière que les observations soient des conséquences de ces composants désignés.

Ces liens qui avaient été établis comme naturels entre d'un côté cohérence et bon comportement, et de l'autre côté abduction et modèles de défaillances, ont aiguisé l'esprit de certains chercheurs. Poole explique en 1989 [Poo89] que les approches abductives et à base de cohérence ne portent pas sur une même logique, et qu'il n'y a pas "une unique vraie définition logique du diagnostic". Hamscher en 1991 [Ham91], et de Kleer et Williams [dKW89] d'une part, ainsi que Struss et Dressler [SD89] d'autre part, montrent en 1989 les avantages qu'il y a à combiner modèles de fautes et modèles de bon comportement.

Pour plus de détails sur l'approche à base de modèles du point de vue intelligence artificielle, on peut se référer au livre "Readings in Model-Based Diagnosis" par Walter Hamscher, Luca Console et Johan de Kleer en 1992 réunissant une quarantaine d'articles sur le diagnostic à base de modèle [HCdK92].

Les deux communautés, DX (*Diagnosis*) qui vient du domaine de l'Intelligence Artificielle, et FDI (*Fault Detection and Isolation*) qui vient du domaine du contrôle automatique, se sont établies au fur et à mesure des années. Chacune a développé ses propres concepts, outils et techniques, chacune utilise sa propre terminologie et son propre ensemble de conférences et publications. Après quelques longues années à travailler en parallèle, les deux communautés se sont rapprochées pour travailler ensemble, par exemple lors de conférences comme *DX* (*International Conference on Principles of Diagnosis*) ou encore *SAFEPROCESS*. Un article d'un groupe de chercheurs de différentes universités françaises et venant des deux communautés [CDL$^+$04] traite de ce rapprochement et démontre, entre autres choses, que les résultats fournis par l'approche DX à base de cohérence d'une part, et par l'approche FDI à base de redondance analytique d'autre part, sont identiques (sous condition de complétude). On peut se référer aussi à [CDL$^+$00b, CDL$^+$00a, CPVG05, BCL$^+$04].

Durant les années 70 et 80, plusieurs systèmes de diagnostic à base de modèle ont été conçus. La plupart d'entre eux s'intéressaient à la localisation de fautes sur les circuits électriques (analogiques et digitaux) comme INTER [dK76] et WATSON [Bro76] en 1976, SOPHIE [Bro82] et HT [DSH$^+$82] en 1982, ou encore GDE [dKW87] en 1987. Il y avait aussi ABEL [PSS81] en 1981 qui portait sur le diagnostic médical tout comme LOCALIZE [FHMM82] en 1982 traitant de la localisation des lésions du système nerveux périphérique. Pendant cette période, on trouve encore par exemple LES/LOX en 1985 qui s'intéressait au diagnostic de fautes au niveau du chargement d'oxygène liquide pour les navettes spatiales.

Il semble aujourd'hui que les axes de travail se dirigent vers l'application de l'approche à base de modèles au monde industriel. En effet, parmi les différents sujets traités lors de la conférence DX'09, il est ressorti que les théories de diagnostic n'ont pas rencontré un grand succès dans l'industrie, ce problème venant sans doute de la diversité des problèmes et de l'application à des périmètres restreints des algorithmes généralement proposés. Dans l'optique d'accélérer la recherche concernant le diagnostic des systèmes complexes et pour encourager le développement de plateformes à cet effet, le centre de recherche AMES de la NASA a décidé de lancer une compétition annuelle qui aura lieu durant la conférence DX (voir `http://www.dx-competition.org`). A noter que pour la première édition de cette compétition, ce qui dépeint bien la situation, le logiciel qui a remporté la compétition se trouve être une solution non académique.

2.3.1.3 Principe général du diagnostic à base de modèle

Dans cette partie, nous allons présenter le principe général du diagnostic à base de modèle, c'est-à-dire les bases qui sont communes aux différentes méthodes reposant sur le raisonnement à base de modèle et utilisées pour faire du diagnostic.

La première nécessité quand on veut faire du diagnostic à base de modèle sur un système, c'est d'avoir un modèle du système. Faire le modèle d'un système, c'est représenter les relations entre les composants du système et leur comportement. La modélisation est objective car elle n'a qu'à représenter la réalité des choses sans les interpréter. Le modèle représente les vrais échanges entre les composants, et c'est une des principales raisons pour son utilisation. Le modèle n'est pas donc forcément spécifique au diagnostic, mais peut découler par exemple de la modélisation d'un concepteur de systèmes.

La deuxième nécessité pour faire du diagnostic à base de modèle c'est de disposer d'un modèle faisant le lien entre les observations reçues et les causes qui ont pu les engendrer. Il faut que cette relation soit formalisée pour pouvoir être exploitable par des outils d'analyse automatique.

Le but du modèle est de capturer la connaissance humaine des systèmes et potentiellement de leur comportement dynamique.

2.3.1.4 Utilisations variées

Le raisonnement à base de modèle a quelques propriétés très puissantes, comme par exemple le fait qu'il soit utilisable dans plusieurs domaines différents, ou bien encore le fait qu'il permette de choisir un certain type de modèle suivant l'utilisation que l'on veut en faire.

Une autre des plus puissantes caractéristiques du raisonnement à base de modèle est qu'une fois réalisé, le modèle peut être utilisé pour plusieurs tâches différentes, comme par exemple :

- Le design qui s'intéresse à l'architecture, à l'aspect de conception du système.
- La sûreté de fonctionnement qui s'intéresse à la fiabilité du système.
- La simulation qui s'intéresse à la reproduction du comportement du système.
- La surveillance qui s'intéresse à l'observabilité et à l'observation du système.
- Le diagnostic et le pronostic.

Le MBR permet la caractérisation d'un modèle. En effet, à travers les différents types de modèles, différents aspects sont abordés. Le choix du type de modèle se fait donc selon différents facteurs, comme par exemple :

- Quelles informations sont disponibles ?
- Quels est l'utilisation du modèle ?
- Quelles observations sont disponibles, et sous quelle forme ?
- Quelle action de réparation et de test peut être faite ?

On entrevoit très vite que les performances de diagnostic sont directement liées à la qualité du modèle, c'est-à-dire à son aptitude à décrire le système à surveiller. Cette construction du modèle du système est la grosse difficulté du raisonnement à base de modèle. C'est cette phase qui va consommer du temps et des ressources dans le but d'obtenir un modèle robuste.

2.3.1.5 Modélisation

2.3.1.5.1 Importance de la modélisation

La réalisation du modèle est la clé de voûte du raisonnement à base de modèle. En effet la qualité du modèle et son degré de représentativité du système réel influencent directement les performances de son diagnostic, de son pronostic, de sa surveillance, de sa simulation, etc. Il est donc évident que cette phase ne doit pas être négligée, et c'est ce qui fait que l'application du raisonnement à base de modèle est coûteuse en terme de temps et de ressources.

Mais la modélisation permet la réutilisation. Cela consiste à récupérer le modèle d'un composant du système et à pouvoir le réutiliser à chaque fois que le composant intervient dans le système. En outre, il peut être réutilisé pour le modèle d'un système différent qui comporterait des composants communs aux deux systèmes. La réutilisation garantie une certaine robustesse et permet de diminuer le temps de création d'un modèle qui utilise en partie les mêmes composants qu'un autre modèle déjà construit.

2.3.1.5.2 Validation et Vérification formelle de modèle

Nous avons souligné à quel point le modèle était important dans ce raisonnement, et dans la qualité du diagnostic particulièrement. Il faut que le modèle soit représentatif du système, et il

serait dommageable de ne pas penser aux erreurs que nous pouvons faire lorsque nous le créons. Pour cela, deux méthodes sont employées : la validation et la vérification formelle de modèle.

La validation de modèle consiste à la création et l'application de tests par simulation sur un modèle pour détecter de potentielles erreurs et les corriger. Ce sont le plus souvent des tests représentatifs ou critiques qui sont exécutés, afin de voir comment le modèle se comporte. L'avantage principal de cette méthode est qu'elle est intuitive et facile à mettre en place. Cependant, son inconvénient principal est qu'elle ne peut raisonnablement pas être exhaustive.

La méthode de vérification formelle, bien plus puissante, consiste à vérifier des propriétés sur le modèle ou le système grâce au modèle. Typiquement, une propriété à vérifier peut être de savoir s'il est possible d'atteindre (propriété dite d'accessibilité) un état critique du système. Dans le cas où cela serait possible, un exemple d'exécution le montrant peut être exhibé. L'avantage principal de cette méthode est qu'elle va vérifier la propriété pour tous les cas, c'est-à-dire répondre clairement en disant "oui, c'est possible, et de cette façon" ou "non, ce n'est pas possible, en aucun cas", et ce, de manière automatique, laissant juste à l'utilisateur le soin d'exprimer la propriété. Son principal défaut est ses limites concernant son applicabilité à la vérification de systèmes particulièrement complexes.

2.3.1.5.3 Différents types de modélisation

Dans la littérature, on trouve énormément d'adjectifs différents pour caractériser un modèle. D'ailleurs, rien que de par le fait que le mot "modèle" indique tout ce qui peut représenter un système, on a déjà énormément de matière. Luca Console, professeur d'informatique à l'université de Turin, s'est amusé à représenter sous forme d'arbre quelques-uns uns (de manière non exhaustive, comme il aime le préciser) des adjectifs employés pour caractériser la modélisation dans le raisonnement à base de modèle.

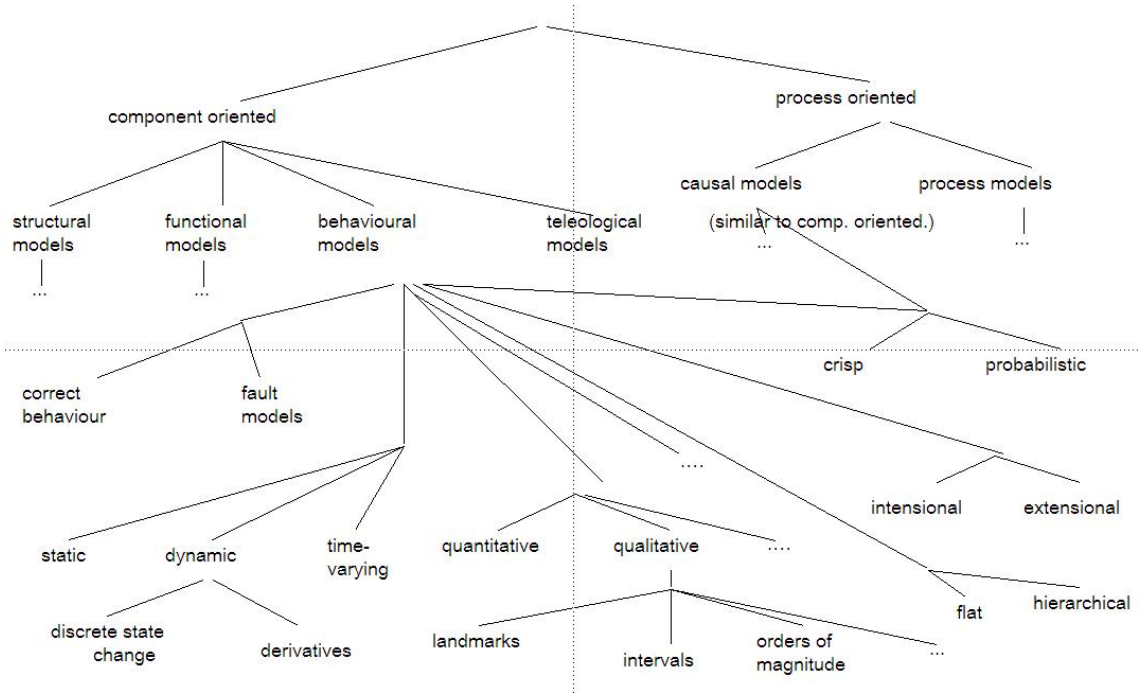


FIGURE 2.6 – Quelques adjectifs pouvant caractériser un modèle [C97CUT]

Il ressort des recherches que ces types de modèles, ces adjectifs, sont donc choisis suivant l'utilisation que l'on veut faire du modèle, selon l'optique dans laquelle on se place. L'arbre proposé par Luca Console a le mérite de nous exposer à quel point le problème est complexe. En effet, si les nœuds frères ont une cohérence entre eux et sont comparables, les nœuds cousins eux, ne portent pas

toujours sur deux caractéristiques comparables d'un système. Par exemple, si les adjectifs *"static"* et *"dynamic"* désignent l'aptitude d'un système à évoluer, l'adjectif *"qualitative"* d'un nœud cousin désigne l'aptitude à raisonner/déduire à partir d'information peu précises.

Puisque ces termes cousins désignent des caractéristiques différentes, ils peuvent être utilisés pour définir un même modèle. A cela s'ajoute le fait que certains termes frères dans l'arbre ne sont pas non plus exclusifs. C'est le cas de *"behavioural"* et *"functional"* par exemple qui peuvent s'appliquer à un même modèle. Un modèle peut donc être comportemental (*behavioural*), fonctionnel (*functional*), dynamique (*dynamic*) et basé sur le comportement correct (*correct behaviour*) par exemple. Mais parmi ce grand nombre de qualificatifs, on peut tout de même remarquer que certains termes reviennent souvent dans la littérature, comme *"structural"*, *"functional"*, *"causal"*, *"behavioural"* ou encore *"qualitative"*. Nous allons donc donner une brève définition de ces termes.

2.3.1.5.3.1 Modélisation structurelle

Ce type de modélisation traite de la structure physique du système. C'est en fait la modélisation la plus intuitive et bas niveau qui consiste à modéliser les composants et les liens qu'ils ont entre eux. Pour des détails, on pourra consulter les documents [BKJS06, BKLS07].

2.3.1.5.3.2 Modélisation fonctionnelle

Ce type de modélisation est utilisé pour modéliser un système par rapport au but que le concepteur du système lui a donné et qu'il doit réaliser, ainsi que les fonctions qu'il doit remplir. Une description satisfaisante de ce type doit répondre de manière explicite à ces questions :

- Pourquoi le système a-t-il été conçu au départ ?
- Qu'est censé faire le système pour atteindre son but ?
- Quelles sont les fonctions existantes et comment interagissent-elles ?
- Comment les fonctions sont allouées sur la structure physique du système ?

Le but de ce type de modélisation est de représenter le but global du système, les fonctions qui atteignent ce but et le comportement de la structure physique pour réaliser ces fonctions.

2.3.1.5.3.3 Modélisation causale

Ce type de modélisation a pour but de représenter le comportement d'un système par sa logique de cause à effet. La causalité formalise les relations de dépendances entre les composants. C'est un raisonnement intuitif lorsque l'on cherche la cause d'une défaillance sur un composant. Le modèle causal permet alors de remonter directement aux causes possibles de la faute.

2.3.1.5.3.4 Modélisation comportementale

Faire une modélisation comportementale d'un système, c'est créer un modèle qui reproduit le comportement du système analysé tel qu'il y a une correspondance directe (une à une) entre le comportement du système original et le comportement du système simulé. Le modèle peut dans certains cas simuler les différents comportements que peut avoir le système. Pour des détails, on pourra consulter [MS91].

2.3.1.5.3.5 Modélisation qualitative

Le raisonnement qualitatif a pour principe de raisonner à partir d'informations peu précises. Un modèle qualitatif est généralement utilisé pour décrire la structure physique d'un système, et pour, à l'aide de techniques de raisonnement appropriés, en déduire le comportement du système. Pour des détails, on pourra consulter [YL91, Str97, Sta00, Jos98, For96].

2.3.1.6 Méthode de diagnostic par résidus et hypothèses

Nous avons abordé le raisonnement à base de modèle d'un point de vue général, et nous avons vu qu'il y a différentes façons de faire ce type de diagnostic. Nous allons maintenant nous intéresser à un type particulier de diagnostic à base de modèle dans le but d'avoir une idée de comment cela peut fonctionner.

Nous allons nous pencher sur le diagnostic par résidus et hypothèses que l'on peut retrouver dans un article de Randall Davis et Walter Hamscher [DH88]. Dans cet article, ils proposent, une fois que l'on a détecté un comportement non attendu dans le système, de commencer à faire le diagnostic à partir des résidus (différences entre le comportement observé et celui attendu) et de le découper en trois phases : la phase de génération d'hypothèses, la phase de test d'hypothèses, et la phase de discrimination d'hypothèses.

On peut donner une idée de ce qu'est ce diagnostic à travers la figure 2.7.

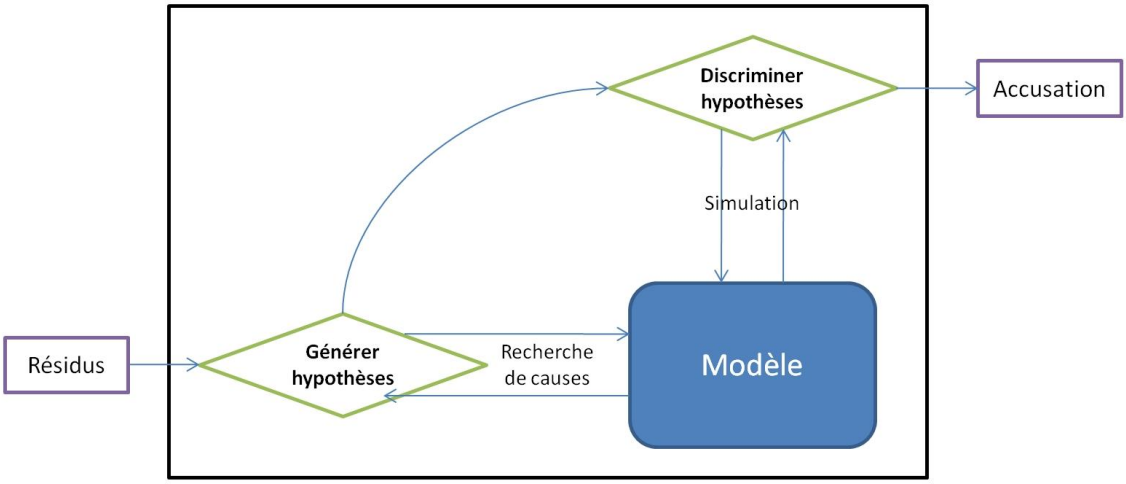


FIGURE 2.7 – Diagnostic par résidus et hypothèses

La phase de génération d'hypothèses démarre donc avec les résidus. Le but est de déterminer quels sont les composants qui ont pu engendrer ces résidus. Un bon générateur d'hypothèses se veut complet (c'est-à-dire capable de construire toutes les hypothèses plausibles), non redondant (c'est-à-dire capable de ne générer qu'une seule fois chaque hypothèse), et informé (c'est-à-dire capable de créer peu d'hypothèses qui se révélerons incorrectes par la suite).

Par exemple, le générateur le plus simple est celui qui considère tous les composants comme hypothèses. Il est forcément complet, non redondant, par contre, il n'est pas du tout informé. Il peut être ensuite amélioré grâce à plusieurs observations, comme par exemple le fait que pour être suspect, un composant doive être d'une manière ou d'une autre en lien avec les résidus...

La phase de test d'hypothèses a pour but de décider quels composants, parmi ceux qui ont été incriminés par la première phase, ont pu défaillir et engendrer toutes les observations. Cela se fait souvent par simulation (sur le modèle du système) de comportement défaillant des composants considérés.

Si le générateur d'hypothèses est complet, cela implique qu'après cette étape de test, il reste uniquement tous les composants ayant pu amener le système dans le comportement non attendu dans lequel il se trouve. Dans le cas où le modèle est représentatif du système, après cette phase, on se retrouve avec au minimum un suspect. De plus, s'il ne reste qu'un seul composant, c'est que ce dernier est la cause du comportement déviant du système. S'il reste plusieurs suspects, il va falloir déterminer lequel d'entre eux est le coupable. C'est ce qui est réalisé durant la troisième phase.

La phase de discrimination d'hypothèses a pour enjeu de réussir à voir comment les hypothèses découlant des deux premières phases peuvent se distinguer les unes des autres, et ainsi permet

d'éliminer une nouvelle fois celles qui ne sont pas la cause de la situation problématique observée sur le système.

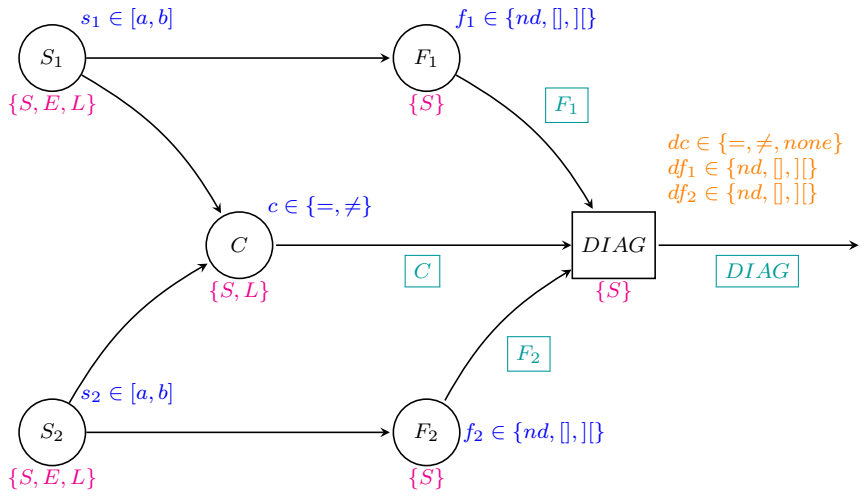
Typiquement, cette phase peut consister à exhiber les différences entre les différents suspects et à, par exemple, définir une procédure de tests à faire sur le système, ce qui va finalement permettre d'identifier la cause du problème.

On remarque que la deuxième étape du diagnostic proposé par Randall Davis et Walter Hamscher ne sert qu'à réduire le nombre de suspects construits durant la première phase. On peut se dire alors que la première phase peut se suffire à elle-même si elle est déjà précise au point que la simulation n'apporterait rien de plus, si le générateur est assez informé. Une idée est donc, en utilisant le modèle, de déduire à partir des résidus, et plus généralement, de l'observation du système, les problèmes qui auraient pu les engendrer. Dans ce sens, des techniques comme la génération de séquences ou l'utilisation du μ-calcul peuvent constituer des pistes.

2.3.2 Illustration du diagnostic à base de modèle

Dans cette partie nous allons montrer une utilisation du diagnostic à base de modèle sur l'Exemple (introduit dans la partie 2.1.2 et représenté en figure 2.1). Nous illustrerons ce diagnostic par le raisonnement à base de modèles particulier dont nous avons décrit le fonctionnement dans la partie précédente, et que nous avons appelé "diagnostic à base de modèles par résidus et hypothèses".

La première étape pour permettre d'effectuer le diagnostic à base de modèle d'un système, c'est d'avoir un modèle le représentant. Nous proposons un modèle (la façon de modéliser un système n'étant pas unique, il est tout à fait possible d'utiliser un autre modèle) en figure 2.8.



– Valeur transférée
– États
– Variable d'observation
– *Transition*

FIGURE 2.8 – Modèle du système

Nous retrouvons dans le modèle les états des différents composants ainsi que les valeurs transférées par les composants que nous avions déjà identifiés dans le système. A cela s'ajoutent les transitions qui définissent comment sont obtenues les valeurs à transférer et variables d'observation à partir des valeurs d'entrée :

- C : $\begin{cases} \text{si } (s_1 \neq s_2) & \text{alors } c = \prime \neq \prime \\ \text{si } (s_1 = s_2) & \text{alors } c = \prime = \prime \\ \text{si } (s_1 = \emptyset \text{ ou } s_2 = \emptyset) & \text{alors } c = \prime \neq \prime \end{cases}$
- F_1 : $\begin{cases} \text{si } (s_1 \in [x,y]) & \text{alors } f_1 = [] \\ \text{si } (s_1 \notin [x,y]) & \text{alors } f_1 =][\\ \text{si } (s_1 = \emptyset) & \text{alors } f_1 = nd \end{cases}$
- F_2 : $\begin{cases} \text{si } (s_2 \in [x,y]) & \text{alors } f_2 = [] \\ \text{si } (s_2 \notin [x,y]) & \text{alors } f_2 =][\\ \text{si } (s_2 = \emptyset) & \text{alors } f_2 = nd \end{cases}$
- $DIAG$: $\begin{cases} \text{si } (c = \emptyset) & \text{alors } diagc = none \\ \text{si } (c = \prime = \prime \text{ ou } c = \prime \neq \prime) & \text{alors } diagc = c \\ & diagf_1 = f_1 \\ & diagf_2 = f_2 \end{cases}$

On peut apporter quelques précisions sur la syntaxe et le vocabulaire utilisés :

- Les transitions sont présentées sous la formes d'expressions "si-alors". La partie "condition" considère les valeurs en entrée et calcule les valeurs que les composants vont transmettre.
- Lorsqu'une condition est du type "$\emptyset$", cela signifie que la valeur n'a pas été reçue. Par exemple, une des transitions sortante du composant $DIAG$ exprime le fait que si DIAG ne reçoit pas de valeur du composant C, alors il va mettre à jour la variable d'observation de façon à ce qu'elle représente qu'il n'a rien reçu pour C. C'est pour cela que la variable *diagc* est mise à *none*.
- Lorsqu'une condition est du type "appartient", cela exprime le fait que la valeur testée est ou non dans un intervalle particulier. Par exemple, une des transitions sortantes du composant F_2 teste si la valeur transmise par le composant S_2 est dans l'intervalle attendu par la fonction, l'intervalle [x,y].

Dans cette illustration, tout comme dans les illustrations précédentes, nous allons utiliser des triplets pour représenter les observations et des formules logiques pour exprimer les accusations.

Le modèle est dans l'état nominal qui correspond à l'observation $(=, [], [])$. Considérons une situation problématique dans la quelle on observe $(\neq, [],][)$. Littéralement, le système observe que les deux sondes n'envoient pas la même valeur (le comparateur C dit $\neq$), que la valeur envoyée par la sonde S_1 est dans l'intervalle attendu (la fonction F_1 dit $[]$) et que par contre la valeur envoyée par la sonde S_2 n'est pas dans l'intervalle attendu (la fonction F_2 dit $][$).

On relève donc des différences entre le comportement (nominal) du système attendu par le modèle, et le comportement observé. À partir de là, on obtient les résidus. Ici, C envoie $\neq$ au lieu de $=$, F_1 envoie bien $[]$, et F_2 envoie $][$ au lieu de $[]$: les résidus sont donc $c = \prime \neq \prime$ et $f_2 =][$.

Dans le diagnostic par résidus et hypothèses, et comme pour de nombreuses méthodes de diagnostic dont la phase amont de détection est basée sur les résidus, on démarre le diagnostic à partir de ces résidus. Dans notre cas, nous entrons dans la première phase du diagnostic qui est la phase de génération d'hypothèses.

Considérons une génération d'hypothèses simple et naïve possible :

- $c = \prime \neq \prime$ donc le générateur regarde quelles sont toutes les entrées possibles au composant C dans le modèle : les sondes S_1 et S_2. À partir de là, le générateur d'hypothèses construit une accusation : $S_1 \vee S_2$.
- Le générateur en fait de même pour le deuxième résidu $f_2 =][$ et génère l'accusation : S_2.
- On remarque que le générateur pourrait être plus intelligent par exemple en exploitant plus profondément le modèle et en ne s'arrêtant pas uniquement à son aspect physique [13].

13. la beauté intérieure du modèle -transitions, logique, etc.- n'est pas exploitée).

L'accusation finale de la première phase du diagnostic est obtenue par une conjonction des différentes accusations. Ici on obtient :

$$\begin{aligned} & [S_1 \vee S_2] \wedge [S_2] \\ = \ & (S_1 \wedge S_2) \vee S_2 \end{aligned}$$

Nous passons maintenant dans la deuxième phase de notre diagnostic : la phase de test d'hypothèses. Par simulation, en se restreignant aux suspects pointés du doigt par la première phase, des précisions vont être obtenues. Considérons nos résidus : $c = \ ' \neq'$ et $f_2 =][$. Des simulations vont être effectuées à partir des suspects S_1 et S_2 (typiquement en changeant l'état des composants suspects dans le modèle et en observant les effets sur les variables c et f_2). Avec la simulation, on remarque donc que nous obtienons à la fois $c = \ ' \neq'$ et $f_2 =][$ quand à la fois S_2 est erronée et S_1 est soit fonctionnelle, soit erronée, soit en panne (dans n'importe quel état en fait ; cela signifie qu'aucune information particulière n'est obtenue sur S_1). Du point de vue logique, l'accusation est : $S_1.SEL \wedge S_2.E$.

Nous entrons dans la troisième et dernière phase de notre diagnostic avec l'accusation $S_1.SEL \wedge S_2.E$. Cette phase, dite de discrimination d'hypothèses, consiste donc en la distinction des différentes hypothèses. Ici, nous en avons donc trois : $(S_1.S \wedge S_2.E)$ ou $(S_1.E \wedge S_2.E)$ ou $(S_1.L \wedge S_2.E)$. Dans toutes les hypothèses, on sait que S_2 est défaillant. La procédure de distinction peut donc consister à remplacer la sonde défaillante S_2 et à tester par rapport à cette dernière la sonde S_1. Dans notre exemple, considérons que notre sonde S_1 répond bien au test, on en déduit donc que l'accusation est la suivante : $S_1.S \wedge S_2.E$, c'est-à-dire $S_2.E$.

Nous venons de voir une utilisation d'un diagnostic à base de modèle. Les méthodes de génération, de test, et de discrimination d'hypothèses choisies pour l'illustration ne sont pas uniques, tout comme le modèle représentant le système ou encore la méthode de diagnostic à base de modèle. Néanmoins, cela permet de toucher du doigt la puissance, l'exhaustivité, l'objectivité d'un modèle et d'avoir une idée de l'utilisation qui peut en être faite.

2.3.3 Avantages et limites

Un des principaux points forts de l'approche à base de modèle pour le diagnostic est la précision qu'il permet d'obtenir pour le diagnostic de défaillances. Cette approche permet de donner toutes les causes pouvant expliquer une observation, et ce de manière exacte, sans cause ajoutée et sans cause oubliée. La gestion des pannes multiples est aussi un gros avantage, à partir du moment où le modèle est complet/représentatif.

À cela s'ajoute le fait que le raisonnement à base de modèle a une grosse capacité de réutilisation. En effet le système est décrit de manière modulaire, ce qui permet par exemple dans le cas où tous les composants sont modélisés indépendamment, de les réutiliser à chaque fois qu'ils apparaissent dans le système, ou même encore lorsqu'ils interviennent dans d'autres systèmes.

On peut aussi relever que les modèles sont évolutifs et donc permettent de traiter un côté dynamique à travers leur simulation.

En outre, l'utilité de la modélisation ne se restreint pas au diagnostic, mais s'applique aussi dans d'autres domaines du cycle de vie d'un système.

Le principal point négatif de l'approche à base de modèle est que sa phase de modélisation prend du temps et des ressources. En effet, pour avoir un bon modèle, il faut s'appliquer à bien le créer, ce qui prend du temps. Mais on peut voir qu'au fur et à mesure des modélisations, la bibliothèque de composants et de modèles s'enrichira et permettra de réaliser plus facilement les modèles futurs. De plus, l'effort de formalisation du modèle permet de lever de nombreuses ambiguïtés dans le processus de développement.

Et malheureusement, avoir un diagnostic très précis implique ici d'avoir un diagnostic un peu plus lent.

2.3.4 Formalismes / Outils

2.3.4.1 Introduction

Dans cette partie nous référençons et commentons les formalismes et les outils les plus utilisés pour faire du diagnostic à base de modèle.

Dans un premier temps nous verrons donc les formalismes sous forme d'un tableau de synthèse dans lequel nous commenterons leurs différentes caractéristiques ; des références seront aussi proposés pour les différents formalismes. Enfin, dans un second temps, nous nous intéresserons aux outils existants utilisés pour faire de la modélisation de systèmes et du diagnostic.

2.3.4.2 Formalismes

Nous avons décidé de présenter les différents formalismes rencontrés sous forme de tableau en figure 2.10. Dans ce tableau, nous donnons quelques attributs des formalismes selon leur utilisation, selon leur domaine, selon leurs caractéristiques et finalement selon leurs limites. Une dernière colonne est présente, pour plus d'informations, donnant des références vers des documents qui utilisent ces formalismes.

Dans le cas de la représentation de systèmes complexes, la taille des modèles qui les représentent devient rapidement très importante. Par conséquent, le modèle est difficilement exploitable de manière efficace sans outil. Il est donc très important de disposer d'outils d'analyse de modèles possédant des algorithmes optimisés.

2.3.4.3 Outils

Nous allons donc vous présenter différents outils qui sont utilisés pour modéliser des systèmes dans l'optique de faire du diagnostic.

2.3.4.3.1 RODON

Produit par Sörman Information & Media AB, RODON est un outil utilisant les idées du diagnostic à base de cohérence, venues avec l'apparition de GDE [dKW87]. C'est l'une des seules implémentations commerciales de ce type d'outil de diagnostic.

RODON est basé sur un modèle orienté composant virtuel d'un système qui est appelé "Virtual Product Model" (VPM) et couvre différentes tâches d'analyse intervenant dans le cycle de vie d'un produit technique. Cela va du design au diagnostic, en passant par de l'analyse de risques à travers les analyses FMEA (Failure Mode and Effects Analysis) et FTA (Fault Tree Analysis).

La modélisation dans RODON est graphique.

Pour plus d'informations, on peut se référer à de nombreux documents : [AB09, Lun00, BIFM09, AB01, Sä07, Isa09, LW07, AS07, Mat05]

2.3.4.3.2 eXpress

Produit par DSI International, le logiciel eXpress est un outil qui utilise une modélisation graphique fonctionnelle des dépendances pour représenter un système. Il regroupe plusieurs domaines associés au diagnostic à base de modèle, comme la conception, la testabilité ou encore le diagnostic et le pronostic de systèmes.

Pour plus d'informations, on peut se référer à `http://www.dsiintl.com`.

2.3.4.3.3 SIMFIA

Produit par EADS APSYS, SIMFIA est un logiciel de modélisation qui permet d'analyser et de simuler le fonctionnement global d'un équipement, système ou processus et d'automatiser les études de sûreté de fonctionnement et de Soutien Logistique Intégré (SLI). Il génère des FMEA (Failure Mode and Effects Analysis), des diagrammes de fiabilités, des arbres de défaillances...

Formalismes	Utilisation	Domaine	Caractéristiques	Limites	Références
Réseaux Neuronaux	Classification dans des systèmes complexes typiques non déterministes	• Observation de caractéristiques paramétriques • Peut être utile en cas de bruit ou de coupures dans l'enregistrement des données	• Structure créée par l'intermédiaire d'un apprentissage dirigé • Peut expliquer des aspects mal compris des systèmes • Peur s'adapter pour expliquer de nouvelles connaissances	• Difficultés à mettre au point et à valider • Exige un apprentissage considérable	[MNS+94]
Logique Floue	Système de contrôle et système de classification où les entrées et sorties peuvent être et devraient être décrit en langage plain	Toutes données pouvant être organisées en catégorie	• Permet d'obtenir une estimation sans une connaissance détaillée • Facilité de développement et de compréhension (Langage) • Utilise le concept de vérité partielle	La sortie peut être absurde	[Moh97]
Dempster-Shafer	• Les systèmes à logique floue utilisent cette méthode • Combinaison de réseaux d'opinion	Sortie des algorithmes de détection et de classification	Généralisation du bayésien qui permet l'incertitude (l'information de probabilité que nous n'avons pas). Intègre des notions d'"opinion" et de "vraisemblance"	La sortie peut être absurde	[Sme94]
Réseaux de Petri	• Utilisation pour le diagnostic de systèmes asynchrones et distribués • Utilisation pour la surveillance	• Fréquemment utilisés pour les systèmes à événements discrets • Description d'état de système	• Utilisation du dépliage (*unfolding*) de réseaux de Petri • Permet d'éviter le problème d'explosion d'états	• La diminution du nombre d'états provoque l'augmentation du temps de calcul • Ne correspondent pas à un modèle structurel	• [VK94] • [BFHJ02]
Réseaux Bayésiens	Raisonneur de haut niveau	Permet un diagnostic robuste à partir d'une connaissance ou de possibilités de modélisation incomplètes	• Description des entités • Description des interrelations (Processus, Physique, Proximal) • Encapsule une connaissance a priori • Intègre les informations sur les probabilités de défaillances	• Difficultés d'application aux larges systèmes (complexité) • Probabilité moins facile à aborder que la causalité	• [Agr96] • [Jen96] • [SK05] • [VER07] • [MMZ08] • [Dar09]
Modèles de Markov	Modélisation de la fiabilité de composantes système (ex : équipements électroniques)	• Description d'état du système • Peut générer MTTF et MTBF	L'évolution de système est indépendant de l'état actuel	• Un processus interactif peut être consommateur de ressources • Difficultés d'application aux larges systèmes (complexité) • Probabilité moins facile à aborder que la causalité	• [RL86] • [KPPH00]
AltaRica	Modélisation du comportement d'un système	• Description du (dys)fonctionnement des composants du système • Description des liens et échanges entre les composants	• Possibilité de validation et de vérification formelle des systèmes modélisés • Génération d'arbres de défaillances, de formules booléennes, de réseaux de Petri, de systèmes de transitions...	Pas d'interface de modélisation libre	• [GR02] • [Vin03b] • [GPV07] • [BCD+07] • [GPKV11]

TABLE 2.10 – Caractéristiques de formalismes pour le MBR

En outre, SIMFIA dans sa suite logiciel comporte un générateur de séquences qui peut donc être utilisé pour faire de la simulation et du diagnostic.

La modélisation (graphique ou avec du code) permet d'utiliser entre autres un dérivé du langage AltaRica(langage cité dans liste des formalismes en figure 2.10), et comporte donc plusieurs avantages et possibilités de ce langage.

Pour plus d'informations, on peut se référer à plusieurs adresses :

- `http://www.apsys.eads.net/fr/17/Nos-Progiciels`
- `http://www.apsys-software.com/simfia.xhtml`

2.3.4.3.4 OCAS

Produit par Dassault, l'outil OCAS (ou CECILIA OCAS) répond au même principe que l'outil SIMFIA d'EADS APSYS. En effet, OCAS est lui aussi basé sur le même dérivé du langage AltaRica(appelé AltaRicaDataflow). Il permet de modéliser le comportement des systèmes de manière graphique ou directement avec du code AltaRica. À partir de cela, on peut obtenir par exemple des arbres de défaillance ou des modèles stochastiques.

Pour plus d'informations, on peut se référer aux documents [BCK+02, Sch02, DP02] présentés lors des Journées AltaRica(voir `http://altarica.labri.u-bordeaux.fr/wiki/community:workshops`)...

2.3.4.3.5 AltaRica-Studio

Développé au LaBRI, AltaRica-Studio est une suite d'outils qui permet l'exploitation du langage AltaRica(cf. table 2.10). La suite intègre notamment ARC, un outil d'exploitation de modèle AltaRica qui est utilisé pour manipuler l'automate associé au modèle.

Nous décrivons plus en détails ces outils dans la section 6.1.

2.3.4.3.6 Livingstone

Livingstone est un système de diagnostic développé par la NASA. Il constitue une partie d'un système complet de maintenance utilisé à l'origine pour les véhicules spatiaux. Le but de ce système est d'aider le véhicule à savoir réagir aux problèmes qui subviennent, et ce, sans intervention humaine. Livingstone modélise le système à diagnostiquer en en représentant les composants et leur comportement.

Pour plus d'informations, on peut se référer aux documents [SSB02, Mat05] ou aux sites `http://ti.arc.nasa.gov/tech/rse/vandv/livingstone/` et `http://www.nasa.gov/centers/ames/research/technology-onepagers/livingstone2-modelbased.html`.

2.4 Synthèse et choix

2.4.1 Synthèse des différentes approches

Nous avons abordé dans ce chapitre trois grands types de raisonnement qui peuvent être utilisés pour faire du diagnostic de systèmes : l'approche à base de règles, l'approche à base de cas, et l'approche à base de modèle. Ces approches, qui sont les plus répandues dans les mondes académique et industriel, ont été décrites dans ce début de document.

Il est intéressant après cela d'essayer de donner une vue d'ensemble des trois approches que nous avons décrites afin de décider quelle approche est la plus appropriée dans le cadre dans lequel s'inscrivent les travaux de thèse.

Nous proposons donc en guise de synthèse la table 2.11.

Critères \ Approches	RBR	CBR	MBR
Précision du diagnostic	+	•	++
Vitesse du diagnostic	++	+	•
Difficulté de construction des tables/bases/modèles	•	+	−
Efficacité au lancement du produit	+	•	+
Efficacité sur les problèmes complexes (multi-pannes...)	−	+	++
Exhaustivité des résultats	−	−	++
Capacité d'apprentissage	−	++	−
Objectivité des connaissances retranscrites	•	++	+
Réutilisabilité (utilisation pour différents systèmes)	•	−	++
Validation des tables/bases/modèles	+	•	++
Vérification formelle des tables/bases/modèles	−	−	++
Polyvalence (utilisation pour plusieurs domaines)	−	•	++

++ : très bon ; + : bon ; • : neutre ; − : mauvais/inapproprié

TABLE 2.11 – Évaluation des différentes approches

Il ressort clairement qu'aucune des trois méthodes n'est la méthode parfaite de tout point de vue. Cela indique, étant donné que les approches ont chacune leurs points forts sur des critères différents, qu'il est important de réfléchir à ce que l'on veut réellement pour le diagnostic de notre système.

Nous pouvons tirer différentes conclusions de l'analyse des raisonnements que nous avons faite, résumée à travers ce tableau :

- L'approche à base de règles est rapide, simple et assez efficace car il utilise uniquement des actions d'inférence et d'évaluation. En revanche, les tables de règles sont potentiellement incomplètes et incohérentes puisque souvent écrites "à la main" par des experts, et elles sont pas adaptées à la prise en compte de problèmes complexes.
- L'approche à base de cas est objective puisque la base est renseignée uniquement avec les cas qui arrivent. Elle est aussi la seule des approches à avoir une capacité d'apprentissage. D'un autre côté, la base n'est pas exploitable dès le départ, et il est difficile de résoudre un cas qui n'a pas déjà été rencontré. En outre, les résultats de diagnostic extraits de la base ne sont pas tout le temps précis et son utilisation est restreinte au diagnostic du système considéré (et ni à un autre système, ni à un autre domaine que le diagnostic).
- Finalement, l'approche à base de modèle est une approche robuste, précise et efficace. Le modèle permet une accusation exhaustive et gère les multi-pannes grâce à la description formelle du comportement du système. Il est en outre parfaitement adapté pour la réutilisation à la fois pour d'autres systèmes (capitalisation) et pour d'autres domaines comme la sûreté de fonctionnement et le *design* de systèmes. En contrepartie, la création du modèle est une étape longue (tout de même appuyée par la validation et la vérification formelle), aucune capacité d'apprentissage automatique n'est présente et le diagnostic n'est pas le plus rapide.

L'évaluation et la comparaison des différents raisonnement présentés que nous venons de réaliser nous permet de faire des choix quant au raisonnement que nous allons considérer pour faire du diagnostic dans le cadre des travaux de la thèse.

2.4.2 Choix et orientation des travaux

Suite à l'évaluation des différentes approches pour le diagnostic, et en regard du contexte maintenance dans lequel s'inscrivent les travaux de thèse, il apparaît que certains critères d'évaluation sont plus importants que les autres. Ces critères sont les suivants (ils ne sont pas classés par ordre d'importance) :

- Ne pas oublier des causes possibles à une défaillance pour toujours trouver la solution lors de pannes $\Longrightarrow$ Critère : Exhaustivité.
- Obtenir des résultats de diagnostic précis pour gagner en efficacité lors des interventions de maintenance $\Longrightarrow$ Critère : Précision du diagnostic.
- Être efficace sur les problèmes multi-pannes qui sont représentatif de ce qui arrive dans le domaine de l'aéronautique $\Longrightarrow$ Critère : Efficacité sur les problèmes complexes.
- Faciliter la construction des tables/bases/modèles pour que l'effort de développement de la solution soit modéré $\Longrightarrow$ Critère : Difficulté de construction.
- Avoir une méthode générique/réutilisable pour que la solution choisie soit viable d'un point de vue industriel $\Longrightarrow$ Critères : Réutilisabilité/Polyvalence.
- Pouvoir faire des analyses sur les tables/bases/modèles pour vérifier leur bon comportement, leur exhaustivité et vérifier que les contraintes industrielles sont respectées $\Longrightarrow$ Critères : Validation/Vérification formelle.

L'approche qui répond le plus favorablement à ces critères importants est l'approche à base de modèle. En effet, elle nous apparaît comme incontournable. Sa mise en place apporterait des améliorations dans la chaîne de vie d'un produit : au niveau de la conception du produit, de sa sûreté de fonctionnement, de la simulation du fonctionnement des systèmes et de leur vérification, ainsi que bien entendu au niveau de la maintenance des systèmes et de leur diagnostic.

Créer un modèle serait aussi un processus plus naturel pour un humain. En effet, aujourd'hui, c'est un expert qui fait l'effort de créer la base de règles de diagnostic. Il sait comment fonctionne le système puisqu'il a une représentation de son comportement en tête. Il va essayer de voir les situations qui peuvent arriver, trouver leur(s) cause(s) et les transcrire en règles. Finalement, l'opération qu'il fait revient à parcourir le modèle du système qu'il a dans la tête. L'esprit humain n'est pas fait pour ce type d'opérations, encore moins quand les systèmes sont complexes, contrairement à l'informatique. L'idée est donc de faire transcrire à l'expert le modèle qu'il a dans la tête et de l'exploiter ensuite. Un autre avantage de cette façon de faire est que lors de la création d'un modèle, il n'est pas question de penser directement au modèle dans son ensemble mais de manière plus locale, en modélisant d'abord les composants, puis les relations entre eux, etc. ce qui est beaucoup plus facile pour l'expert et laisse beaucoup moins de place aux oublis de règles.

Nous pouvons aussi imaginer exploiter plusieurs de ces approches afin d'utiliser leurs points positifs et d'éviter leurs points négatifs.

Il serait par exemple très intéressant de combiner d'une part le raisonnement à base de règles, rapide et peu gourmand en stockage mais dont les règles sont très difficiles à écrire et souvent incomplètes, et d'autre part le raisonnement à base de modèle qui lui est complet et nécessite seulement d'écrire le modèle, mais est gourmand en stockage et peu rapide. Pour ce faire, l'idée est de générer l'ensemble des règles de diagnostic à partir du modèle, le diagnostic serait ainsi complet, moins gourmand en stockage, rapide et automatique. Même si cette méthode est moins efficace que l'approche à base de modèle pure, elle pourrait aussi constituer une technologie intermédiaire, à la fois plus abordable que la technologie tout modèle et plus proche des processus actuels.

En outre, en cas d'ambiguïté sur un diagnostic effectué à partir du modèle ou des règles, on peut imaginer une phase de post-traitement exploitant une base de cas (retours d'expérience) afin de raffiner les résultats et d'éliminer les causes les moins probables. Ce type de méthode peut donc clairement être mise en place pour la phase d'aide à la décision (cf. section 1.3)

C'est, pour ces différentes raisons, cette orientation que nous avons choisie pour les travaux dans le groupe d'études à Thales Avionics ainsi que pour cette thèse.

En ce qui concerne les différents formalismes que nous avons vus pour les approches à base de modèles, il apparaît que le langage AltaRica correspond tout à fait à nos attentes concernant la modélisation des systèmes qui nous concernent. En outre, la capacité de vérification formelle de modèles dont dispose le langage permet de vérifier l'exactitude et la robustesse des modèles. Le langage AltaRica est de plus compatible avec différents outils (libres, et du commerce). C'est ce langage que nous choisissons pour la modélisation dans le contexte de nos travaux.

Deuxième partie

Monitoring et performances de diagnostic

Étant donné le problème de l'impact du monitoring sur les performances de diagnostic, cette partie traite des méthodes pour améliorer la stratégie de monitoring tout en ayant de bonnes performances de diagnostic.

Deux phases sont à considérer dans ce problème. Tout d'abord une première phase qui consiste à définir ce que sont les performances de diagnostic et comment il est possible de les mesurer. Ensuite, une deuxième phase s'intéresse aux méthodes que l'on peut mettre en place afin d'optimiser la stratégie de monitoring du système sous diagnostic relativement à ces mesures.

Dans cette partie, un premier chapitre définit le type de système que nous manipulons, puis un chapitre est consacré à chacune des deux phases décrites dans le paragraphe précédent.

Chapitre 3

Formalisation du problème sur une approche à base de modèle

Le chapitre précédent a présenté différentes approches pour faire du diagnostic, et parmi elles, a mis en avant l'approche à base de modèle. Dans notre travail, une première partie consiste à évaluer la capacité du système à fournir de l'information utile pour le diagnostic. Mais avant de regarder cela, il nous faut décrire l'objet modèle que nous utiliserons par la suite pour représenter le système ainsi que les notions qui lui sont associées.

Ce chapitre se concentre sur ces aspects et décrit donc le type de systèmes que nous considérons dans cette thèse, à savoir les systèmes à événements discrets, ainsi que dans quelle mesure ces systèmes sont des systèmes sous diagnostic. Ensuite, dans ce chapitre, nous mettons en avant quelques propriétés des systèmes sous diagnostic, puis nous nous intéressons aux résultats de diagnostic qui peuvent être calculés à partir de ce type de système. Finalement, nous introduisons un exemple d'illustration qui sera utilisé tout au long des chapitres de la partie II.

3.1 Système à événements discrets

Le type de systèmes auquel nous nous intéressons dans cette thèse est le type *systèmes à événements discrets*. Ces événements discrets permettent notamment de représenter/modéliser des événements modifiant l'état du système que l'on veut diagnostiquer. Pour la suite de ce document, nous choisissons de modéliser le système par un *système de transitions étiquetées*, tel que défini ci-après.

Définition 1 (Système de transitions étiquetées) *Un* système de transitions étiquetées *(ou LTS) est un n-uplet* $S = \langle Q, Q_i, E, \rightarrow \rangle$ *où :*

- Q *est un ensemble de configurations.*
- $Q_i \subseteq Q$ *est l'ensemble des configurations initiales.*
- E *est un ensemble d'événements.*
- $\rightarrow \subseteq (Q \times E \times Q)$ *est une relation de transition.*

Pour $(q, q') \in Q^2$ et $e \in E$, $(q, e, q') \in \rightarrow$ est usuellement noté $q \xrightarrow{e} q'$.

Associées à un LTS $S = \langle Q, Q_i, E, \rightarrow \rangle$, nous retrouvons les fonctions habituelles qui donnent les successeurs ainsi que les configurations accessibles d'un ensemble de configurations $Q' \subseteq Q$ pour un ensemble d'événements $E' \subseteq E$. Nous avons donc tout d'abord la fonction qui donne les successeurs immédiats :

$$succ(Q', E') = \{q' \in Q \mid \exists (q, e) \in Q' \times E', q \xrightarrow{e} q'\}$$

À partir des successeurs immédiats, nous pouvons obtenir les successeurs de rang n (successeurs accessibles en n transitions) à travers la fonction $succ^n$ suivante :

$$succ^n(Q', E') = \underbrace{succ(\dots(succ}_{n \text{ fois}}(Q', E')\dots), E')$$

$$\text{où } succ^0(Q', E') = Q'$$

Par la suite, nous écrirons $succ(c, e)$ pour $succ(\{c\}, \{e\})$ avec $q \in Q$ et $e \in E$.

Nous pouvons aussi introduire la fonction $reach^n(Q', E')$ définissant l'ensemble des configurations accessibles depuis un ensemble de configurations Q' en utilisant au plus n transitions étiquetées par des événements de E' :

$$reach^n(Q', E') = \bigcup_{i=0\dots n} succ^i(Q', E')$$

Finalement, on peut exprimer l'ensemble de toutes les configurations accessibles depuis Q' pour E' à travers la fonction $reach$:

$$reach(Q', E') = \bigcup_{i \geq 0} succ^i(Q', E')$$

Lorsque l'ensemble des configurations Q est fini, $reach = reach^{|Q|}$.

En théorie du contrôle [RW87] on partitionne généralement l'ensemble des événements en deux catégories : les événements dits contrôlables, représentés par l'ensemble E_c, et les événements incontrôlables, représentés par $E_{nc} = \overline{E_c}$.

Les événements contrôlables représentent les actions que le système prend pour faire en sorte qu'il continue de fonctionner le plus correctement possible. Ce sont des actions internes au système. Les événements représentant les actions de reconfigurations, les actions de propagation des effets ou encore les actions de transmission des informations de monitoring font partie des événements contrôlables.

En revanche, les événements incontrôlables représentent les événements sur lesquels le système n'intervient pas. Ce sont des actions principalement externes au système. C'est typiquement le cas des défaillances, des actions de réparation, ou encore des réactions du système à l'environnement.

La notion de configurations stables et de configurations instables découle de l'aspect contrôlable des événements. Cette considération n'a un réel sens que lorsque les systèmes considérés sont dits *réactifs*. Dans un système de ce type, le système réagit pour contrôler/corriger/propager les effets des événements incontrôlables, et ce en provoquant des événements contrôlables.

Dans ce contexte, les configurations stables représentent les configurations dans lesquelles les réactions sont terminées. Les configurations stables Q_s sont donc les configurations à partir desquelles aucun événement contrôlable n'arrive et sont naturellement définies par le fait qu'elles sont sources *uniquement* d'événements non contrôlables.

$$Q_s = \{q \in Q \mid \forall (e, q') \in E \times Q,\ q \xrightarrow{e} q' \implies e \in E_{nc}\}$$

Les configurations dites instables sont des configurations de transition, temporaires, entre l'occurrence d'une défaillance et la fin de la réaction du système. Les configurations instables Q_{ns} sont donc les configurations dans lesquelles le système réagit. Elles sont les seules sources d'événements contrôlables. À noter qu'ici il est possible qu'un événement incontrôlable arrive pendant la phase de réaction du système, lorsque le système est dans une configuration instable.

Les notions de successeurs et accessibles peuvent être définis plus particulièrement pour les configurations stables. Nous obtenons donc l'ensemble des configurations stables successeurs immédiats de $Q' \subseteq Q$ pour $E' \subseteq E$ comme le plus petit point fixe de la fonction $succ_s$:

$$succ_s(Q', E') = (succ(Q', E') \cap Q_s) \cup succ_s(succ(Q', E') \cap Q_{ns}, E)$$

Nous obtenons les successeurs stables de rang n de la même manière que pour la définition générale, à travers la fonction $succ_s^n$:

$$succ_s^n(Q', E') = \underbrace{succ_s(\dots(succ_s}_{n \text{ fois}}(Q', E')\dots), E')$$

$$\text{où } succ_s^0(Q', E') = Q' \cap Q_s$$

L'ensemble des configurations stables accessibles depuis Q' pour E' en utilisant au plus n transitions est défini de la même façon que son équivalent sans la notion de stable :

$$reach_s^n(Q', E') = \bigcup_{i=0\dots n} succ_s^i(Q', E')$$

L'ensemble de toutes les configurations stables accessibles depuis Q' pour E' correspond à l'ensemble de toutes les configurations accessibles depuis Q' pour E' réduit aux configurations stables. Il peut aussi être défini d'une façon similaire à $reach$:

$$\begin{aligned} reach_s(Q', E') &= reach(Q', E') \cap Q_s \\ &= \bigcup_{i \geq 0} succ_s^i(Q', E') \end{aligned}$$

Avec la fonction $reach_s^n$, nous pouvons nous intéresser à la définition de groupes de configurations par leur profondeur en partant de l'ensemble des configurations initiales. Plus exactement, l'ensemble de configurations défini par la formule $reach_s^n(Q_i, E)$ contient les configurations stables atteignables depuis l'ensemble des configurations initiales en traversant au plus n configurations stables.

Le principe de cette formule se rapproche d'une notion que l'on retrouve fréquemment lors de l'utilisation de systèmes à événements discrets pour le diagnostic : la possiblité de borner les systèmes par un nombre maximum d'occurrences d'événements d'un certain type. Par exemple, nous sommes dans un cas de ce type lorsque nous souhaitons ne pas considérer plus que des occurrences triples d'événements pour un système, car nous estimons que l'occurrence de quatre événements est assez improbable pour être négligée.

La formule $reach_s^n(Q_i, E)$ ne décrit pas cela. Cette dernière se base sur le nombre de configurations stables traversées plutôt que sur le nombre d'événements d'un certain type déclenchés. Même si la caractérisation d'une configuration comme stable est fondée sur la non-contrôlabilité des événements, cela n'équivaut pas à compter le nombre d'événements non contrôlables déclenchés car rien n'interdit en théorie de rencontrer plusieurs événements non contrôlables entre configurations stables (même si en pratique, dans les systèmes réactifs, cette situation n'arrive pas). De plus, avec la fonction $reach_s^n$, nous considérons tous les événements de type non contrôlable alors que nous aimerions pouvoir n'en considérer qu'une partie. En effet, il est intéressant par exemple de pouvoir réduire le système en profondeur selon le nombre de défaillances sans y intégrer les événements de réparation qui sont pourtant aussi non contrôlables.

Nous voulons maintenant caractériser l'ensemble des configurations accessibles depuis les configurations initiales en déclenchant au maximum n événements d'un certain type, que nous noterons Q^n. Pour cela, nous introduisons $E_b \subseteq E$, l'ensemble des événements sur lesquels nous voulons borner la profondeur du système. L'idée de la méthode pour calculer Q^n est la suivante :

1. Considérer l'ensemble des configurations initiales.
2. Calculer l'ensemble des configurations qui sont accessibles depuis les précédentes sans utiliser d'événement bornant.
3. Calculer l'ensemble des successeurs immédiats des configurations de l'étape précédente qui sont atteints en utilisant exactement un événement bornant.
4. Réitérer l'étape 2 avec les dernières configurations calculées puis l'étape 3, et cela encore $n-1$ fois (pour s'assurer de déclencher n événements bornants).

5. Q^n est alors composé de l'ensemble des configurations parcourues lors des différentes étapes.

Pour définir Q^n, nous commençons par introduire la fonction $succ_{E_b}$ dont le plus petit point fixe donne l'ensemble des configurations accessibles depuis $Q' \subseteq Q$ en n'autorisant l'utilisation d'une transition étiquetée par un événement bornant que pour atteindre le premier successeur (on autorise en réalité tout événement de E' comme étiquette de première transition). Cela couvre les étapes 2 et 3.

$$succ_{E_b}(Q', E') = succ(Q', E') \cup succ_{E_b}(succ(Q', E'), \overline{E_b})$$

Notons que la conidition nécessaire pour que la fonction précédente corresponde à l'étape 2 dès la première itération, il est nécessaire de partir d'un ensemble Q' qui contient toutes les configurations qui sont accessibles depuis les configurations initiales sans utiliser d'événement bornant. Ainsi, avec $succ_{E_b}$, nous obtenons toutes les configurations atteignables en déclenchant exactement un événement bornant de E'.

Nous pouvons maintenant obtenir les configurations qui sont accessibles depuis Q' en utilisant soit n transitions étiquetées par un événement bornant (si le premier événement de E' déclenché est un événement bornant), soit $n-1$ (si ce n'est pas le cas). Nous introduisons pour cela la fonction $succ^n_{E_b}$ qui couvre l'étape 4 de la méthode.

$$succ^n_{E_b}(Q', E') = \underbrace{succ_{E_b}(\dots(succ}_{n \text{ fois}}(Q', E')\dots), E')$$

$$\text{où } succ^0_{E_b}(Q', E') = Q'$$

À nouveau, si dans le paramètre Q' (issu d'une étape précédente), nous avons atteint toutes les configurations accessibles en se limitant aux événements non bornants, alors $succ^n_{E_b}$ donne les configurations accessibles en utilisant exactement n événements bornants.

Finalement, nous pouvons définir Q^n.

$$Q^n = \bigcup_{i=0\dots n} succ^i_{E_b}(reach(Q_i, \overline{E_b}), E_b)$$

Ici il faut noter que le premier paramètre de la fonction $succ^i_{E_b}$ appelée correspond à l'ensemble des configurations accessibles depuis les initiales sans utiliser d'événement bornants. Cela était la condition nécessaire pour que les fonctions définies couvrent exactement les étapes 1 à 4 décrites précédemment.

Remarquons que $Q^n \subseteq Q$ et donc ne se réduit pas forcément à des configurations stables.

Introduisons finalement quelques notions et notations afin de pouvoir ensuite définir ce qu'est un système sous diagnostic.

Considérons l'ensemble V des variables du modèle. Une configuration de Q correspond à une valuation de variables de V. Par exemple, si on suppose que les variables de V prennent leur valeurs dans un unique domaine $\mathcal{D}$, alors une *valuation* de variables de V est une application de $\mathcal{F}(V, \mathcal{D})$, où $\mathcal{F}(V, \mathcal{D})$ représente l'ensemble des fonctions de V dans $\mathcal{D}$. En outre, si $V' \subseteq V$ et $Q' \subseteq Q$ alors on note $Q'[V'] \subseteq \mathcal{F}(V', \mathcal{D})$ l'ensemble des valuations de Q' restreintes au variables de V'. D'une manière similaire, si $V' \subseteq V$ et $\sigma \in \mathcal{F}(V, \mathcal{D})$ alors on note $\sigma[V'] \in \mathcal{F}(V', \mathcal{D})$ la restriction de la valuation σ aux variables de V'. L'ensemble des valuations des variables de V est noté $\mathcal{V}(V)$. Ainsi nous pouvons noter $Q \subseteq \mathcal{V}(V)$.

Par la suite, $BF(V_1)$ décrira l'ensemble des formules booléennes construites sur un ensemble de variables V_1. Si $f \in BF(V_1)$, alors $[\![f]\!] \subseteq \mathcal{V}(V_1)$ représente l'ensemble des valuations qui satisfont la formule f. Pour $F \subseteq BF(V_1)$, $[\![F]\!] \subseteq \mathcal{V}(V_1)$ représente l'ensemble des valuations qui satisfont toutes les formules de F.

3.2 Système sous diagnostic (SUD)

Dans la section précédente, nous avons défini le type de système que nous souhaitons analyser d'un point de vue diagnostic. Pour pouvoir effectuer du diagnostic sur un système, il est nécessaire de mettre en place la surveillance de celui-ci. Pour ce faire, la première étape est de définir un *observateur*.

Définition 2 (Observateur) *Un observateur O consiste en un ensemble de variables d'observation V_O.*

La seconde étape qui va finir de mettre en place la surveillance du système est de décrire la façon dont l'observateur va mettre à jour ses variables en fonction de l'état du système. Nous associons donc à l'observateur une *relation d'observation*, qui permet de lier l'observateur au système, et obtenons alors un système sous diagnostic.

Définition 3 (Système sous diagnostic) *Un système sous diagnostic (SUD) A correspond à un système S auquel sont associés un observateur O et une relation d'observation $\mathcal{R}_O \subseteq BF(V \uplus V_O)$, où $\uplus$ désigne l'opération d'union disjointe. On peut le noter $A = \langle S, O, \mathcal{R}_O \rangle$.*

Les variables de V_O sont distinctes des variables de V et n'interviennent pas dans la sémantique du modèle A. Elles représentent les résultats des informations de monitoring du système. Ces variables d'observations sont le résultat des *messages de monitoring*. La relation $\mathcal{R}_O$ décrit un ensemble de contraintes sur les variables du modèle. Elle a pour rôle de lier les variables au système et de définir les valeurs que ces dernières prennent en fonction de celles des variables du système.

Pour mieux appréhender les notions d'observateur et de relation d'observation, nous pouvons voir ce que cela représente dans une suite avionique. Dans la suite avionique, qui correspond au système sous diagnostic, nous pouvons identifier l'observateur comme étant l'ensemble des capteurs et la relation d'observation comme l'ensemble des mécanismes de propagation des messages de monitoring.

Pour que $\mathcal{R}_O$ soit une relation d'observation valide, il faut qu'elle préserve les configurations du système S. Cela est important car le système sous diagnostic doit préserver le comportement du système de départ. Ainsi nous avons :

$$Q \subseteq [\![\mathcal{R}_O]\!][V] \tag{3.1}$$

Le système sous diagnostic forme alors un nouveau système de transitions étiquetées contraint par une relation d'observation. Ce nouveau modèle, entièrement défini par le système $S = \langle Q, Q_i, E, \rightarrow \rangle$, l'observateur O et la relation d'observation $\mathcal{R}_O$, peut être représenté par le LTS $A = \langle C, I, E, T \rangle$ où :

- $C = \{c \in [\![\mathcal{R}_O]\!] \mid c[V] \in Q\}$ est un ensemble de configurations.
- $I = \{c \in C \mid c[V] \in Q_i\}$ est l'ensemble des configurations initiales.
- E est l'ensemble des événements.
- $T = \{(c, e, c') \in C \times E \times C \mid c[V] \xrightarrow{e} c'[V]\}$ est un ensemble de transitions.

L'ensemble des configurations $C \subseteq \mathcal{V}(V \uplus V_O)$ comprend toutes les valuations de variables c de $V \uplus V_O$ qui vérifient les contraintes de la relation d'observation $\mathcal{R}_O$ et telles qu'il existe une configuration q de Q pour laquelle $q = c[V]$. On retrouvera par la suite fréquemment cette relation qui lie les configurations q et c, il est donc intéressant de la caractériser.

Définition 4 (Projection/Élévation d'une configuration) *Soient C_1 et C_2 deux ensembles de configurations, et V_1 un ensemble de variables. Une configuration c_1 de C_1 est la* projection *d'une configuration c_2 de C_2 si $c_1 = c_2[V_1]$. On dit alors aussi que c_2 est une* élévation *de c_1.*

Lemme 1 $C[V] = Q$

Preuve : On applique la restriction à l'ensemble des variables V sur la définition de C.

$$\begin{array}{lll} C[V] & = \{c \in [\![\mathcal{R}_O]\!] \mid c[V] \in Q\}[V] & (\text{déf. } C) \\ & = \{c \in [\![\mathcal{R}_O]\!][V] \mid c[V] \in Q\} & (V \subseteq V) \\ & = \{q \in [\![\mathcal{R}_O]\!][V] \mid q \in Q\} & \\ & = \{q \in Q\} & (\text{éq. } 3.1) \\ & = Q & \end{array}$$

$\Diamond$

L'ensemble des configurations initiales $I \subseteq C$ pour A est entièrement défini par les configurations initiales du système S.

L'ensemble des événements E pour A est le même que celui du système sans observateur.

L'ensemble $T \subseteq (C \times E \times C)$ des transitions de A est similaire à la relation de transition du système d'origine. Il découle de la définition de T la propriété suivante.

Corollaire 1 *Considérons deux configurations q, q' de Q. Pour toutes configurations c, c' de C élévations de q, q', $q \xrightarrow{e} q' \iff (c, e, c') \in T$.*

Grâce à ce corollaire et au lemme 1, nous pouvons introduire un nouveau lemme caractérisant le système de transitions T.

Lemme 2 *Considérons deux configurations q, q' de Q. Si $q \xrightarrow{e} q'$ alors il existe deux configurations c, c' dans C respectivement élévations de q et q' telles que $(c, e, c') \in T$.*

Preuve : D'après le lemme 1, toute configuration de Q a une projection dans C. Ainsi

$$\forall (q, q') \in Q^2, \exists (c, c') \in C^2, q = c[V] \wedge q' = c'[V]$$

et donc par définition de l'ensemble T

$$\forall (q, q') \in Q^2, q \xrightarrow{e} q' \implies \exists (c, c') \in C^2, q = c[V] \wedge q' = c'[V] \wedge (c, e, c') \in T$$

$\Diamond$

Nous allons maintenant montrer qu'un système S et un système sous diagnostic A, construit à partir de S, sont bisimilaires. Nous introduisons donc la notion de bisimilarité de systèmes.

Définition 5 (Bisimilarité) *Soient deux LTS $\alpha = \langle X, X_i, \Sigma, \rightarrow_\alpha \rangle$ et $\beta = \langle Y, Y_i, \Sigma, \rightarrow_\beta \rangle$. α et β sont bisimilaires s'il existe une relation $\sim \subseteq X \times Y$ telle que pour tout couple (x, y) de $X \times Y$, si $x \sim y$ alors pour tout σ dans Σ :*

$$\begin{cases} \forall x' \in X, x \xrightarrow{\sigma}_\alpha x' \implies \exists y' \in Y, y \xrightarrow{\sigma}_\beta y' \wedge x' \sim y' & (3.2) \\ \forall y' \in Y, y \xrightarrow{\sigma}_\beta y' \implies \exists x' \in X, x \xrightarrow{\sigma}_\alpha x' \wedge x' \sim y' & (3.3) \end{cases}$$

En outre il faut vérifier que :

$$\begin{cases} \forall x_i \in X_i, \exists y_i \in Y_i, x_i \sim y_i & (3.4) \\ \forall y_i \in Y_i, \exists x_i \in X_i, x_i \sim y_i & (3.5) \end{cases}$$

Théorème 1 *Soit S un système. Tout système sous diagnostic $A = \langle S, O, \mathcal{R}_O \rangle$ vérifie que A et S sont bisimilaires.*

Preuve : Soient $S = \langle Q, Q_i, E, \rightarrow \rangle$ un système et $A = \langle C, I, E, T \rangle$ un système sous diagnostic. Considérons la relation de projection $\mathcal{R} \subseteq Q \times C$ telle que pour tout couple (q, c) de $Q \times C$, $(q, c) \in \mathcal{R} \iff q = c[V]$.

Considérons $(q, c) \in Q \times C$ tels que $(q, c) \in \mathcal{R}$ et e un événement quelconque de E :

- pour tout q' de Q :

$$\begin{array}{lll} q \xrightarrow{e} q' & \Longrightarrow \exists (c'', c') \in C^2, q = c''[V] \wedge q' = c'[V] \wedge (c'', e, c') \in T & \text{(lem. 2)} \\ & \Longrightarrow \exists c' \in C, q' = c'[V] \wedge (c, e, c') \in T & \text{(cor. 1)} \\ & \Longrightarrow \exists c' \in C, (q', c') \in \mathcal{R} \wedge (c, e, c') \in T & \end{array}$$

- pour tout c' de C :

$$\begin{array}{lll} (c, e, c') \in T & \Longrightarrow c[V] \xrightarrow{e} c'[V] & \text{(déf. } T\text{)} \\ & \Longrightarrow \exists q' \in Q, q' = c'[V], c[V] \xrightarrow{e} q' & \text{(lem. 1)} \\ & \Longrightarrow \exists q' \in Q, q' = c'[V], q \xrightarrow{e} q' & ((q, c) \in \mathcal{R}) \\ & \Longrightarrow \exists q' \in Q, (q', c') \in \mathcal{R} \wedge q \xrightarrow{e} q' & \end{array}$$

Si nous regardons maintenant les configurations initiales :

$$\begin{array}{ll} & \forall c_i \in I, c_i[V] \in Q_i \qquad \text{(déf. } I\text{)} \\ \Longrightarrow & \forall c_i \in I, \exists q_i \in Q_i, q_i = c_i[V] \\ \Longrightarrow & \forall c_i \in I, \exists q_i \in Q_i, (q_i, c_i) \in \mathcal{R} \end{array}$$

$$\begin{array}{ll} & \forall q_i \in Q, \exists c_i \in C, q_i = c_i[V] \qquad \text{(lem. 1)} \\ \Longrightarrow & \forall q_i \in Q_i, \exists c_i \in C, q_i = c_i[V] \\ \Longrightarrow & \forall q_i \in Q_i, \exists c_i \in I, q_i = c_i[V] \qquad \text{(déf. I)} \\ \Longrightarrow & \forall q_i \in Q_i, \exists c_i \in I, (q_i, c_i) \in \mathcal{R} \end{array}$$

$\Diamond$

Dire que le système S et le système sous diagnostic A sont bisimilaires signifie qu'ils se comportent de la même façon. Le retrait de l'observateur et des variables d'observation n'impacte pas le comportement intrinsèque du système. Cela permet de changer et comparer différents observateurs pour un même système.

3.3 Décomposition d'un SUD

Nous venons de montrer que si A est un système sous diagnostic construit à partir d'un système S de la façon décrite précédemment, alors A et S sont bisimilaires.

Nous allons voir maintenant qu'étant donné un système α, il est possible d'isoler un ensemble de variables V_r pour obtenir, sous réserve de répondre à certaines propriétés, un nouveau système β. Nous allons montrer que le système α est alors un système sous diagnostic obtenu à partir de β, d'un observateur O_r et d'une relation d'observation $\mathcal{R}_X$.

Définissons tout d'abord la notion de système projeté.

Définition 6 (Système projeté) *Considérons le LTS $\alpha = \langle X, X_i, \Sigma, \rightarrow_\alpha \rangle$ et l'ensemble de variables V^α tels que :*

- *$X \subseteq \mathcal{V}(V^\alpha)$ est un ensemble de configurations.*
- *$X_i \subseteq X$ est l'ensemble des configurations initiales.*
- *Σ est un ensemble d'événements.*
- *$\rightarrow_\alpha \subseteq (X \times \Sigma \times X)$ est une relation de transition.*

Soit $V_r \subseteq V^\alpha$ un ensemble de variables. Le système projeté représenté par le LTS $\beta = \langle Y, Y_i, \Sigma, \rightarrow_\beta \rangle$ est obtenu par restriction du système α sur $V^\beta = V^\alpha \setminus V_r$:

- $Y = X[V^\beta]$.
- $Y_i = X_i[V^\beta]$
- Σ *est le même ensemble d'événements.*
- $\rightarrow_\beta = \{t \in Y \times \Sigma \times Y \mid \exists (x, \sigma, x') \in \rightarrow_\alpha, t = (x[V^\beta], \sigma, x'[V^\beta])\}$.

On notera $\beta = proj(\alpha, V^\beta)$.

L'idée est maintenant de montrer que α est un système sous diagnostic vis-à-vis du système projeté β. Mais avant cela nous avons besoin de caractériser certaines propriétés particulières d'un système, qui, nous le verrons, sont nécessaires pour démontrer ce point.

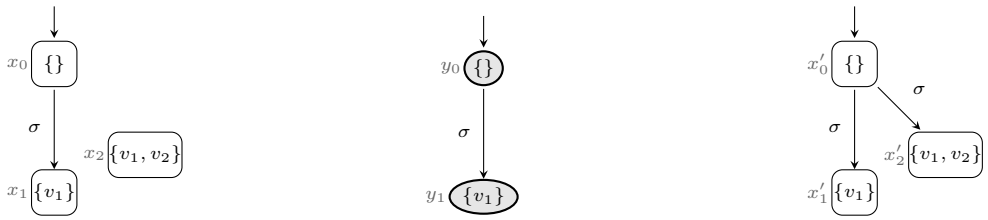
Propriété 1 (Complétude des transitions) *Soient un système* $\alpha = \langle X, X_i, \Sigma, \rightarrow_\alpha \rangle$ *et un ensemble de variables* V_β*. La propriété de complétude des transitions affirme que toutes les configurations de* X *qui ont la même projection sur* V_β *ont les mêmes transitions entrantes et sortantes. Elle est représentée par l'équation suivante :*

$$\begin{aligned} \forall (x, x') \in X^2, \forall \sigma \in \Sigma, (x[V^\beta] = x'[V^\beta]) \implies (\forall x'' \in X, x'' \xrightarrow{\sigma}_\alpha x \iff x'' \xrightarrow{\sigma}_\alpha x' \\ \wedge\, x \xrightarrow{\sigma}_\alpha x'' \iff x' \xrightarrow{\sigma}_\alpha x'') \end{aligned} \tag{3.6}$$

Cette propriété est importante car montrer que $\alpha = \langle \beta, O_r, \mathcal{R}_X \rangle$ signifie que lorsque nous calculons $\beta = proj(\alpha, V^\beta)$ et que nous construisons le système sous diagnostic à partir de β, O_r et $\mathcal{R}_X$, alors nous obtenons α. Le problème que l'on rencontre lorsque la propriété 1 n'est pas vérifiée est le suivant : en créant β, il y a des configurations de α ayant la même projection qui n'ont pas les mêmes transitions entrantes ou sortantes. Pourtant, la configuration y de β obtenue par projection aura elle l'ensemble de toutes les transitions. Ainsi, si l'on reconstruit un système sous diagnostic à partir de β, les configurations de α (élévations de y) susmentionnées seront cibles de toutes les transitions entrantes de y, ainsi que sources de toutes les transitions sortantes de y. Le système sous diagnostic obtenu sera alors différent de α.

On peut aussi faire la remarque que cette propriété n'est pas couverte par la notion de bisimilarité.

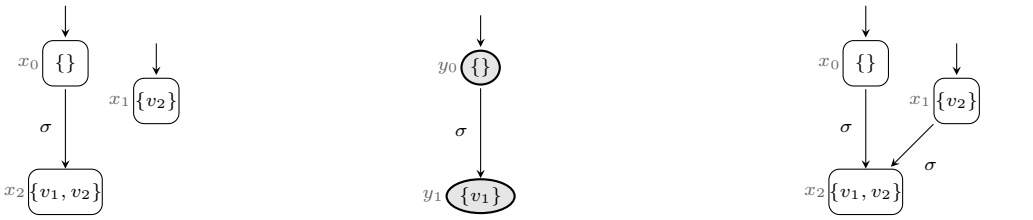
Nous allons voir deux exemples qui montrent la nécessité de la propriété précédente. Ces exemples, représentés en figures 3.1 et 3.2, illustrent l'explication que nous avons fournie dans le paragraphe précédent. La première figure (3.1) illustre la nécessité de la propriété pour les transitions entrantes, et la deuxième (3.2) s'intéresse aux transitions sortantes.



A. Système α B. Système $\beta = proj(\alpha, V^\beta)$ C. Système $\langle \beta, O_r, \mathcal{R}_X \rangle$

FIGURE 3.1 – Illustration de la propriété 1 – transitions entrantes

Considérons le système α et son ensemble de variables $V^\alpha = \{v_1, v_2\}$ tels que représentés en A. On obtient par projection de α sur l'ensemble de variables $V^\beta = \{v_1\}$ le système β représenté en B. On remarque que les deux configurations x_1 et x_2, dont –uniquement– la première est cible d'une transition ayant pour source x_0, ont la même projection y_1. Finalement, en prenant O_r défini par l'ensemble de variables $V_r = V^\alpha \setminus V^\beta = \{v_2\}$ et $\mathcal{R}_X \subseteq BF(V^\alpha)$ telle que $[\![\mathcal{R}_X]\!] = X$, le système sous diagnostic $\langle \beta, O_r, \mathcal{R}_X \rangle$ obtenu et représenté en C est différent du système α de départ. On note en outre que les systèmes α et β sont bisimilaires.



A. Système α B. Système $\beta = proj(\alpha, V^\beta)$ C. Système $\langle \beta, O_r, \mathcal{R}_X \rangle$

FIGURE 3.2 – Illustration de la propriété 1 – transitions sortantes

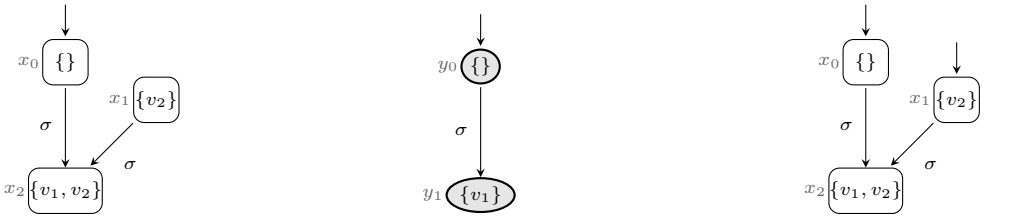
La figure 3.2 donne l'exemple d'un système α ne répondant pas à la partie "transitions sortantes" de la propriété 1. L'idée de cet exemple est identique à celle de l'exemple précédent, adaptée aux transitions sortantes au lieu des transitions entrantes.

Une deuxième propriété est nécessaire pour les mêmes raisons que la propriété 1 précédemment énoncée. C'est une propriété proche de la précédente mais qui concerne les états initiaux plutôt que les transitions.

Propriété 2 (Complétude des états initiaux) *Soient un système $\alpha = \langle X, X_i, \Sigma, \rightarrow_\alpha \rangle$ et un ensemble de variables V_β. La propriété de complétude des états initiaux affirme que si une configuration x_i de X est initiale, alors toutes les autres configurations qui ont la même projection que x_i sur V_β sont aussi initiales. Elle est représentée par l'équation suivante :*

$$\forall (x_i, x) \in X_i \times X, (x_i[V^\beta] = x[V^\beta]) \implies x \in X_i) \tag{3.7}$$

L'importance de cette propriété se justifie d'une façon identique à la propriété précédente, en considérant le caractère initial d'une configuration plutôt que ses transitions entrantes. Derechef, la notion de bisimilarité ne couvre pas cette propriété. On peut voir cette propriété illustrée en figure 3.3.



A. Système α B. Système $\beta = proj(\alpha, V^\beta)$ C. Système $\langle \beta, O_r, \mathcal{R}_X \rangle$

FIGURE 3.3 – Illustration de la propriété 2

L'exemple que nous présentons maintenant est similaire à l'illustration proposée en figure 3.1 et à sa description. Nous considérons le système α et son ensemble de variables $V^\alpha = \{v_1, v_2\}$ tels que représentés en A. Par projection de α sur l'ensemble de variables $V^\beta = \{v_1\}$, nous obtenons un nouveau système β représenté en B. On peut remarquer que les deux configurations x_0 et x_1, dont –uniquement– la première est initiale, ont la même projection y_0 dans β. De la même façon que pour la propriété précédente, si on prend O_r défini par l'ensemble de variables $V_r = V^\alpha \setminus V^\beta = \{v_2\}$ et $\mathcal{R}_X \subseteq BF(V^\alpha)$ telle que $[\![\mathcal{R}_X]\!] = X$, le système sous diagnostic $\langle \beta, O_r, \mathcal{R}_X \rangle$ obtenu et représenté en C est à nouveau différent du système α de départ. Là aussi les systèmes α et β sont pourtant bisimilaires.

Les exemples que nous venons de présenter montrent la nécessité des propriétés 1 et 2 pour notre problème. En effet, nous venons de voir que pour un ensemble de variables V_r, le système

$\langle proj(\alpha, V^\alpha \setminus V_r), O_r, \mathcal{R}_X \rangle$ où O_r est défini par l'ensemble de variables V_r et où $\mathcal{R}_X \subseteq BF(V^\alpha)$ est telle que $[\![\mathcal{R}_X]\!] = X$, est un système sous diagnostic (par construction) qui est différent de α si ce dernier ne respecte pas les propriétés énoncées.

Ayant vu la nécessité de ces propriétés, nous allons maintenant regarder si elles sont suffisantes pour montrer que α est un système sous diagnostic vis-à-vis du système projeté β. Commençons tout d'abord par introduire un nouveau lemme.

Lemme 3 *Supposons que α vérifie la propriété 1 pour V^β. Considérons deux configurations x, x' de X. Pour toutes configurations y, y' de Y, projections respectives de x et x', $x \xrightarrow{\sigma}_\alpha x' \iff y \xrightarrow{\sigma}_\beta y'$.*

Preuve : Montrons la double implication.

$\boxed{\Longrightarrow}$ $\forall (x, x') \in X^2, \forall \sigma \in \Sigma, x \xrightarrow{\sigma}_\alpha x'$
$\Longrightarrow \exists (y, y') \in Y^2, y = x[V^\beta], y' = x'[V^\beta], y \xrightarrow{\sigma}_\beta y'$ (déf. $\to_\beta$)
or les élévations y et y' sont uniques donc
$\Longrightarrow \forall (y, y') \in Y^2, y = x[V^\beta], y' = x'[V^\beta], y \xrightarrow{\sigma}_\beta y'$

$\boxed{\Longleftarrow}$ $\forall (y, y') \in Y^2, \forall \sigma \in \Sigma, y \xrightarrow{\sigma}_\beta y'$
$\Longrightarrow \exists (x, x') \in X^2, y = x[V^\beta], y' = x'[V^\beta], x \xrightarrow{\sigma}_\alpha x'$ (déf. $\to_\beta$)

$$\text{or } \forall (x_1, x_1') \in X^2, \begin{cases} x[V^\beta] = x_1[V^\beta] \\ x'[V^\beta] = x_1'[V^\beta] \end{cases} \Longrightarrow \begin{cases} x \xrightarrow{\sigma}_\alpha x' \\ x_1 \xrightarrow{\sigma}_\alpha x' \\ x \xrightarrow{\sigma}_\alpha x_1' \\ x_1 \xrightarrow{\sigma}_\alpha x_1' \end{cases} \quad \text{(prop. 1)}$$

$\Longrightarrow \forall (x, x') \in X^2, y = x[V^\beta], y' = x'[V^\beta], x \xrightarrow{\sigma}_\alpha x'$ (éq. 3.6)

$\Diamond$

Théorème 2 *Soient $\alpha = \langle X, X_i, \Sigma, \to_\alpha \rangle$ un système et V^β un ensemble de variables de α. Soit $\beta = \langle Y, Y_i, \Sigma, \to_\beta \rangle$ le projeté de α sur V^β. Si α vérifie les propriétés 1 et 2 pour V^β, alors $\alpha = \langle \beta, O_r, \mathcal{R}_X \rangle$ est un système sous diagnostic.*

Preuve : Considérons l'observateur O_r défini par l'ensemble de variables $V_r = V^\alpha \setminus V^\beta$ et la relation $\mathcal{R}_X \subseteq BF(V^\alpha)$ telle que $[\![\mathcal{R}_X]\!] = X$. Montrons que $\alpha = \langle \beta, O_r, \mathcal{R}_X \rangle$ est un système sous diagnostic.

De manière triviale, la relation vérifie $Y \subseteq [\![\mathcal{R}_X]\!][V^\beta]$ puisque par construction, $Y = X[V^\beta]$.

Pour être un système sous diagnostic, $\alpha = \langle X, X_i, \Sigma, \to_\alpha \rangle$ doit respecter les règles de construction suivantes :

1. $X \stackrel{?}{=} \{x \in [\![\mathcal{R}_X]\!] \mid x[V^\beta] \in Y\}$.
2. $X_i \stackrel{?}{=} \{x \in X \mid x[V^\beta] \in Y_i\}$.
3. Σ est le même ensemble d'événements.
4. $\to_\alpha \stackrel{?}{=} \{(x, \sigma, x') \in X \times \Sigma \times X \mid x[V^\beta] \xrightarrow{\sigma}_\beta x'[V^\beta]\}$.

Vérifions chacune de ces règles :

1. $\begin{aligned} \{x \in [\![\mathcal{R}_X]\!] \mid x[V^\beta] \in Y\} &= \{x \in X \mid x[V^\beta] \in Y\} \\ &= X \qquad \text{(déf. } Y\text{)} \end{aligned}$

2. $\{x \in X \mid x[V^\beta] \in Y_i\} = X_i$ (déf. Y_i (existence) et prop. 2 (complétude))

3. Trivial.

4.
$$\begin{array}{lll} & \{(x,\sigma,x') \in X \times \Sigma \times X & \mid \quad x[V^\beta] \xrightarrow{\sigma}_\beta x'[V^\beta]\} \\ = & \{(x,\sigma,x') \in X \times \Sigma \times X & \mid \quad \exists (y,y') \in Y^2, y = x[V^\beta], \\ & & \quad\ y' = x'[V^\beta], y \xrightarrow{\sigma}_\beta y'\} \\ = & \{(x,\sigma,x') \in X \times \Sigma \times X & \mid \quad x \xrightarrow{\sigma}_\alpha x'\} \qquad \text{(lem. 3)} \\ = & \to_\alpha & \end{array}$$

◊

Corollaire 2 *Les systèmes $\alpha = \langle \beta, O_r, \mathcal{R}_X \rangle$ et $\beta = proj(\alpha, V^\beta)$ sont bisimilaires.*

Nous pouvons résumer à travers la figure 3.4 les relations entre les différents objets que nous venons d'introduire.

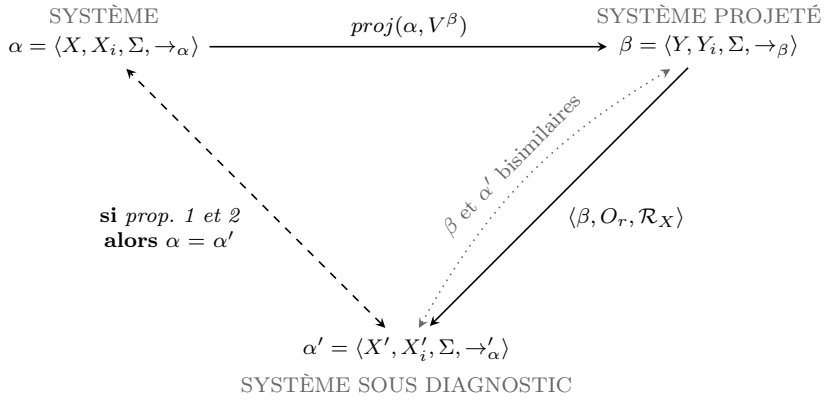


FIGURE 3.4 – Relations entre Système, Système Projeté et SUD

Le premier système que nous avons introduit en section 3.1 est représenté par le LTS $S = \langle Q, Q_i, E, \to \rangle$. À partir du système S, d'un observateur O et d'une relation d'observation $\mathcal{R}_O$ répondant à certaines propriétés, nous avons montré en section 3.2 comment construire le système sous diagnostic $A = \langle C, I, E, T \rangle$. Cette construction, notée $A = \langle S, O, \mathcal{R}_O \rangle$, assure la bisimilarité des systèmes S et A. Nous retrouvons tout cela représenté sur la partie droite de la figure 3.4. S joue ainsi le rôle de système projeté, et A de système sous diagnostic construit par $\langle S, O, \mathcal{R}_O \rangle$.

Dans la partie 3.3, nous avons montré comment il est possible d'isoler un sous-ensemble de variables V_r d'un système $\alpha = \langle X, X_i, \Sigma, \to_\alpha \rangle$, et comment obtenir le système projeté $\beta = \langle Y, Y_i, \Sigma, \to_\beta \rangle$. Finalement nous avons vu que si V_r est choisi de telle manière à ce que $V^\beta = V^\alpha \setminus V_r$ associé à α respecte les propriétés de complétude, alors le système obtenu par $\langle \beta, O_r, \mathcal{R}_X \rangle$, où l'observateur O_r est défini par l'ensemble de variables V_r et la relation $\mathcal{R}_X$ est telle que $[\![\mathcal{R}_X]\!] = X$, n'est autre que α. On peut alors en déduire que α est un système sous diagnostic.

Notation 1 *Dans la suite du document, afin d'éviter les lourdeurs dans les définitions, nous utiliserons toujours S pour désigner le système $S = \langle Q, Q_i, E, \to \rangle$ de la définition 1, et A pour désigner le système sous diagnostic $A = \langle C, I, E, T \rangle = \langle S, O, \mathcal{R}_O \rangle$ de la définition 3. De même, les ensembles V pour S ainsi que V_O et $\mathcal{R}_O$ pour A seront réutilisés.*

3.4 Retrait de variables d'observation d'un SUD

Nous nous intéressons dans cette section à une autre opération sur les systèmes sous diagnostic : le retrait de variables d'observation.

Nous avons montré dans la section précédente et résumé en figure 3.4 les relations entre *Système*, *Système projeté* et *Système sous diagnostic*. Nous voulons montrer ici qu'étant donné un système de départ, retirer des variables d'observation d'un système sous diagnostic construit à partir de ce système de départ permet d'obtenir un nouveau système sous diagnostic dont le système de départ est le même.

Dans ce but, commençons par définir le système obtenu par suppression de variables d'observation d'un système sous diagnostic. Pour cela, nous nous appuyons sur le système S et sur le système sous diagnostic A.

Définition 7 (SUD privé de variables d'observation) *Soit $V_s \subseteq V_O$ un ensemble de variables d'observation à supprimer. Pour $V^\gamma = V \uplus (V_O \setminus V_s)$, le système $\gamma = \langle Z, Z_i, E, \rightarrow_\gamma \rangle$ obtenu à partir de A auquel on a retiré V_s correspond à la projection de A sur V^γ : $\gamma = proj(A, V^\gamma)$.*

On notera alors $\gamma = A \setminus V_s$.

Nous venons de définir le système obtenu par retrait de variables d'observation d'un système sous diagnostic. Montrons que le système obtenu a lui aussi la propriété d'être un système sous diagnostic du même système de départ.

Théorème 3 *Soit $\gamma = \langle Z, Z_i, E, \rightarrow_\gamma \rangle$ le système obtenu par retrait de l'ensemble de variables d'observation V_s du système sous diagnostic $A = \langle C, I, E, T \rangle$. γ est tel que $\gamma = \langle S, O_\gamma, \mathcal{R}_\gamma \rangle$ où O_γ est défini par l'ensemble de variables d'observation $V_{O_\gamma} = V_O \setminus V_s$ et la relation $\mathcal{R}_\gamma \subseteq BF(V^\gamma)$ est telle que $[\![\mathcal{R}_\gamma]\!] = [\![\mathcal{R}_O]\!][V^\gamma]$.*

Preuve : Montrons que $\gamma = \langle S, O_\gamma, \mathcal{R}_\gamma \rangle$ est un système sous diagnostic.

Rappelons que $S = \langle Q, Q_i, E, \rightarrow \rangle$.

On sait déjà que $Q \subseteq [\![\mathcal{R}_O]\!][V]$ (éq. 3.1). On a donc

$$\begin{array}{lll} Q \subseteq [\![\mathcal{R}_O]\!][V] & \Longrightarrow & Q \subseteq ([\![\mathcal{R}_O]\!][V^\gamma])[V] \quad (V \subseteq V^\gamma) \\ & \Longrightarrow & Q \subseteq [\![\mathcal{R}_\gamma]\!][V] \quad (\text{déf } \mathcal{R}_\gamma) \end{array}$$

Pour être un système sous diagnostic, $\gamma = \langle Z, Z_i, E, \rightarrow_\gamma \rangle$ doit respecter les règles de construction suivantes :

1. $Z \stackrel{?}{=} \{z \in [\![\mathcal{R}_\gamma]\!] \mid z[V] \in Q\}$.
2. $Z_i \stackrel{?}{=} \{z \in Z \mid z[V] \in Q_i\}$.
3. E est le même ensemble d'événements.
4. $\rightarrow_\gamma \stackrel{?}{=} \{(z, e, z') \in Z \times E \times Z \mid z[V] \stackrel{e}{\rightarrow} z'[V]\}$.

Vérifions chacune de ces règles :

1. $$\begin{array}{lll} & C = \{c \in [\![\mathcal{R}_O]\!] \mid c[V] \in Q\} & (\text{déf } C) \\ \Longrightarrow & C[V^\gamma] = \{c \in [\![\mathcal{R}_O]\!] \mid c[V] \in Q\}[V^\gamma] & \\ \Longrightarrow & C[V^\gamma] = \{c \in [\![\mathcal{R}_O]\!][V^\gamma] \mid c[V] \in Q\} & (V \subseteq V^\gamma) \\ \Longrightarrow & C[V^\gamma] = \{c \in [\![\mathcal{R}_\gamma]\!] \mid c[V] \in Q\} & (\text{déf } \mathcal{R}_\gamma) \\ \Longrightarrow & Z = \{z \in [\![\mathcal{R}_\gamma]\!] \mid z[V] \in Q\} & (\text{déf } Z) \end{array}$$

2. $$\begin{array}{lll} & I = \{c \in C \mid c[V] \in Q_i\} & (\text{déf } I) \\ \Longrightarrow & I[V^\gamma] = \{c \in C \mid c[V] \in Q_i\}[V^\gamma] & \\ \Longrightarrow & I[V^\gamma] = \{c \in C[V^\gamma] \mid c[V] \in Q_i\} & (V \subseteq V^\gamma) \\ \Longrightarrow & I[V^\gamma] = \{c \in Z \mid c[V] \in Q_i\} & (\text{déf } Z) \\ \Longrightarrow & Z_i = \{z \in Z \mid z[V] \in Q_i\} & (\text{déf } Z_i) \end{array}$$

3. Trivial

4. $\rightarrow_\gamma$ $= \{t_\gamma \in Z \times E \times Z \mid \exists (c, e, c') \in T,\ t_\gamma = (c[V^\gamma], e, c'[V^\gamma])\}$ (déf. $\rightarrow_\gamma$) $\Diamond$
$= \{(z, e, z') \in Z \times E \times Z \mid \exists (c, c') \in C^2, z = c[V^\gamma]$
$\wedge\ z' = c'[V^\gamma] \wedge c[V] \xrightarrow{e} c'[V]\}$ (déf. T)
$= \{(z, e, z') \in Z \times E \times Z \mid \exists (c, c') \in C^2, z = c[V^\gamma]$
$\wedge\ z' = c'[V^\gamma] \wedge (c[V^\gamma])[V] \xrightarrow{e} (c'[V^\gamma])[V]$ ($V \subseteq V^\gamma$)
$= \{(z, e, z') \in Z \times E \times Z \mid \exists (c, c') \in C^2, z = c[V^\gamma]$
$\wedge\ z' = c'[V^\gamma] \wedge z[V] \xrightarrow{e} z'[V]\}$
$= \{(z, e, z') \in Z \times E \times Z \mid z[V] \xrightarrow{e} z'[V]\}$ (déf. Z)

Ce qui est important dans cette preuve, et ce qui la rend possible, c'est que les variables supprimées V_s sont des variables d'observations, et donc plus exactement que l'ensemble des variables restantes V^γ est un sur-ensemble de l'ensemble des variables V du système de départ S.

Le fait de conserver toutes les variables système et de n'enlever que des variables d'observation permet d'obtenir la même configuration $c[V]$ que c soit un élément de $[\![\mathcal{R}_O]\!]$ que l'on projette ensuite sur V^γ, ou qu'il soit directement un élément de $[\![\mathcal{R}_O]\!][V^\gamma]$. Cette propriété a été utilisée pour les points 1 et 2 de la preuve.

La propriété $V \subseteq V^\gamma$ est aussi appliquée pour prouver le point 4. On utilise alors le fait que $\forall c \in C, c[V] = c[V^\gamma][V]$.

3.5 Observations d'un SUD

Un système sous diagnostic est donc composé d'une partie système qui décrit son comportement et d'une relation d'observation qui permet, couplée à un observateur, de surveiller l'état du système. De cela découle la notion d'observation.

Définition 8 (Observation) *Une observation est décrite par une contrainte sur les variables d'observation. L'ensemble des observations possibles est noté Obs.*

$$Obs \subseteq BF(V_O)$$

Une observation est définie par une contrainte car elle ne correspond pas toujours à une valuation de toutes les variables de V_O. En effet, on peut ne pas connaître la valeur de toutes les variables d'observation du système sous surveillance, ce qui ne permet d'obtenir qu'une valuation partielle. Cette valuation partielle peut alors être exprimée par une formule booléenne sur V_O.

Si l'on se trouve dans le cas de valuations totales, on obtient $Obs \subseteq \mathcal{V}(V_O)$. Les observations correspondent alors à la partie observable des configurations de A et on a $Obs \subseteq C[V_O]$.

Maintenant que nous avons défini le modèle de système observé avec lequel nous travaillons, nous pouvons nous intéresser au problème du diagnostic sur ce modèle. Étant donnée $obs_{diag} \in Obs$ une observation représentant la situation observée, effectuer un diagnostic consiste à identifier les événements qui ont pu provoquer obs_{diag}. Ces explications peuvent être exprimées sous différentes formes : par un ensemble de séquences d'événements, par un ensemble de coupes d'événements, d'une forme minimale ou pas et avec ou sans probabilités, ou simplement encore par exemple par le dernier événement provoquant obs_{diag}. Nous décrirons certaines de ces notions dans la prochaine partie.

Un objectif du travail est de formaliser la notion de qualité d'un observateur pour le diagnostic. Nous définirons donc par la suite des critères :

- Quantitatifs → qui permettent de mesurer les caractéristiques d'un observateur.
- Qualitatifs → qui permettent de comparer différents observateurs et de définir un ordre entre eux.

D'un point de vue diagnostic, un observateur parfait est un observateur qui fournira uniquement des informations nécessaires au diagnostiqueur, lui permettant d'effectuer un diagnostic sans ambiguité, d'identifier de manière claire les événements ayant causé l'observation. Le cas qui assure un observateur parfait est celui où l'on a une observation différente reçue pour chacun des événements possibles, la relation entre les observations et les événements formant ainsi une relation de bijection. Une façon que l'on peut imaginer pour obtenir le cas parfait est de tout surveiller, d'attacher à chaque événement possible un mécanisme d'observation.

Un tel observateur n'est jamais mis en place en pratique et cela pour plusieurs raisons :

- Il est en général physiquement impossible de surveiller tous les dysfonctionnements d'un système.
- Les procédés d'observation d'un système mêlant sondes, comparateurs, etc. apportent souvent des contraintes (dimensions, poids, coût...).
- Certains événements peuvent ne pas être discriminés au niveau de leurs effets, empêchant de ce fait toute relation bijective entre les observations et les événements.
- De par des contraintes de sûreté et de sécurité, une redondance peut être requise sur certains monitoring.

Finalement, un bon observateur est un observateur capable de délivrer des informations suffisantes au diagnostiqueur pour que ce dernier puisse fournir des diagnostics avec peu d'ambiguïté. Dans la suite du document, nous nous intéresserons donc à définir et évaluer la qualité des informations fournies par l'observateur.

3.6 Caractérisation des résultats de diagnostic

Nous avons dit dans la partie précédente que les résultats de diagnostic pouvaient être de plusieurs formes. Nous définissons dans cette partie quelques unes de ces différentes formes d'explications.

Introduisons tout d'abord $E_v \subseteq E$, l'ensemble des événements *visibles*. Le fait qu'un événement soit défini comme visible implique qu'il pourra apparaître dans les résultats de diagnostic comme cause. À l'opposé, les événements *non visibles* ne feront pas partie des causes possibles et seront ignorés. En pratique, les événements visibles, que l'on veut voir présents dans les résultats de diagnostic, sont souvent les défaillances, qui sont les causes des effets observés.

Dans le cadre d'un problème de diagnostic, étant donnée une observation *obs*, nous voulons identifier les événements qui amènent le système dans les configurations où l'on observe *obs*. Ces configurations sont appelées *configurations cibles*.

Définition 9 (Configurations cibles) *Soit obs l'observation à diagnostiquer.* $C_{obs} = C \cap [\![obs]\!]$ *est l'ensemble des configurations vérifiant obs.*

Un chemin est une séquence finie $p = c_0, e_0, \ldots, c_n$ où $c_{i|i\in[\![0;n]\!]}$ est une configuration de C, $e_{i|i\in[\![0;n-1]\!]}$ un événement de E et pour $i \in [\![0; n-1]\!]$, $c_{i+1} \in succ(c_i, e_i)$. n est la longueur du chemin. On note c_0 par $first(p)$ et c_n par $last(p)$. Le mot $\lambda(p) = e_0 \ldots e_{n-1} \in E^*$ est appelé *trace* de p. Si la longueur d'un chemin est 0, alors sa trace est le mot vide ϵ. L'ensemble des chemins d'un LTS A est noté $Path(A)$.

Intéressons-nous maintenant à l'ensemble des traces menant le système dans un ensemble de configurations. Étant donné un LTS $A = \langle C, I, E, T\rangle$ et un ensemble de configurations $C_1 \subseteq C$, l'ensemble des traces menant à C_1 est défini par :

$$L(A, C_1) = \{\lambda(p) \mid p \in Path(A) \wedge first(p) \in I \wedge last(p) \in C_1\}$$

Les éléments de $L(A, C_1)$ sont des événements de E. Si on considère une observation *obs*, $L(A, C_{obs})$ représente l'ensemble des traces menant à une configuration cible.

Nous expliquions un peu plus haut qu'il peut être intéressant de ne pas faire apparaître tous les événements dans les résultats. Le morphisme de visibilité $\Phi_v : E^* \to E_v^*$ est défini de la manière suivante :

- $\Phi_v(\epsilon) = \epsilon$
- $\Phi_v(e) = \epsilon$ si $e \notin E_v$
- $\Phi_v(e) = e$ si $e \in E_v$
- $\Phi_v(w_1.w_2) = \Phi_v(w_1).\Phi_v(w_2)$ pour tout w_1, w_2 de E^*.

Définition 10 (Séquence de défaillances) *Une séquence de défaillances est la trace d'un chemin $p = c_0, e_0, \ldots, c_n$ où c_0 est une configuration initiale et c_n est une configuration cible, dont on ne conserve que les événements visibles.*

Nous désignerons par la suite par séquences *les séquences de défaillances.*

Étant donné un LTS A et un ensemble de configurations C_1, on définit l'ensemble des séquences de A menant à C_1 par :

$$Sequences(A, C_1) = \{\Phi_v(w) \mid w \in L(A, C_1)\}$$

Calculer l'ensemble de toutes les séquences de défaillances pour un système peut s'avérer être une opération difficile d'un point de vue combinatoire. La première chose est que $Sequences(A, C_1)$ peut tout à fait être infini, dans le cas par exemple où un événement peut se répéter indéfiniment (boucle). Et même si le modèle est tel que l'ensemble des séquences est fini, il suffit que le modèle soit trop complexe pour que le calcul de toutes les séquences se révèle impossible.

Il est possible de gérer une partie des cas des ce type en définissant l'ensemble des séquences bornées en longueur par un entier k :

$$Sequences(A, C_1, k) = \{w \mid w \in Sequences(A, C_1) \wedge |w| \leq k\}$$

En considérant C_{obs} pour une observation *obs* dans A comme paramètre pour les séquences, nous pouvons obtenir l'ensemble des séquences de défaillances menant à une configuration cible. Ces séquences représentent alors les causes possibles de l'observation à diagnostiquer. C'est une façon d'exprimer les résultats de diagnostic.

Lors de l'établissement d'un diagnostic, il n'est pas forcément utile de savoir dans quel ordre sont arrivés les événements ayant provoqué la situation observée. Il est donc intéressant d'avoir un résultat qui ne prend en compte que la liste des événements pouvant être la cause de la situation observée. C'est ce type de résultats que décrit la notion de *coupe* de défaillances.

Définition 11 (Coupe de défaillances) *Une coupe de défaillances décrit l'ensemble des événements qui ont pu amener le système dans un ensemble de configuration donnée. Étant donnée une séquence $w = e_0 \ldots e_{n-1}$, $cut(w) = \{e \in E \mid \exists i \in [\![0; n-1]\!], e = e_i\}$ représente la coupe de w.*

Nous désignerons par la suite par coupes *les coupes de défaillances.*

Étant donné un LTS A et un ensemble de configurations C_1, on définit l'ensemble des coupes de A menant à C_1 par :

$$Cuts(A, C_1) = \{cut(w) \mid w \in Sequences(A, C_1)\}$$

Nous pouvons remarquer qu'avec la notion de coupes, les occurrences multiples des événements ainsi que leur ordre d'apparition ne sont pas conservés.

On peut à nouveau utiliser C_{obs}, où *obs* est une observation à diagnostiquer, comme paramètre pour le calcul des coupes afin d'obtenir l'ensemble des événements qui peuvent expliquer l'apparition de *obs*.

Lorsque nous cherchons à diagnostiquer une observation *obs*, les ensembles $Sequences(A, C_{obs})$ et $Cuts(A, C_{obs})$ peuvent contenir des éléments redondants pour le diagnostic. Dans le cadre de systèmes cohérents [BdSB84, KMS99, Sch04], le fait qu'un événement arrive alors que C_{obs} a déjà

été atteint ne peut ramener le système dans un mode nominal, i.e. dans une configuration qui peut être considérée comme une configuration de bon fonctionnement. Cette redondance d'information est considérée comme non importante par la communauté diagnostic et en particulier dans notre domaine, celui de la maintenance, dans lequel ces informations ne sont pas indispensables pour la réparation de la panne. Nous allons donc définir un objet qui ne conserve que les événements dits *implicants premiers*[CM93, RD97].

Nous pouvons donc appliquer ce principe aux ensembles de séquences et de coupes et introduire la notion de minimisation de ces ensembles. Les séquences sont minimisées par la relation d'ordre *sous-mot* ($\sqsubseteq$) et les coupes par l'*inclusion* ($\subseteq$).

Définition 12 (Séquence minimale) *Soit A un LTS et C_1 un ensemble de configurations. Une séquence $seq \in Sequences(A, C_1)$ est minimale si $\forall seq' \in Sequences(A, C_1)$, $seq' \sqsubseteq seq \implies seq' = seq$.*

L'ensemble des séquences minimales pour un LTS A et un ensemble de configurations C_1 est défini par :

$$MinSeq(A, C_1) = min_{\sqsubseteq}\{w \mid w \in Sequences(A, C_1)\}$$

Nous pouvons noter que si E est fini, $MinSeq(A, C_1)$ est lui aussi fini.

Définition 13 (Coupe minimale) *Soit A un LTS, et C_1 un ensemble de configurations. Une coupe $cm \in Cuts(A, C_1)$ est minimale si $\forall cm' \in Cuts(A, C_1)$, $cm' \subseteq cm \implies cm' = cm$.*

L'ensemble des coupes minimales pour un LTS A et un ensemble de configurations C_1 est défini par :

$$MinCuts(A, C_1) = min_{\subseteq}\{w \mid w \in Cuts(A, C_1)\}$$

On pourra se référer à [Rau01] pour plus d'informations concernant les coupes minimales.

3.7 Exemple d'illustration

Nous définissons dans cette partie un exemple qui correspond aux définitions que nous venons de mettre en place en ce début de section. Il sera utilisé pour illustrer les différentes définitions et métriques que nous introduirons par la suite dans ce chapitre.

3.7.1 Système à événements discrets S_{ex}

Considérons le LTS $S_{ex} = \langle Q_{ex}, Q_{i_{ex}}, E_{ex}, \rightarrow_{ex} \rangle$ comme modèle de système où :

- $Q_{ex} = \{q_1, q_2, q_3, q_4, q_5, q_6, q_7, q_8\}$
- $Q_{i_{ex}} = \{q_1\}$
- $E_{ex} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$
- $\rightarrow_{ex}$= $\{(q_1, e_1, q_2)$, (q_1, e_2, q_3), (q_1, e_3, q_4), (q_1, e_4, q_5), (q_1, e_5, q_5), (q_1, e_6, q_5), (q_1, e_7, q_6), (q_3, e_1, q_7), $(q_4, e_2, q_8)\}$

Nous considérons pour cet exemple que toutes les configurations sont stables, $Q_s = Q$.

S_{ex} est représenté en figure 3.5.

L'ensemble des variables du modèle S_{ex} est $V_{ex} = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$. Par soucis de simplicité, nous considérerons dans cet exemple que les variables sont booléennes.

Définissons maintenant les configurations comme valuations. Comme les variables sont booléennes, nous pouvons représenter les valuations par des ensembles de variables valuées à *vrai*, les variables n'apparaissant pas dans ces ensembles étant valuées à *faux*. Les valuations pour notre exemple sont décrites dans la table 3.1.

La figure 3.6 représente le LTS S_{ex} où les configurations ont été remplacées par leur valuation.

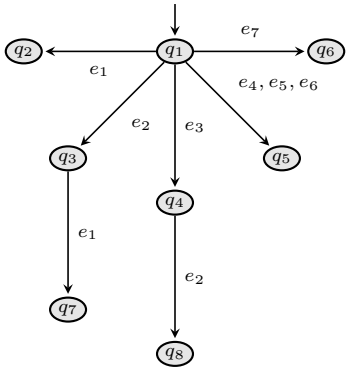


FIGURE 3.5 – Modèle S_{ex}

Q_{ex}	q_1	q_2	q_3	q_4	q_5	q_6	q_7	q_8
V_{ex}	$\{\}$	$\{v_1, v_5\}$	$\{v_2, v_6\}$	$\{v_2, v_3\}$	$\{v_3, v_4\}$	$\{v_7\}$	$\{v_1, v_2, v_6\}$	$\{v_2, v_3, v_5\}$

TABLE 3.1 – Valuations associées aux configurations de Q_{ex}

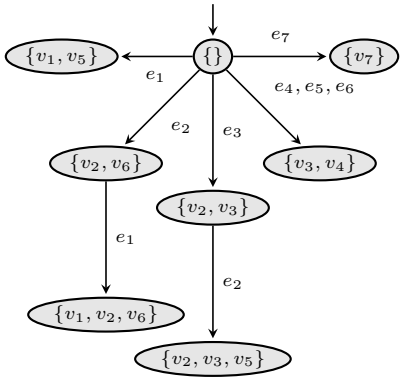


FIGURE 3.6 – S_{ex} avec valuations

3.7.2 Système sous diagnostic A_{ex}

Pour surveiller le système nous définissons maintenant un observateur O_{ex} tel que $V_{O_{ex}} = \{o_1, o_2, o_3, o_4\}$. La relation d'observation $\mathcal{R}_{O_{ex}}$, qui permet d'associer l'observateur O_{ex} au système S_{ex}, est décrite par l'ensemble des contraintes suivantes :

$$\mathcal{R}_{O_{ex}} = \begin{cases} o_1 = v_1 \\ o_2 = v_2 \\ o_3 = v_3 \\ o_4 = v_4 \end{cases}$$

Nous obtenons alors le modèle du système sous diagnostic $A_{ex} = \langle C_{ex}, I_{ex}, E_{ex}, T_{ex} \rangle$ où :

- $C_{ex} = \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\}$
- $I_{ex} = \{c_1\}$
- $E_{ex} = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$

- $T_{ex} = \{(c_1, e_1, c_2), (c_1, e_2, c_3), (c_1, e_3, c_4), (c_1, e_4, c_5), (c_1, e_5, c_5), (c_1, e_6, c_5), (c_1, e_7, c_6), (c_3, e_1, c_7), (c_4, e_2, c_8)\}$

L'ensemble des variables du modèle A_{ex} est $V_{ex} \uplus V_{O_{ex}}$. Nous pouvons donner l'ensemble des valuations correspondant aux configurations de C_{ex} avec la table 3.2.

C_{ex}	$V_{ex} \uplus V_{O_{ex}}$
c_1	$\{\}$
c_2	$\{v_1, v_5, o_1\}$
c_3	$\{v_2, v_6, o_2\}$
c_4	$\{v_2, v_3, o_2, o_3\}$
c_5	$\{v_3, v_4, o_3\}$
c_6	$\{v_7\}$
c_7	$\{v_1, v_2, v_6, o_1, o_2\}$
c_8	$\{v_2, v_3, v_5, o_2, o_3\}$

TABLE 3.2 – Valuations associées aux configurations de C_{ex}

On peut alors illustrer le système sous diagnostic par le LTS de la figure 3.7. Il est aussi intéressant de représenter le système sous diagnostic en ne conservant que les variables d'observation. Cela représente ce qui va pouvoir être observé par le diagnostiqueur et ce qu'il aura à disposition pour effectuer des diagnostics. Le LTS réduit aux variables d'observation est représenté sur la figure 3.8.

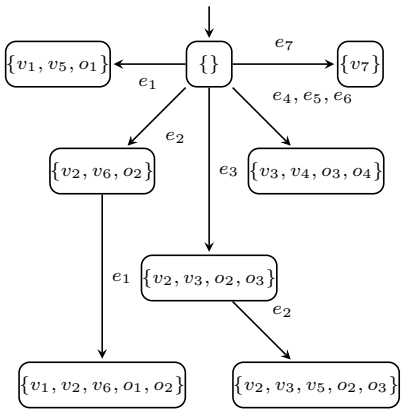


FIGURE 3.7 – Système sous diagnostic A_{ex}

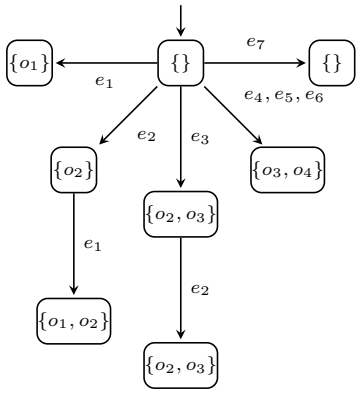


FIGURE 3.8 – A_{ex} avec $C_{ex}[V_{O_{ex}}]$

Considérons que tout événement du système sous diagnostic $A_{ex} = \langle C_{ex}, I_{ex}, E_{ex}, T_{ex} \rangle$ est bornant en profondeur, i.e. $E_{b_{ex}} = E_{ex}$. L'ensemble des groupes de configurations par leur profondeur est donc $\{C^0_{ex}, C^1_{ex}, C^2_{ex}\}$ où $C^0_{ex} = I_{ex}$, $C^1_{ex} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$ et $C^2_{ex} = C_{ex}$.

3.7.3 Décomposition d'un SUD

Avec notre exemple, nous venons d'illustrer la construction d'un système sous diagnostic. Illustrons maintenant la décomposition d'un système sous diagnostic.

Considérons notre système sous diagnostic exemple $A_{ex} = \langle C_{ex}, I_{ex}, E_{ex}, T_{ex} \rangle$ qui joue le rôle qu'avait α dans la section précédente (cf. 3.3).

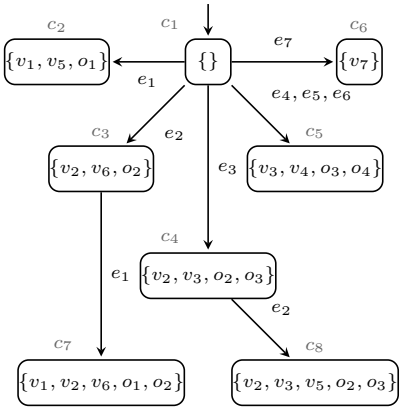


FIGURE 3.9 – Système A_{ex}

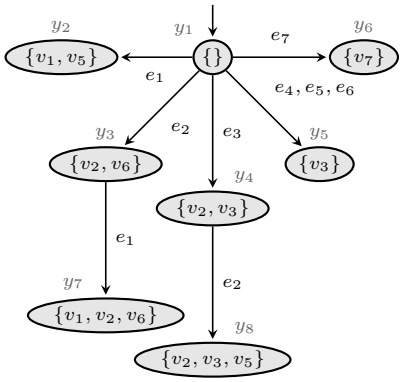


FIGURE 3.10 – $\beta_{ex} = proj(A_{ex}, V^{\beta_{ex}})$

Soit $V^{\beta_{ex}} = \{v_1, v_2, v_3, v_5, v_6, v_7\}$. En figures 3.9 et 3.10 sont illustrés respectivement le système A_{ex} à décomposer et la projection de ce système sur $V^{\beta_{ex}}$.

On construit le système projeté $proj(A_{ex}, V^{\beta_{ex}})$ à partir de A_{ex} et $V^{\beta_{ex}}$, et on obtient $\beta_{ex} = \langle Y_{ex}, Y_{i_{ex}}, E_{ex}, \rightarrow_{\beta_{ex}} \rangle$ tel que :

- $Y_{ex} = C_{ex}[V^{\beta_{ex}}] = \{y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8\}$.
- $Y_{i_{ex}} = I_{ex}[V^{\beta_{ex}}] = \{y_1\}$
- E_{ex} est le même ensemble d'événements.
- $\rightarrow_{\beta_{ex}} = \{t \in Y_{ex} \times E_{ex} \times Y_{ex} \mid \exists (x, e, x') \in T_{ex}, t = (x[V^{\beta_{ex}}], e, x'[V^{\beta_{ex}}])\}$
  $= \{(y_1, e_1, y_2), (y_1, e_2, y_3), (y_1, e_3, y_4), (y_1, e_4, y_5), (y_1, e_5, y_5), (y_1, e_6, y_5), (y_1, e_7, y_6), (y_3, e_1, y_7), (y_4, e_2, y_8)\}$.

Une fois le système projeté obtenu, nous cherchons à construire le système sous diagnostic $\langle \beta_{ex}, O_{r_{ex}}, \mathcal{R}_{C_{ex}} \rangle$ où l'observateur $O_{r_{ex}}$ est défini par l'ensemble de variables $V_{r_{ex}} = (V \uplus V_{O_{ex}}) \setminus V^{\beta_{ex}}$ et la relation $\mathcal{R}_{C_{ex}}$ est telle que $[\![\mathcal{R}_{C_{ex}}]\!] = C_{ex}$. On obtient donc :

- $V_{r_{ex}} = \{v_4, o_1, o_2, o_3, o_4\}$
- $[\![\mathcal{R}_{C_{ex}}]\!] = C_{ex}$ (cf. table 3.2)

Le nouveau système sous diagnostic $\alpha_{ex} = \langle \beta_{ex}, O_{r_{ex}}, \mathcal{R}_{C_{ex}} \rangle$ est construit et nous avons $\alpha_{ex} = \langle X_{ex}, X_{i_{ex}}, E_{ex}, \rightarrow_{\alpha_{ex}} \rangle$ tel que :

- $X_{ex} = \{x \in [\![\mathcal{R}_{C_{ex}}]\!] \mid x[V^{\beta_{ex}}] \in Y_{ex}\}$ est un ensemble de configurations.
- $X_{i_{ex}} = \{x \in X_{ex} \mid x[V^{\beta_{ex}}] \in Y_{i_{ex}}\}$ est l'ensemble des configurations initiales.
- E_{ex} est le même ensemble d'événements.
- $\rightarrow_{\alpha_{ex}} = \{(x, e, x') \in X_{ex} \times E_{ex} \times X_{ex} \mid x[V^{\beta_{ex}}] \xrightarrow{e}_{\beta_{ex}} x'[V^{\beta_{ex}}]\}$ est un ensemble de transitions.

Pour l'ensemble des configurations X_{ex} de α_{ex}, on a :

$$\begin{aligned} X_{ex} &= \{x \in [\![\mathcal{R}_{C_{ex}}]\!] \mid x[V^{\beta_{ex}}] \in Y_{ex}\} \\ &= \{c_1, c_2, c_3, c_4, c_5, c_6, c_7, c_8\} \\ &= C_{ex} \end{aligned}$$

Pour l'ensemble des configurations initiales $X_{i_{ex}}$ de α_{ex}, on a :

$$\begin{aligned} X_{i_{ex}} &= \{x \in X_{ex} \mid x[V^{\beta_{ex}}] \in Y_{i_{ex}}\} \\ &= \{c_1\} \\ &= I_{ex} \end{aligned}$$

Pour l'ensemble des événements de α_{ex}, on a E_{ex}.

Pour l'ensemble des transitions $\rightarrow_{\alpha_{ex}}$ de α_{ex}, on a :

$$\begin{aligned}\rightarrow_{\alpha_{ex}} &= \{(x, e, x') \in X_{ex} \times E_{ex} \times X_{ex} \mid x[V^{\beta_{ex}}] \xrightarrow{e}_{\beta_{ex}} x'[V^{\beta_{ex}}]\} \\ &= \{(c_1, e_1, c_2), (c_1, e_2, c_3), (c_1, e_3, c_4), (c_1, e_4, c_5), (c_1, e_5, c_5), \\ &\quad (c_1, e_6, c_5), (c_1, e_7, c_6), (c_3, e_1, c_7), (c_4, e_2, c_8)\} \\ &= T_{ex}\end{aligned}$$

On obtient donc $\alpha_{ex} = A_{ex}$ et on retrouve les relations décrites sur la figure 3.4.

3.7.4 Retrait de variables d'observation d'un SUD

Donnons l'exemple de systèmes sous diagnostic obtenus à partir de A_{ex} par retrait de variables d'observation.

Considérons $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \gamma_4\}$ un ensemble de systèmes obtenus par suppression de variables d'observation depuis A_{ex}. Les systèmes de Γ sont tels que pour $j \in [\![1 \dots 4]\!]$, $\gamma_j = A_{ex} \setminus \{o_j\}$ et ainsi $\gamma_j = \langle Z_{ex_j}, Z_{i_{ex_j}}, E_{ex}, \rightarrow_{\gamma_j} \rangle$.

Les LTS des systèmes de Γ sont représentés en figures 3.11, 3.12, 3.13 et 3.14.

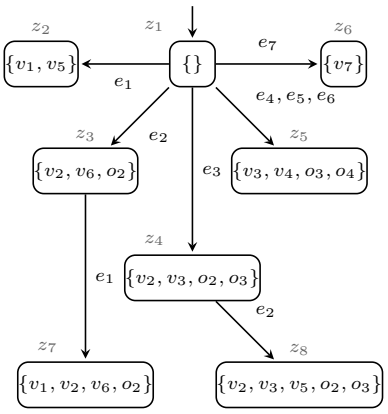


FIGURE 3.11 – Système γ_1

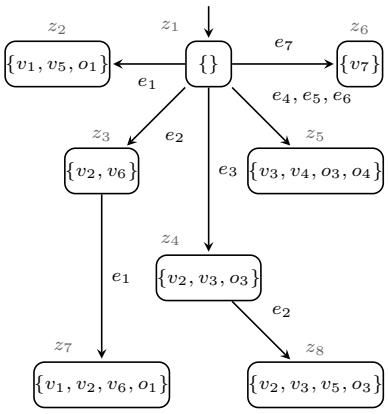


FIGURE 3.12 – Système γ_2

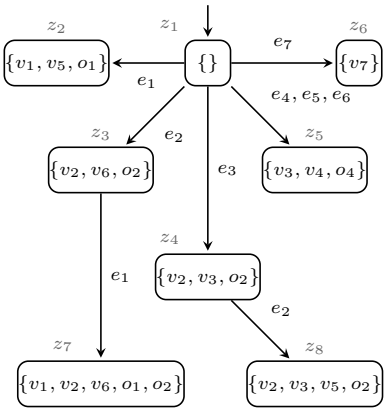


FIGURE 3.13 – Système γ_3

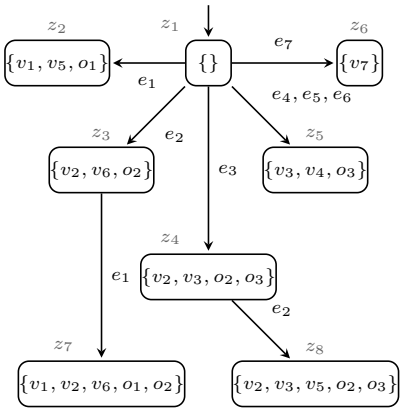


FIGURE 3.14 – Système γ_4

Nous obtenons donc pour $j \in [\![1 \dots 4]\!]$, $\gamma_j = \langle Z_{ex_j}, Z_{i_{ex_j}}, E_{ex}, \rightarrow_{\gamma_j} \rangle$ où :

- $Z_{ex_j} = \{z_1, z_2, z_3, z_4, z_5, z_6, z_7, z_8\}$
- $Z_{i_{ex_j}} = \{z_1\}$
- Le même E_{ex}.
- $\rightarrow_{\gamma_j}$= $\{(z_1, e_1, z_2)$, (z_1, e_2, z_3), (z_1, e_3, z_4), (z_1, e_4, z_5), (z_1, e_5, z_5), (z_1, e_6, z_5), (z_1, e_7, z_6), (z_3, e_1, z_7), $(z_4, e_2, z_8)\}$

3.7.5 Séquences et coupes de défaillances

Nous pouvons finalement illustrer les différents résultats de diagnostic présentés en 3.6.

Soit Obs_{ex} l'ensemble des observations pour lesquelles nous calculons les séquences, séquences minimales, coupes et coupes minimales. Considérons que pour cet exemple $Obs_{ex} = C_{ex}[V_O]$. Nous considérons donc que toutes les observations qui nous intéressent correspondent à la partie observable des configurations, i.e. à des valuations totales des variables observables du système sous diagnostic A_{ex}.

Les valuations étant totales, nous représenterons une observation, tout comme nous l'avons fait pour les configurations, par l'ensemble des variables, observables ici, évaluées à vrai, les autres variables observables étant donc évaluées à faux.

Nous proposons en illustration deux tables. Ces tables représentent, pour chacune des observations de Obs_{ex}, l'ensemble des séquences et séquences minimales d'une part (table 3.3), et l'ensemble des coupes et coupes minimales d'autre part (table 3.4).

obs	$Sequences(A_{ex}, C_{obs})$	$MinSeq(A_{ex}, C_{obs})$
$\{\}$	$\{\epsilon, e_7\}$	$\{\epsilon\}$
$\{o_1\}$	$\{e_1\}$	$\{e_1\}$
$\{o_2\}$	$\{e_2\}$	$\{e_2\}$
$\{o_1, o_2\}$	$\{e_2e_1\}$	$\{e_2e_1\}$
$\{o_2, o_3\}$	$\{e_3, e_3e_2\}$	$\{e_3\}$
$\{o_3, o_4\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$

TABLE 3.3 – Séquences et Séquences minimales de A_{ex} pour Obs_{ex}

obs	$Cuts(A_{ex}, C_{obs})$	$MinCuts(A_{ex}, C_{obs})$
$\{\}$	$\{\{\}, \{e_7\}\}$	$\{\{\}\}$
$\{o_1\}$	$\{\{e_1\}\}$	$\{\{e_1\}\}$
$\{o_2\}$	$\{\{e_2\}\}$	$\{\{e_2\}\}$
$\{o_1, o_2\}$	$\{\{e_1, e_2\}\}$	$\{\{e_1, e_2\}\}$
$\{o_2, o_3\}$	$\{\{e_3\}, \{e_2, e_3\}\}$	$\{\{e_3\}\}$
$\{o_3, o_4\}$	$\{\{e_4\}, \{e_5\}, \{e_6\}\}$	$\{\{e_4\}, \{e_5\}, \{e_6\}\}$

TABLE 3.4 – Coupes et Coupes minimales de A_{ex} pour Obs_{ex}

Si les observations considérées dans ces exemples sont exprimées comme des valuations totales de variables de V_O, cela n'est évidemment pas obligatoire. Les observations sont en effet définies comme des formules booléennes.

Par exemple, nous pouvons considérer l'observation $obs_1 = (o_3 = vrai \wedge o_4 = faux \vee o_1 = vrai)$. L'observation ne correspond pas à une valuation totale des variables de V_O. En effet, pour obs_1, rien n'est dit sur l'état de la variable d'observation o_4. Nous avons comme ensemble des configurations cibles $C_{obs_1} = \{c_2, c_4, c_7, c_8\}$ contenant l'ensemble des configurations du système sous diagnostic A_{ex} qui vérifient l'observation obs_1. Nous obtenons alors les résultats suivants :

- $Sequences(A_{ex}, C_{obs}) = \{e_1, e_2e_1, e_3, e_3e_2\}$.
- $MinSeq(A_{ex}, C_{obs}) = \{e_1, e_2e_1, e_3\}$.
- $Cuts(A_{ex}, C_{obs}) = \{\{e_1\}, \{e_1, e_2\}, \{e_3\}, \{e_2, e_3\}\}$.
- $MinCuts(A_{ex}, C_{obs}) = \{\{e_1\}, \{e_3\}\}$.

Chapitre 4

Performances de diagnostic

Ce chapitre permet de caractériser la disposition d'un système à fournir les informations nécessaires pour effectuer un bon diagnostic. L'approche choisie pour représenter le système est le raisonnement à base de modèle, et plus particulièrement un modèle à événements discrets, que nous avons décrit dans le chapitre précédent.

Le problème posé et le modèle défini, nous nous intéresserons maintenant aux définitions et aux métriques qui peuvent caractériser la bonne disposition du système à fournir les informations nécessaires pour effectuer un bon diagnostic, en regardant, dans les deux sections de chapitre, les notions de détectabilité et de diagnosticabilité d'événements. Ces deux notions caractérisent les performances de diagnostic d'un système. Ensuite, nous regarderons comment le contexte avionique dans lequel se déroule le travail peut permettre de particulariser et raffiner les définitions et métriques préalablement introduites. Finalement, nous nous intéresserons dans la dernière section de ce chapitre aux différentes relations de causes à effets qui peuvent être calculées et comment elles peuvent être utilisées pour caractériser les performances d'un système sous diagnostic.

4.1 Détectabilité d'événements

Dans la première section, nous avons posé le problème et défini ses composantes et les notions associées, en particulier l'objet étudié, le système sous diagnostic. Dans cette section, nous caractériserons les performances de détectabilité que le système et son monitoring permettent d'atteindre.

La détectabilité d'événements est une notion centrale pour définir la qualité d'un système pour le diagnostic. Comme nous l'avons vu dans la section 1.3, la détection des événements est la première étape de la chaîne de diagnostic. Elle est donc une notion clé pour le diagnostic qu'il est nécessaire d'évaluer.

Commençons par définir la notion de détectabilité et les notions dépendantes puis définissons dans un second temps des métriques basées sur ces définitions.

4.1.1 Définitions

Les définitions qui suivent sont définies pour un système sous diagnostic $A = \langle C, I, E, T \rangle$. Dans un soucis de généricité, elles sont exprimées relativement à un ensemble de configurations.

Donner des définitions relatives à un ensemble de configurations permet de prendre en compte l'utilisation pratique qui peut en être fait. En effet, selon la taille des systèmes et donc des modèles manipulés, et selon les ressources de calcul disponibles, il n'est pas toujours possible d'obtenir le LTS du modèle. Il peut donc être intéressant dans certains cas, pour rendre exploitable le modèle en pratique et pour pouvoir calculer les différentes fonctions de successeurs et d'accessibles que

nous avons définies dans la section 3.1, de restreindre l'ensemble des configurations sur lequel les calculs seront effectués.

Dans la pratique, on retrouve différents ensembles de configurations caractéristiques :

- I : l'ensemble des configurations initiales est l'ensemble de configurations typiquement considéré lorsque l'on s'intéresse à des systèmes qui ne sont pas simples. On regarde ainsi le comportement du système depuis un état initial. Ainsi, nous restreignons notre analyse aux simples pannes, i.e. au système dans lequel les séquences de pannes sont au maximum de longueur 1.
- C : l'ensemble de toutes les configurations du système est considéré le plus souvent sur des systèmes de taille raisonnable, pour lesquels il est possible de calculer le LTS correspondant.
- C_s : l'ensemble des configurations stables correspond à l'ensemble le plus représentatif du comportement que l'on peut observer de la part d'un système sous diagnostic.
- C^n : l'ensemble des configurations accessibles en utilisant au maximum n événements bornants est moins classique que les précédents, néanmoins, comme nous le soulignons lors de sa définition en fin de section 3.1, il a un réel intérêt dans des domaines comme le nôtre ou comme la sûreté de fonctionnement des systèmes avioniques par exemple, où l'on peut ne garder que les cas de pannes simple/double/triple (un/deux/trois événements de défaillance maximum expliquent une observation), considérant que les autres cas sont négligeables et non représentatifs.
- $C_s^n = C_s \cap C^n$: l'ensemble des stables qui sont accessibles en utilisant au plus n transitions étiquetées par des événements bornants combine la stabilité des configurations et le fait de borner en profondeur le système. Cet ensemble peut s'avérer le plus adapté au traitement de modèles dysfonctionnels (intégrant des défaillances) de grande taille.

Avant de définir la notion de détectabilité, nous devons introduire quelques définitions. Commençons par la définition de l'ensemble des événements déclenchables dans un ensemble de configurations. Pour cette définition comme pour l'ensemble des définitions et métriques, nous proposons son illustration avec l'exemple décrit en section 3.7.

Définition 14 (Ensemble des événements déclenchables) *Un événement e est déclenchable dans un ensemble de configurations C' s'il est non contrôlable et s'il existe un successeur stable pour e pour au moins une des configurations stables de C'.*

$$E^{C'} = \{e \in E_{nc} \mid succ_s(C' \cap C_s, e) \neq \emptyset\} \tag{4.1}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$E^{C'}$	E_{ex}	$\{e_1\}$	$\{e_2\}$	$\{e_1, e_2\}$	E_{ex}

∘

Tout comme nous l'avons fait avec la fonction *succ*, l'ensemble de configurations $\{c\}$ réduit à un singleton est simplifié par c.

La notion d'effets d'un événement n'est définie que pour les configurations dans lesquelles un événement est déclenchable. Il est important d'introduire cela car il faut pouvoir distinguer les événements qui ne produisent aucun effet dans une configuration, des événements qui ne sont pas déclenchables dans cette configuration. Nous n'accuserons pas un événement non déclenchable dans un ensemble de configurations C' de ne pas être détectable dans C'.

Définition 15 (Effets d'un événement) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. La fonction $effets : \mathcal{P}(C) \times E^{C'} \longrightarrow \mathcal{V}(V_O)$ définit l'ensemble des situations possibles après l'occurrence de e dans C'.*

$$effets(C', e) = succ_s(C' \cap C_s, e)[V_O] \tag{4.2}$$

Exemple:

C	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$effets(C, e_1)$	$\{\{o_1\}\}$	$\{\{o_1, o_2\}\}$	nd	$\{\{o_1, o_2\}\}$	$\{\{o_1\}, \{o_1, o_2\}\}$
$effets(C, e_2)$	$\{\{o_2\}\}$	nd	$\{\{o_2, o_3\}\}$	$\{\{o_2, o_3\}\}$	$\{\{o_2\}, \{o_2, o_3\}\}$
$effets(C, e_3)$	$\{\{o_2, o_3\}\}$	nd	nd	nd	$\{\{o_2, o_3\}\}$
$effets(C, e_4)$	$\{\{o_3, o_4\}\}$	nd	nd	nd	$\{\{o_3, o_4\}\}$
$effets(C, e_5)$	$\{\{o_3, o_4\}\}$	nd	nd	nd	$\{\{o_3, o_4\}\}$
$effets(C, e_6)$	$\{\{o_3, o_4\}\}$	nd	nd	nd	$\{\{o_3, o_4\}\}$
$effets(C, e_7)$	$\{\{\}\}$	nd	nd	nd	$\{\{\}\}$

où nd *signifie* non défini.

∘

Nous considérons à travers la définition précédente qu'un effet d'un événement correspond à la partie observable d'une configuration stable. Les configurations instables ne sont donc que des états de transition. Cela signifie qu'elles ne contiennent pas la cause des observations relevées dans les configurations stables leur succédant. En effet, les causes correspondant à des événements non contrôlables, le cas où un événement non contrôlable arrive pendant une phase de stabilisation n'est pas considéré comme cause car dans les systèmes réactifs, la réaction est rapide, et l'occurrence de deux événements non contrôlables dans ce laps de temps est hautement improbable.

Avec cette définition nous connaissons les effets possibles lors de l'occurrence d'un événement. Ces effets représentent la *signature* de l'événement. Partant de là, nous pouvons décider de la discriminabilité de deux événements.

Du fait de la possibilité pour un événement d'avoir plusieurs effets, nous intégrerons, pour certaines notions, deux types de caractérisation. Pour ces notions, nous aurons une version dite "faible" et une autre dite "forte". Cela découle du fait de considérer d'une part la propriété vérifiée quels que soient les effets des événements, on parle alors de notion "forte", et d'autre part la propriété vérifiée pour au moins un des effets des événements, et on parle alors de notion "faible". Intuitivement, cela correspondra souvent aux notions avec les deux versions des quantifieurs, universelle pour la notion "forte" où on s'assurera que la propriété est tout le temps vraie, et existentielle pour la notion "faible" où on s'assurera que la propriété est vraie pour au moins un cas.

Définition 16 (Discriminabilité forte de deux événements) *Soient C' un ensemble de configurations, e et e' deux événements déclenchables dans C'. e et e' sont* fortement discriminables *dans C' si $effets(C', e) \cap effets(C', e') = \emptyset$.*

Exemple:

e_1 et e_2 sont fortement discriminables dans c_1.
e_2 et e_3 sont fortement discriminables dans c_1.
e_2 et e_3 ne sont pas fortement discriminables dans C_{ex}.
e_4 et e_5 ne sont pas fortement discriminables dans C_{ex}.
La discriminabilité forte de e_4 et e_5 n'est pas définie dans $\{c_3, c_4\}$.

∘

Lorsque deux événements sont fortement discriminables, cela signifie qu'ils ne se retrouveront pas dans un même diagnostic puisqu'ils n'ont aucun effet en commun. On est donc sûr que quel que soit l'effet que l'on observe, les événements ne feront pas tous les deux partie de l'accusation. Ce caractère "sûr" explique la notion de "forte".

Nous introduisons maintenant la version "faible" de la discriminabilité.

Définition 17 (Discriminabilité faible de deux événements) *Soit C' un ensemble de configurations, soient e et e' deux événements déclenchables dans C'. e est* faiblement discriminable *de e' dans C' si $effets(C', e) \not\subseteq effets(C', e')$ et e et e' ne sont pas fortement discriminables. On dit que deux événements sont faiblement discriminables si chaque événement est faiblement discriminable de l'autre.*

Exemple:
e_1 *et* e_2 *ne sont pas faiblement discriminables dans* c_1.
e_2 *et* e_3 *ne sont pas faiblement discriminables dans* C_{ex}.
e_4 *et* e_5 *ne sont pas faiblement discriminables dans* C_{ex}.
La discriminabilité faible de e_4 *et* e_5 *n'est pas définie dans* $\{c_3, c_4\}$.

∘

Dire qu'un événement e est faiblement discriminable d'un deuxième événement e' signifie donc que e et e' partagent des effets mais que e est associé à au moins un effet que e' ne provoque pas. Ainsi nous sommes sûr qu'il existe un diagnostic dans lequel apparaîtra e mais pas e'. Même si e et e' ne sont pas toujours (fortement) discriminables, e est donc discriminable de e' dans au moins une situation. L'existence d'une telle situation justifie la caractérisation "faible" de la définition.

Par la suite, nous dirons qu'un événement e est non discriminable d'un événement e' dans un ensemble de configurations C' s'il n'est ni fortement, ni faiblement discriminable de e' dans C' (tous les effets de e sont des effets de e'). Deux événements sont donc non discriminables si chacun est non discriminable de l'autre (cela signifie qu'ils sont toujours présents tous les deux dans les diagnostics, i.e. qu'ils ont exactement les mêmes effets). Remarquons que s'il est toujours possible de dire si un événement est fortement, faiblement, ou non discriminable d'un autre événement, il n'est néanmoins pas possible d'en dire autant concernant les paires d'événements. En effet, si les deux événements d'une paire ne sont pas discriminable de la même façon l'un envers l'autre, la discriminabilité ne sera pas définie pour la paire en tant que telle.

Nous dirons que deux événements sont discriminables s'ils sont soit faiblement discriminables, soit fortement discriminables.

Nous proposons en table 4.1 la discriminabilité complète des événements de E_{ex} pour l'exemple. Notons que e_2 est faiblement discriminable de e_3 mais que e_3 n'est pas discriminable de e_2, dans

e	e_1	e_2	e_3	e_4	e_5	e_6	e_7
e_1 discriminable de e dans C_{ex}	ndis	F	F	F	F	F	F
e_2 discriminable de e dans C_{ex}	F	ndis	f	F	F	F	F
e_3 discriminable de e dans C_{ex}	F	ndis	ndis	F	F	F	F
e_4 discriminable de e dans C_{ex}	F	F	F	ndis	ndis	ndis	F
e_5 discriminable de e dans C_{ex}	F	F	F	ndis	ndis	ndis	F
e_6 discriminable de e dans C_{ex}	F	F	F	ndis	ndis	ndis	F
e_7 discriminable de e dans C_{ex}	F	F	F	F	F	F	ndis

où *ndis*, *f* et *F* signifient respectivement *non discriminable*, *faiblement discriminable* et *fortement discriminable.*

TABLE 4.1 – Discriminabilité des événements de E_{ex} dans C_{ex}

C_{ex}. En effet, $effets(C_{ex}, e_3) \subset effets(C_{ex}, e_2)$. La paire (e_2, e_3) n'est pas qualifiable en tant que telle pour la discriminabilité dans C_{ex}.

La notion de discriminabilité que nous venons d'introduire amène à celle de détectabilité. Définissons tout d'abord $Obs_{OK} \subseteq \mathcal{V}(V_O)$ comme l'ensemble des observations représentant un bon fonctionnement du système (fonctionnement dit *nominal*).

Ces observations nominales représentent les situations où le système va bien, et donc dans lesquelles aucun problème ne sera détecté par le système. C'est la raison pour laquelle la notion de détectabilité est définie vis-à-vis de ces observations spécifiques.

Définition 18 (Détectabilité forte d'un événement) *Soit* C' *un ensemble de configurations, soit* e *un événement déclenchable dans* C'. e *est* fortement détectable *dans* C' *si* $effets(C', e) \cap Obs_{OK} = \emptyset$.

Exemple:
Considérons $Obs_{OK_{ex}} = \{\{\}, \{o_1, o_2\}\}$.

e_1 *est fortement détectable dans* c_1.
e_1 *n'est pas fortement détectable dans* C_{ex}.
e_2 *est fortement détectable dans* C_{ex}.
La détectabilité forte de e_7 *n'est pas définie dans* $\{c_3, c_4\}$.

∘

La détectabilité forte d'un événement est définie par ce qui pourrait se rapprocher de la discriminabilité forte de cet événement avec les observations nominales du système. Le fait de pouvoir, d'un point de vue observable, être confondu avec l'une de ces configurations, implique qu'il y a des cas dans lesquels l'événement va arriver mais où on ne saura pas le remarquer. Le système ne sera pas dans un état d'alerte.

Définition 19 (Détectabilité faible d'un événement) *Soient* C' *un ensemble de configurations et* e *un événement déclenchable dans* C'. e *est* faiblement détectable *dans* C' *s'il n'est pas fortement détectable et que* $effets(C', e) \not\subseteq Obs_{OK}$.

Exemple:
e_1 *n'est pas faiblement détectable dans* c_1.
e_1 *est faiblement détectable dans* C_{ex}.
e_2 *n'est pas faiblement détectable dans* C_{ex}.
La détectabilité faible de e_7 *n'est pas définie dans* $\{c_3, c_4\}$.

∘

En vérifiant que e est faiblement détectable, on s'assure qu'il y a au moins un de ces effets qui n'est pas confondu avec un bon fonctionnement du système, et donc qu'il y aura au moins une situation dans laquelle l'occurrence de e sera détectée.

Pour résumer, la définition de détectabilité forte permet de s'assurer qu'un événement ne peut pas arriver sans que l'on n'observe une situation problématique dans le système. La définition de détectabilité faible, elle, exprime que l'événement est détectable dans certaines situations, et non détectable dans d'autres.

Nous présentons en table 4.2 la détectabilité des événements de E_{ex} pour l'exemple.

e	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
détectabilité de e_1 dans C'	F	ndét	*nd*	ndét	f
détectabilité de e_2 dans C'	F	*nd*	F	F	F
détectabilité de e_3 dans C'	F	*nd*	*nd*	*nd*	F
détectabilité de e_4 dans C'	F	*nd*	*nd*	*nd*	F
détectabilité de e_5 dans C'	F	*nd*	*nd*	*nd*	F
détectabilité de e_6 dans C'	F	*nd*	*nd*	*nd*	F
détectabilité de e_7 dans C'	ndét	*nd*	*nd*	*nd*	ndét

où *nd*, *ndét*, *f* et *F* signifient respectivement *non déclenchable* ($\Longleftrightarrow$ détectabilité non définie), *non détectable*, *faiblement détectable* et *fortement détectable*.

TABLE 4.2 – Détectabilité des événements de E_{ex}

À partir de là nous pouvons rassembler les événements selon leur détectabilité.

Définition 20 (Ensemble des événements fortement (resp. faiblement) détectables) *Soit* C' *un ensemble de configurations. L'*ensemble des événements fortement (resp. faiblement) détectables *dans* C' *est noté* $E_{FD}^{C'}$ *(resp.* $E_{fD}^{C'}$*).*

$$E_{FD}^{C'} = \{e \in E^{C'} \mid e \text{ est fortement détectable dans } C'\} \tag{4.3}$$

$$E_{fD}^{C'} = \{e \in E^{C'} \mid e \text{ est faiblement détectable dans } C'\} \tag{4.4}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$E_{FD}^{C'}$	$\{e_1, e_2, e_3, e_4, e_5, e_6\}$	$\emptyset$	$\{e_2\}$	$\{e_2\}$	$\{e_2, e_3, e_4, e_5, e_6\}$
$E_{fD}^{C'}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_1\}$

∘

L'ensemble des événements détectables dans C' correspond à l'ensemble des événements qui y sont soit fortement soit faiblement détectables. Ainsi $E_D^{C'} = E_{FD}^{C'} \cup E_{fD}^{C'}$.

Une autre notion, souvent associée à la notion de détectabilité, exprime le fait qu'à chaque fois qu'un événement arrive, la situation du système change. Nous décrivons cette caractéristique en parlant d'événement perceptible.

Pour les définitions à venir, introduisons la notation $C'^{\xrightarrow{e}} = \{c \in C' \mid e \in E^c\}$ représentant, pour un événement e, l'ensemble des configurations de C' dans lesquelles e est déclenchable.

Exemple:

e	e_1	e_2	e_3	e_4	e_5	e_6	e_7
$C_{ex}^{\xrightarrow{e}}$	$\{c_1, c_3\}$	$\{c_1, c_4\}$	$\{c_1\}$	$\{c_1\}$	$\{c_1\}$	$\{c_1\}$	$\{c_1\}$

∘

Définition 21 (Événement fortement perceptible) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. e est* fortement perceptible *dans C' si pour tout c de $C'^{\xrightarrow{e}}$, $c[V_O] \notin effets(c, e)$.*

Exemple:
e_1 est fortement perceptible dans C_{ex}.
e_2 n'est pas fortement perceptible dans C_{ex}.

∘

Remarque *Un événement détectable n'est pas forcément perceptible, et inversement. En effet, un événement dont l'occurrence conserve une observation non nominale du système est détectable mais non perceptible. Un événement dont l'occurrence change l'état du système et l'amène dans une configuration dont la partie observable est dans Obs_{OK} est perceptible mais non détectable (intuitivement, c'est le cas des événements de type* reset*).*

Caractériser les événements perceptibles est intéressant par exemple dans le cas de système surveillés de manière continue. On s'assure ainsi par le fait qu'un événement est fortement perceptible que les observations relevées après l'occurrence de l'événement seront différentes de celles relevées avant son occurrence. Déceler ces différences de manière continue est une méthode utilisée pour faire de la détection d'événements (cf. 2.3.1.6).

Définition 22 (Événement faiblement perceptible) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. e est* faiblement perceptible *dans C' s'il n'est pas fortement perceptible mais que pour au moins un c de $C'^{\xrightarrow{e}}$, $c[V_O] \notin effets(c, e)$.*

Exemple:
e_1 n'est pas faiblement perceptible dans C_{ex}.
e_2 est faiblement perceptible dans C_{ex}.

∘

Le fait qu'un événement soit faiblement perceptible assure qu'il y a au moins un cas dans lequel l'événement change l'observation globale du système au moment où il arrive.

Comme nous l'avions fait pour les définitions concernant les événements détectables (déf. 18 et 19), nous présentons en table 4.3 la perceptibilité des événements de E_{ex} pour l'exemple, puis nous introduisons les ensembles des événements perceptibles.

e	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
perceptibilité de e_1 dans C'	F	F	*nd*	F	F
perceptibilité de e_2 dans C'	F	*nd*	np	np	f
perceptibilité de e_3 dans C'	F	*nd*	*nd*	*nd*	F
perceptibilité de e_4 dans C'	F	*nd*	*nd*	*nd*	F
perceptibilité de e_5 dans C'	F	*nd*	*nd*	*nd*	F
perceptibilité de e_6 dans C'	F	*nd*	*nd*	*nd*	F
perceptibilité de e_7 dans C'	np	*nd*	*nd*	*nd*	np

où *nd*, *np*, *f* et *F* signifient respectivement *non déclenchable* ($\Longleftrightarrow$ perceptibilité non définie), *non perceptible*, *faiblement perceptible* et *fortement perceptible*.

TABLE 4.3 – Perceptibilité des événements de E_{ex}

Définition 23 (Ensemble des événements fortement (resp. faiblement) perceptibles) *Soit C' un ensemble de configurations. L'*ensemble des événements fortement (resp. faiblement) perceptibles *dans C' est noté $E^{C'}_{FP}$ (resp. $E^{C'}_{fP}$).*

$$E^{C'}_{FP} = \{e \in E^{C'} \mid e \text{ est fortement perceptible dans } C'\} \tag{4.5}$$

$$E^{C'}_{fP} = \{e \in E^{C'} \mid e \text{ est faiblement perceptible dans } C'\} \tag{4.6}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$E^{C'}_{FP}$	$\{e_1, e_2, e_3, e_4, e_5, e_6\}$	$\{e_1\}$	$\emptyset$	$\{e_1\}$	$\{e_1, e_3, e_4, e_5, e_6\}$
$E^{C'}_{fP}$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\{e_2\}$

$\circ$

Tout comme pour les événements détectables, l'ensemble des événements perceptibles dans C' est obtenu par l'union des deux ensembles de perceptibilité, forte et faible. $E^{C'}_{P} = E^{C'}_{FP} \cup E^{C'}_{fP}$.

4.1.2 Métriques

Dans cette partie nous introduisons différentes métriques qui peuvent être appliquées pour caractériser la détectabilité des événements d'un système sous diagnostic. Ces métriques s'appuient sur les définitions de la partie précédente et sont représentées par des ratios d'événements.

Nous retrouvons, comme lors des définitions, les deux caractérisations "faible" et "forte" pour ces métriques.

Métrique 1 (Ratio de détectabilité forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio de détectabilité forte (resp. faible) *$RDF_{C'}$ (resp. $RDf_{C'}$) , souvent appelé aussi* couverture*, donne la proportion d'événements fortement (resp. faiblement) détectables dans C', où on ne considère que les événements qui y sont déclenchables.*

$$RDF_{C'} = \frac{|E^{C'}_{FD}|}{|E^{C'}|} \tag{4.7}$$

$$RDf_{C'} = \frac{|E^{C'}_{fD}|}{|E^{C'}|} \tag{4.8}$$

Exemple:

$RDF_{c_1} = \frac{|E^{c_1}_{FD}|}{|E^{c_1}|} = \frac{6}{7} \simeq 86\%$.
$RDF_{c_3} = \frac{|E^{c_3}_{FD}|}{|E^{c_3}|} = \frac{0}{1} = 0\%$.
$RDF_{C_{ex}} = \frac{|E^{C_{ex}}_{FD}|}{|E^{C_{ex}}|} = \frac{5}{7} \simeq 71\%$.

$RDf_{c_1} = \frac{|E^{c_1}_{fD}|}{|E^{c_1}|} = \frac{0}{7} = 0\%$.
$RDf_{c_3} = \frac{|E^{c_3}_{fD}|}{|E^{c_3}|} = \frac{0}{1} = 0\%$.
$RDf_{C_{ex}} = \frac{|E^{C_{ex}}_{fD}|}{|E^{C_{ex}}|} = \frac{1}{7} \simeq 14\%$.

∘

Nous pouvons donc déduire le ratio de détectabilité dans C' : $RD_{C'} = RDF_{C'} + RDf_{C'}$. Nous noterons que ce ratio donne toujours un résultat compris entre 0 et 1 car les deux ensembles d'événements $RDF_{C'}$ et $RDf_{C'}$ sont disjoints.

Exemple:

$RD_{c_1} = RDF_{c_1} + RDf_{c_1} = \frac{6}{7} \simeq 86\%$.
$RD_{c_3} = RDF_{c_3} + RDf_{c_3} = \frac{0}{1} = 0\%$.
$RD_{C_{ex}} = RDF_{C_{ex}} + RDf_{C_{ex}} = \frac{6}{7} \simeq 86\%$.

∘

De la même façon nous pouvons définir les deux métriques de *ratio de perceptibilité*.

Métrique 2 (Ratio de perceptibilité forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio de perceptibilité forte (resp. faible) *$RPF_{C'}$ (resp. $RPf_{C'}$) donne la proportion d'événements fortement (resp. faiblement) perceptibles dans C', où on ne considère que les événements qui y sont déclenchables.*

$$RPF_{C'} = \frac{|E^{C'}_{FP}|}{|E^{C'}|} \tag{4.9}$$

$$RPf_{C'} = \frac{|E^{C'}_{fP}|}{|E^{C'}|} \tag{4.10}$$

Exemple:

$RPF_{c_1} = \frac{|E^{c_1}_{FP}|}{|E^{c_1}|} = \frac{6}{7} \simeq 86\%$.
$RPF_{c_3} = \frac{|E^{c_3}_{FP}|}{|E^{c_3}|} = \frac{1}{1} = 100\%$.
$RPF_{C_{ex}} = \frac{|E^{C_{ex}}_{FP}|}{|E^{C_{ex}}|} = \frac{5}{7} \simeq 71\%$.

$RPf_{c_1} = \frac{|E^{c_1}_{fP}|}{|E^{c_1}|} = \frac{0}{7} = 0\%$.
$RPf_{c_3} = \frac{|E^{c_3}_{fP}|}{|E^{c_3}|} = \frac{0}{1} = 0\%$.
$RPf_{C_{ex}} = \frac{|E^{C_{ex}}_{fP}|}{|E^{C_{ex}}|} = \frac{1}{7} \simeq 14\%$.

∘

En outre, il est intéressant de définir des métriques qui regardent la proportion d'événements qui sont détectables et perceptibles.

Métrique 3 (Ratio de détectabilité et perceptibilité forte) *Soit C' un ensemble de configurations. Le* ratio de détectabilité et perceptibilité forte *$RDPF_{C'}$ donne la proportion d'événements fortement détectables et perceptibles dans C', où on ne considère que les événements qui y sont déclenchables.*

$$RDPF_{C'} = \frac{|E^{C'}_{FD} \cap E^{C'}_{FP}|}{|E^{C'}|} \tag{4.11}$$

Exemple:

$RDPF_{c_1} = \frac{|E^{c_1}_{FD} \cap E^{c_1}_{FP}|}{|E^{c_1}|} = \frac{6}{7} \simeq 84\%$.
$RDPF_{c_3} = \frac{|E^{c_3}_{FD} \cap E^{c_3}_{FP}|}{|E^{c_3}|} = \frac{0}{1} = 0\%$.
$RDPF_{C_{ex}} = \frac{|E^{C_{ex}}_{FD} \cap E^{C_{ex}}_{FP}|}{|E^{C_{ex}}|} = \frac{4}{7} \simeq 57\%$.

∘

Nous pouvons aussi combiner ces deux propriétés pour considérer la notion de faible.

Métrique 4 (Ratio de détectabilité et perceptibilité faible) *Soit C' un ensemble de configurations. Le* ratio de détectabilité et perceptibilité faible *$RDPF_{C'}$ donne la proportion d'événements détectables et perceptibles dans C', mais non fortement, où on ne considère que les événements qui y sont déclenchables.*

$$RDPf_{C'} = \frac{|(E^{C'}_{fD} \cap E^{C'}_{fP}) \cup (E^{C'}_{fD} \cap E^{C'}_{FP}) \cup (E^{C'}_{FD} \cap E^{C'}_{fP})|}{|E^{C'}|} \tag{4.12}$$

Exemple:

$RDPf_{c_1} = \frac{0}{7} = 0\%$.
$RDPf_{c_3} = \frac{0}{7} = 0\%$.
$RDPf_{C_{ex}} = \frac{|\{e_1\} \cap \{e_2\} \cup \{e_1\} \cap \{e_1,e_3,e_4,e_5,e_6\} \cup \{e_2,e_3,e_4,e_5,e_6\} \cap \{e_2\}|}{|E^{C_{ex}}|} = \frac{2}{7} \simeq 29\%$.

∘

Dans un système, tous les événements n'ont pas forcément la même importance. Il est intéressant et pertinent de calculer des ratios de détectabilité et de perceptibilité qui prennent en compte cette notion d'importance d'événement. Le degré d'importance des pannes peut être exprimé à travers une fonction de pondération w. Cette dernière peut représenter différentes caractéristiques comme par exemple la probabilité d'occurrence de l'événement, sa criticité ou encore son impact sur le système.

Métrique 5 (Ratio pondéré de détectabilité forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio pondéré de détectabilité forte (resp. faible) *$RPDF_{C'}$ (resp. $RPDf_{C'}$), donne la proportion d'événements fortement (resp. faiblement) détectables dans C', où on ne considère que les événements qui y sont déclenchables, et où les événements sont pondérés par une fonction w.*

$$RPDF_{C'} = \frac{\sum_{e \in E^{C'}_{FD}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.13}$$

$$RPDf_{C'} = \frac{\sum_{e \in E^{C'}_{fD}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.14}$$

Exemple:

Considérons $w_{ex} : E_{ex} \longrightarrow \mathbb{N}$, *une fonction de pondération associant à un événement une valeur d'importance :*

e	e_1	e_2	e_3	e_4	e_5	e_6	e_7
$w_{ex}()$	40	10	10	10	10	10	1

On exprime à travers la fonction w *que l'événement* e_7 *est peu important comparés aux autres, et que parmi les autres,* e_1 *est plus important. Pour plus de simplicité dans la notation, on considérera que* $w(E') = \sum_{e \in E'} w(e)$.

$RPDF_{c_1} = \frac{w(\{e_1,e_2,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{40+5*10}{40+5*10+1} = \frac{90}{91} \simeq 99\%$
$\rightarrow$ e_7 *ayant un poids relativement faible, ne pas la détecter n'impacte quasiment pas le résultat de la métrique.*
$RPDF_{c_3} = \frac{w(\emptyset)}{w(\{e_1\})} = \frac{0}{40} = 0\%$
$RPDF_{C_{ex}} = \frac{w(\{e_2,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{5*10}{40+5*10+1} = \frac{50}{91} \simeq 55\%$

$RPDf_{c_1} = \frac{0}{91} = 0\%$
$RPDf_{c_3} = \frac{0}{40} = 0\%$
$RPDf_{C_{ex}} = \frac{w(\{e_1\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{40}{91} \simeq 44\%$

∘

Nous pouvons définir de la même manière les versions pondérées des autres métriques précédentes.

Métrique 6 (Ratio pondéré de perceptibilité forte (resp. faible)) *Soit* C' *un ensemble de configurations. Le* ratio pondéré de perceptibilité forte (resp. faible) $RPPF_{C'}$ *(resp.* $RPPf_{C'}$*), donne la proportion d'événements fortement (resp. faiblement) perceptibles dans* C'*, où on ne considère que les événements qui y sont déclenchables, et où les événements sont pondérés par une fonction* w.

$$RPPF_{C'} = \frac{\sum_{e \in E_{FP}^{C'}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.15}$$

$$RPPf_{C'} = \frac{\sum_{e \in E_{fP}^{C'}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.16}$$

Exemple:

$RPPF_{c_1} = \frac{w(\{e_1,e_2,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{40+5*10}{40+5*10+1} = \frac{90}{91} \simeq 99\%$
$RPPF_{c_3} = \frac{w(\{e_1\})}{w(\{e_1\})} = 100\%$
$RPPF_{C_{ex}} = \frac{w(\{e_1,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{40+4*10}{40+5*10+1} = \frac{80}{91} \simeq 88\%$

$RPPf_{c_1} = \frac{0}{91} = 0\%$
$RPPf_{c_3} = \frac{0}{40} = 0\%$
$RPPf_{C_{ex}} = \frac{w(\{e_2\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{10}{91} \simeq 11\%$

∘

Métrique 7 (Ratio pondéré de détectabilité et perceptibilité forte) *Soit* C' *un ensemble de configurations. Le* ratio pondéré de détectabilité et de perceptibilité forte $RPDPF_{C'}$*, donne la proportion d'événements fortement détectables et perceptibles dans* C'*, où on ne considère que les événements qui y sont déclenchables, et où les événements sont pondérés par une fonction* w.

$$RPDPF_{C'} = \frac{\sum_{e \in E_{FD}^{C'} \cap E_{FP}^{C'}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.17}$$

Exemple:

$RPDPF_{c_1} = \frac{w(\{e_1,e_2,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{40+5*10}{40+5*10+1} = \frac{90}{91} \simeq 99\%$
$RPDPF_{c_3} = \frac{w(\emptyset \cap \{e_1\})}{w(\{e_1\})} = 0\%$
$RPDPF_{C_{ex}} = \frac{w(\{e_1,e_3,e_4,e_5,e_6\} \cap \{e_2,e_3,e_4,e_5,e_6\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{4*10}{40+5*10+1} = \frac{40}{91} \simeq 44\%$

∘

Définissons maintenant la version pondéré de la métrique 4.

Métrique 8 (Ratio pondéré de détectabilité et perceptibilité faible) *Soit C' un ensemble de configurations. Le* ratio pondéré de détectabilité et de perceptibilité faible *$RPDPF_{C'}$, donne la proportion d'événements détectables et perceptibles dans C', mais non fortement, où on ne considère que les événements qui y sont déclenchables, et où les événements sont pondérés par une fonction w.*

$$RPDPf_{C'} = \frac{\sum_{e \in (E_{fD}^{C'} \cap E_{fP}^{C'}) \cup (E_{fD}^{C'} \cap E_{FP}^{C'}) \cup (E_{FD}^{C'} \cap E_{fP}^{C'})} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.18}$$

Exemple:

$RPDPf_{c_1} = \frac{0}{91} = 0\%$
$RPDPf_{c_3} = \frac{0}{40} = 0\%$
$RPDPf_{C_{ex}} = \frac{w(\{e_1,e_2\})}{w(\{e_1,e_2,e_3,e_4,e_5,e_6,e_7\})} = \frac{50}{91} \simeq 55\%$

∘

4.2 Diagnosticabilité d'événements

La notion de diagnosticabilité caractérise la capacité du système à donner suffisamment d'information pour permettre d'effectuer un bon diagnostic.

Une fois que nous nous sommes assurés qu'un événement est détectable, il nous faut regarder le caractère diagnosticable de cet événement, c'est-à-dire savoir si les informations fournies par le système et son monitoring sont suffisantes pour l'isoler lorsqu'il arrive.

Cette partie sur la diagnosticabilité est organisée comme la précédente sur la détectabilité, avec dans un premier temps des définitions caractérisant la diagnosticabilité d'un système sous diagnostic, puis dans un second temps des métriques s'appuyant sur ces définitions.

4.2.1 Définitions

Dans la littérature (e.g. [SSL+95a]), on trouve une définition générique de la diagnosticabilité d'un système qui caractérise le fait que les événements considérés sont tous détectables, qu'ils ont tous des effets différents et qu'il est ainsi possible de tous les distinguer les uns des autres. On rapproche cette notion de la notion d'*isolabilité*.

Une nouvelle fois, des variantes "faible" et "forte" seront distinguées, et les variantes faibles pourront ne pas être explicitées lorsqu'elles sont des adaptations simples des variantes fortes.

Définition 24 (Diagnosticabilité forte d'un événement) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. e est fortement diagnosticable s'il est fortement détectable et fortement discriminable de tout autre événement déclenchable dans C'.*

Exemple:

e_1 est fortement diagnosticable dans c_1.
e_3 est fortement diagnosticable dans c_1.
e_4 n'est pas fortement diagnosticable dans c_1.
e_1 n'est pas fortement diagnosticable dans C_{ex} car il n'y est pas fortement détectable.
e_2 n'est pas fortement diagnosticable dans C_{ex} car il n'y est pas fortement discriminable de e_3.
La diagnosticabilité de e_7 n'est pas définie dans $\{c_3, c_4\}$.

∘

Dire qu'un événement est fortement diagnosticable, c'est assurer que s'il est déclenché, alors non seulement on saura le détecter, mais de plus on saura exactement l'isoler. Ses effets lui sont propres.

Définition 25 (Diagnosticabilité faible d'un événement) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. e est faiblement diagnosticable s'il est détectable et discriminable de tout autre événement déclenchable dans C', sans être fortement diagnosticable pour autant.*

Exemple:

e_1 n'est pas faiblement diagnosticable dans c_1.
e_3 n'est pas faiblement diagnosticable dans C_{ex} car il n'y est pas discriminable de e_2.
e_1 est faiblement diagnosticable dans C_{ex}.
e_2 est faiblement diagnosticable dans C_{ex}.

∘

Nous présentons en table 4.4 la diagnosticabilité des événements de E_{ex} pour l'exemple.

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
diagnosticabilité de e_1 dans C'	F	ndiag	*nd*	ndiag	f
diagnosticabilité de e_2 dans C'	F	*nd*	F	F	f
diagnosticabilité de e_3 dans C'	F	*nd*	*nd*	*nd*	ndiag
diagnosticabilité de e_4 dans C'	ndiag	*nd*	*nd*	*nd*	ndiag
diagnosticabilité de e_5 dans C'	ndiag	*nd*	*nd*	*nd*	ndiag
diagnosticabilité de e_6 dans C'	ndiag	*nd*	*nd*	*nd*	ndiag
diagnosticabilité de e_7 dans C'	ndiag	*nd*	*nd*	*nd*	ndiag

où *nd*, *ndiag*, *f* et *F* signifient respectivement *non déclenchable* ($\Longleftrightarrow$ diagnosticabilité non définie), *non diagnosticable*, *faiblement diagnosticable* et *fortement diagnosticable.*

TABLE 4.4 – Diagnosticabilité des événements de E_{ex}

La notion de diagnosticabilité se généralise ensuite pour caractériser le système sous diagnostic entier.

Définition 26 (Diagnosticabilité forte) *Soit C' un ensemble de configurations. Un système est dit* fortement diagnosticable *dans C' si tous les événements qui y sont déclenchables y sont aussi fortement diagnosticables.*

Exemple:

$\mathcal{A}_{ex}$ n'est pas fortement diagnosticable dans C_{ex}.

∘

Définition 27 (Diagnosticabilité faible) *Soit C' un ensemble de configurations. Un système est dit* faiblement diagnosticable *dans C' s'il n'est pas fortement diagnosticable et si tous les événements qui sont déclenchables dans C' y sont aussi diagnosticables.*

Exemple:

$\mathcal{A}_{ex}$ n'est pas faiblement diagnosticable dans C_{ex}.

∘

La notion de diagnosticabilité que l'on retrouve dans la littérature, exprimée à travers les définitions 24 et 25, classe les événements en deux catégories : diagnosticables et non diagnosticables. Cette définition n'est pas totalement adaptée à notre contexte car est trop forte. Dans un système avionique, savoir distinguer tous les événements les uns des autres revient à réussir à tous les surveiller indépendemment, et à avoir une relation de bijection entre les observations et les événements, comme nous le faisions remarquer dans la partie 3.5, ce qui n'est pas possible. C'est impossible physiquement car surveiller requiert de la place et ajoute une masse non négligeable à l'avion, et ce n'est de toute manière pas envisageable car cela aurait un coût beaucoup trop important.

Les définitions généralisées au système sous diagnostic (déf. 26 et 27) sont encore moins adaptées. Elles ne sont jamais vraies pour un système réel et n'apportent dans ce cas aucune information.

Mais la notion de diagnosticabilité peut être affinée. Nous voulons faire plus que regarder uniquement *si* les événements sont diagnosticables ou pas. Nous voulons une notion plus précise qui nous permet de regarder plutôt *dans quelle mesure* les événements peuvent être dissociés les uns des autres au moment du diagnostic.

Pour ce faire, nous nous intéressons par la suite à identifier les événements qui peuvent avoir les mêmes effets et qui peuvent de ce fait être confondus lors d'un diagnostic. Cela permet d'introduire la notion de groupe d'ambiguité.

Pour introduire la notion de groupe d'ambiguité, considérons e_I, un événement défini de telle façon que $\forall i \in I, (i, e_I, i) \in T$. Ainsi pour tout ensemble de configurations C', nous avons $effets(C', e_I) = (C' \cap I)[V_O]$. Cet événement virtuel qui n'appartient pas à l'ensemble E, est appelé *"événement initial"*. Il sera utile pour caractériser les événements dont les effets ne sont pas distinguables des configurations initiales. On retrouve cette notion dans la littérature, avec par exemple la notion de *"mode normal"* (*normal mode*) dans [Puc08], ou de *"cas normal"* (ou *"cas sans défaut"*) dans [Bat11].

Définition 28 (Groupe d'ambiguité d'un événement) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. La fonction $groupeAmb : \mathcal{P}(C) \times E^{C'} \longrightarrow \mathcal{P}(E^{C'} \cup \{e_I\})$ donne l'ensemble des événements qui partagent des effets avec e dans C'.*

$$groupeAmb(C', e) = \{e' \in E^{C'} \cup \{e_I\} \mid effets(C', e') \cap effets(C', e) \neq \emptyset\} \tag{4.19}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$groupeAmb(C', e_1)$	$\{e_1\}$	$\{e_1\}$	nd	$\{e_1\}$	$\{e_1\}$
$groupeAmb(C', e_2)$	$\{e_2\}$	nd	$\{e_2\}$	$\{e_2\}$	$\{e_2, e_3\}$
$groupeAmb(C', e_3)$	$\{e_3\}$	nd	nd	nd	$\{e_2, e_3\}$
$groupeAmb(C', e_4)$	$\{e_4, e_5, e_6\}$	nd	nd	nd	$\{e_4, e_5, e_6\}$
$groupeAmb(C', e_5)$	$\{e_4, e_5, e_6\}$	nd	nd	nd	$\{e_4, e_5, e_6\}$
$groupeAmb(C', e_6)$	$\{e_4, e_5, e_6\}$	nd	nd	nd	$\{e_4, e_5, e_6\}$
$groupeAmb(C', e_7)$	$\{e_7, e_I\}$	nd	nd	nd	$\{e_7, e_I\}$

où nd *signifie* non défini.

∘

Le groupe d'ambiguité d'un événement e contient l'ensemble des causes avec lesquelles e peut être confondu. Il contient l'ensemble des événements qui ne sont pas fortement discriminables de e. En effet, si les événements produisent la même situation, il est impossible pour le moteur de diagnostic cherchant à diagnostiquer cette situation de donner un résultat de diagnostic complet et correct qui ne contienne pas tous ces événements. Si le résultat n'est pas précis au niveau du diagnostic, il est important de pouvoir identifier que cela vient de la stratégie de monitoring et du design du système, et non forcément du moteur de diagnostic qui ne pourrait pas être plus précis tout en restant correct.

L'introduction de l'événement e_I dans la définition du groupe d'ambiguité permet de prendre en compte l'ambiguité qui peut exister entre l'effet d'un événement et les configurations initiales. À titre d'illustration, considérons deux exemples de systèmes sous diagnostic, A_1 et A_2, dont la

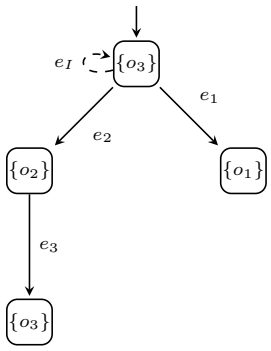


FIGURE 4.1 – Système A_1

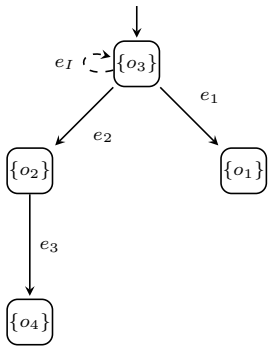


FIGURE 4.2 – Système A_2

représentation proposée en figures 4.1 et 4.2 ne présente que la partie observable des configurations du système.

Considérons que pour ces exemples d'illustration, $Obs_{OK} = \{\{\}\}$. Nous remarquons que pour chacun de ces exemples, si l'on ne tient pas compte de l'événement e_I, tous les événements sont détectables et perceptibles dans l'ensemble de toutes les configurations. En outre, si nous ne considérions pas l'événement e_I, le groupe d'ambiguité de chaque événement ne contiendrait que l'événement lui-même.

Les deux systèmes exposés auraient alors exactement les mêmes caractéristiques. Or, dans le système A_1, nous voyons qu'il y a en réalité deux cas qui peuvent expliquer l'observation $\{o_3\}$, à la fois le cas où rien n'est arrivé, et celui provoqué par e_3 après une occurrence de e_2. Le système A_2 lui ne comporte pas cette ambiguité. Il est donc meilleur d'un point de vue diagnostic que le système A_1.

C'est donc pour gérer l'ambiguité qu'il peut y avoir avec les états initiaux et pour coller au plus près de ce que la notion d'ambiguité représente que nous avons introduit l'événement virtuel e_I. Ainsi, avec la définition que nous avons donnée, le groupe d'ambiguité de e_3 pour le système A_1 sera $\{e_3, e_I\}$ alors que celui pour le système A_2 ne contient que e_3.

Pour caractériser à quel point un événement peut être ambigu, nous introduisons son *degré de diagnosticabilité*.

Définition 29 (Degré de diagnosticabilité d'un événement) *Soient C' un ensemble de configurations et e un événement déclenchable dans C'. La fonction $degDiag : \mathcal{P}(C) \times E^{C'} \longrightarrow \mathbb{N}$ donne le degré d'ambiguité de e dans C'.*

$$degDiag(C', e) = |groupeAmb(C', e)| \tag{4.20}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$degDiag(C', e_1)$	1	1	nd	1	1
$degDiag(C', e_2)$	1	nd	1	1	2
$degDiag(C', e_3)$	1	nd	nd	nd	2
$degDiag(C', e_4)$	3	nd	nd	nd	3
$degDiag(C', e_5)$	3	nd	nd	nd	3
$degDiag(C', e_6)$	3	nd	nd	nd	3
$degDiag(C', e_7)$	2	nd	nd	nd	2

où nd *signifie* non défini.

∘

Le degré de diagnosticabilité d'un événement représente le nombre maximal de causes avec lesquelles il peut être confondu lors d'un diagnostic, l'événement lui-même y compris.

La définition de degré de diagnostic permet d'introduire la notion de n-diagnosticabilité d'un événement.

Définition 30 (n-diagnosticabilité d'un événement) *Soit C' un ensemble de configurations, soit e un événement déclenchable dans C'. e est n-diagnosticable dans C' si son degré de diagnosticabilité est n.*

Exemple:
e_4 est 3-diagnosticable dans c_1.
e_2 est 2-diagnosticable dans C_{ex}.

∘

La caractérisation des événements par leur degré de diagnosticabilité nous permet de connaître l'ambiguité pour chaque événement du système. Intuitivement, l'ambiguité d'un événement peut être associée avec le nombre maximal de tentatives nécessaires pour isoler la cause. Par exemple, si nous considérons un événement d'ambiguïté 2, autrement dit, qui n'est pas fortement discriminable d'un autre événement, il faudra au maximum éliminer un autre cas avant de pouvoir l'isoler.

Dans cette optique, il est intéressant de regarder l'ensemble des événements qui ont le même degré de diagnosticabilité, qui représente alors les événements isolables en un même nombre maximal de tentatives. Pour cela, nous définissons la relation d'isodiagnosticabilité.

Définition 31 (Relation d'isodiagnosticabilité) *Soit C' un ensemble de configurations. La relation d'isodiagnosabilité $\approx_{C'}$ entre événements déclenchables dans C' caractérise les événements qui ont le même degré de diagnosticabilité dans C'.*

$$\forall (e, e') \in (E^{C'})^2, e \approx_{C'} e' \iff degDiag(C', e) = degDiag(C', e') \tag{4.21}$$

Propriété 3 *La classe d'équivalence d'un événement e pour la relation $\approx_{C'}$ représente la* classe de diagnosticabilité *de e dans C' et est notée $[e]_{\approx_{C'}}$.*

Si deux événements e et e' sont liés par $\approx_{C'}$, alors on dit que e et e' sont *isodiagnosticables* dans C'.

Exemple:

C'	$E^{C'}/\approx_{C'}$
c_1	$\{\{e_1, e_2, e_3\}, \{e_4, e_5, e_6\}, \{e_7\}\}$
c_3	$\{\{e_1\}\}$
c_4	$\{\{e_2\}\}$
$\{c_3, c_4\}$	$\{\{e_1, e_2\}\}$
C_{ex}	$\{\{e_1\}, \{e_2, e_3, e_7\}, \{e_4, e_5, e_6\}\}$

où $E^{C'}/\approx_{C'}$ représente l'ensemble quotient de $E^{C'}$ par la relation d'équivalence $\approx_{C'}$.

∘

4.2.2 Métriques

Dans cette partie nous introduisons différentes métriques qui peuvent être appliquées pour mesurer la diagnosticabilité des événements d'un système sous diagnostic.

Considérons une métrique qui s'appuie sur les définitions 24 et 25 et qui considère la proportion des événements du système qui sont diagnosticables.

Métrique 9 (Ratio de diagnosticabilité forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio de diagnosticabilité forte (resp. faible) *$RDiagF_{C'}$ (resp. $RDiagf_{C'}$) donne la proportion d'événements fortement (resp. faiblement) diagnosticables dans C', où ne sont considérés que les événements qui y sont déclenchables.*

$$RDiagF_{C'} = \frac{|\{e \in E^{C'} \mid e \text{ est fortement diagnosticable pour } C'\}|}{|E^{C'}|} \tag{4.22}$$

$$RDiagf_{C'} = \frac{|\{e \in E^{C'} \mid e \text{ est faiblement diagnosticable pour } C'\}|}{|E^{C'}|} \tag{4.23}$$

Exemple:

$RDiagF_{c_1} = \frac{3}{7} \simeq 43\%$.
$RDiagF_{C_{ex}} = \frac{0}{7} = 0\%$.

$RDiagf_{c_1} = \frac{0}{7} = 0\%$.
$RDiagf_{C_{ex}} = \frac{2}{7} \simeq 29\%$.

◦

Comme nous l'avons expliqué, la notion habituelle de diagnosticabilité n'est pas la plus appropriée dans notre contexte. Intéressons-nous plutôt à la notion d'ambiguité qui raffine celle de diagnosticabilité pour définir de nouvelles métriques.

Métrique 10 (Degré de diagnosticabilité moyen) *Soit C' un ensemble de configurations. Le degré de diagnosticabilité moyen $DDM_{C'}$ caractérise la diagnosticabilité du système dans C'.*

$$DDM_{C'} = \frac{\sum_{e \in E^{C'}} degDiag(C', e)}{|E^{C'}|} \tag{4.24}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$DDM_{C'}$	$\frac{14}{7} = 2$	$\frac{1}{1} = 1$	$\frac{1}{1} = 1$	$\frac{2}{2} = 1$	$\frac{16}{7} \simeq 2.29$

◦

Le degré de diagnosticabilité moyen représente l'ambiguité moyenne des événements du système. Plus le degré de diagnostic moyen s'approche de 1, meilleur il est. Un système totalement diagnosticable aura un degré de diagnostic moyen de 1.

Après avoir expérimenté cette métrique sur des exemples variés et après avoir exposé et analysé ses résultats avec des experts diagnostics, la conclusion est que, dans le cadre de notre travail, le degré de diagnosticabilité moyen *constitue un bon indicateur* pour évaluer un système dans son ensemble du point de vue de la diagnosticabilité.

Tout comme pour certaines métriques de détectabilité (e.g. mét. 5), nous pouvons regarder l'intégration d'une fonction de pondération dans la définition de la métrique précédente. On peut considérer ainsi que le degré de diagnosticabilité des événements qui sont moins importants, au sens où ils ont un poids inférieur, doit être moins pris en compte que le degré des événements plus importants.

Métrique 11 (Degré de diagnosticabilité moyen pondéré) *Soit C' un ensemble de configurations. Le degré de diagnosticabilité moyen pondéré $DDMP_{C'}$ caractérise la diagnosticabilité du système dans C', où les événements sont pondérés par une fonction w.*

$$DDMP_{C'} = \frac{\sum_{e \in E^{C'}} degDiag(C', e) * w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{4.25}$$

Exemple:

Utilisons la fonction de pondération w_{ex} définie pour la métrique 5.

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$DDMP_{C'}$	$\frac{152}{91} \simeq 1.67$	$\frac{40}{40} = 1$	$\frac{10}{10} = 1$	$\frac{50}{50} = 1$	$\frac{172}{91} \simeq 1.89$

◦

Nous pouvons finalement définir une métrique basée sur la relation d'isodiagnosticabilité.

Métrique 12 (Ratios de diagnosticabilité) *Soient C' un ensemble de configurations et n un entier strictement positif. Un ratio de diagnosticabilité $RDiag^n_{C'}$ représente la proportion d'événements n-diagnosticables dans C', où ne sont considérés que les événements qui y sont déclenchables.*

$$RDiag^n_{C'} = \frac{|[e]_{\approx_{C'}}|}{|E^{C'}|} \text{ pour un } e \text{ tel que } degDiag(C', e) = n \tag{4.26}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$RDiag^1_{C'}$	$\frac{3}{7} \simeq 43\%$	$\frac{1}{1} = 100\%$	$\frac{1}{1} = 100\%$	$\frac{2}{2} = 100\%$	$\frac{1}{7} \simeq 14\%$
$RDiag^2_{C'}$	$\frac{1}{7} \simeq 14\%$	nd	nd	nd	$\frac{3}{7} \simeq 43\%$
$RDiag^3_{C'}$	$\frac{3}{7} \simeq 43\%$	nd	nd	nd	$\frac{3}{7} \simeq 43\%$

∘

Comme $\approx_{C'}$ est une relation d'équivalence sur $E^{C'}$, l'ensemble de toutes les classes d'équivalence de $E^{C'}$ (ensemble quotient $E^{C'}/\approx_{C'}$) forme une partition de $E^{C'}$. Il y a donc un nombre fini de ratios. Plus précisément, aucun événement ne peut être plus que $|E^{C'}|$-diagnosticable, donc aucun ratio avec un n supérieur à $|E^{C'}|$ ne sera calculé. De plus, le cas où $RDiag^{|E^{C'}|}_{C'}$ sera calculé est celui où tous les événements sont ambigus et donc $|E^{C'}|$-diagnosticables.

Les ratios de diagnosticabilité donnent la répartition des événements du système selon leur degré de diagnosticabilité.

4.3 Performances adaptées au contexte avionique

Le fait que notre travail s'inscrive dans le contexte particulier qu'est l'avionique nous permet de réfléchir à la particularisation des notions de détectabilité et de diagnosticabilité introduites dans les sections précédentes (4.1 et 4.2).

En effet, les systèmes avioniques ont des particularités structurelles qui peuvent être exploitées pour raffiner les définitions et métriques introduites. Nous allons exploiter notamment les particularités qu'ont certains types de composants. Nous utiliserons aussi le fait que notre travail s'inscrit dans le contexte de la maintenance avionique pour donner de nouvelles définitions et métriques.

4.3.1 Détectabilité

4.3.1.1 Définitions

Les définitions générales sur la détectabilité (4.1.1) sont directement utilisables pour le diagnostic des systèmes avioniques. Nous pouvons néanmoins affiner les définitions en particularisant le problème au contexte avionique auquel nous nous intéressons. Nous nous concentrons non plus sur les événements de manière globale, mais sur les *défaillances*, et pouvons ainsi introduire des définitions plus spécifiques en utilisant les caractéristiques des défaillances, comme leur type ou leur probabilité d'occurrence.

Les définitions qui ont été introduites précédemment pour l'ensemble des événements E sont applicables de la même manière pour l'ensemble des défaillances $F \subseteq E$.

Pour un système avionique, considérons :

- $Comp = Comp_m \uplus Comp_l \uplus Comp_f$, l'ensemble des composants du système où $Comp_m$ représente les composants matériels, $Comp_l$ les composants logiques et $Comp_f$ les composants fonctionnels (ou fonctions).
- $Comp_a \subseteq Comp$, l'ensemble des composants actifs du système. En effet, certains composants sont installés mais non actifs dans certains systèmes, par exemple pour permettre d'ajouter

des fonctionnalités pas encore développées par la suite, ou encore dans le cas d'une utilisation partielle d'un équipement, comme un switch qui n'utiliserait pas tous ses ports.

La fonction $comp : F \longrightarrow Comp$ donne le composant auquel est associée une défaillance.

Exemple:

Pour l'exemple, considérons que :

- $F_{ex} = E_{ex}$.
- $Comp_{m_{ex}} = \{comp_1, comp_2, comp_3, comp_4, comp_5, comp_7\}$.
- $Comp_{l_{ex}} = \{comp_6\}$.
- $Comp_{a_{ex}} = \{comp_1, \ldots, comp_5\}$.

On considère en outre que la fonction *comp* est telle que pour toute défaillance e_i de F_{ex}, $comp(e_i) = comp_i$.

∘

Nous pouvons grouper les défaillances selon le type de composants auquel elles sont associées. Considérons plus particulièrement $F_m = \{f \in F \mid comp(f) \in Comp_m\}$ l'ensemble des défaillances matérielles et $F_a = \{f \in F \mid comp(f) \in Comp_a\}$ l'ensemble des défaillances actives.

Exemple:

$F_{m_{ex}} = \{e_1, e_2, e_3, e_4, e_5, e_7\}$.
$F_{a_{ex}} = \{e_1, \ldots, e_5\}$.

∘

4.3.1.2 Métriques

De nombreuses métriques concernant la détectabilité peuvent être calculées en utilisant les différentes caractéristiques que nous avons décrites comme les défaillances matérielles, actives, ou en alliant aussi détectabilité et perceptibilité. Comme lors des définitions et métriques de détectabilité dans le contexte général, nous n'expliciterons que la version "forte" de la métrique.

Métrique 13 (Ratio de détectabilité matérielle forte (resp. faible)) *Soit C' un ensemble de configurations. Le ratio de détectabilité matérielle forte (resp. faible) $RDMF_{C'}$ (resp. $RDMf_{C'}$) donne la proportion de défaillances matérielles fortement (resp. faiblement) détectables dans C', où ne sont considérées que les défaillances matérielles qui y sont déclenchables.*

$$RDMF_{C'} = \frac{|F_m \cap F_{FD}^{C'}|}{|F_m \cap F^{C'}|} \tag{4.27}$$

$$RDMf_{C'} = \frac{|F_m \cap F_{fD}^{C'}|}{|F_m \cap F^{C'}|} \tag{4.28}$$

Exemple:

$RDMF_{c_1} = \frac{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap \{e_1,e_2,e_3,e_4,e_5\}|}{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap F_{ex}|} = \frac{5}{6} \simeq 83\%$
$RDMF_{C_{ex}} = \frac{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap \{e_2,e_3,e_4,e_5\}|}{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap F_{ex}|} = \frac{4}{6} \simeq 67\%$

$RDMf_{c_1} = \frac{0}{6} = 0\%$
$RDMf_{C_{ex}} = \frac{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap \{e_1\}|}{|\{e_1,e_2,e_3,e_4,e_5,e_7\} \cap F_{ex}|} = \frac{1}{6} \simeq 17\%$

∘

Il est fréquent dans le monde avionique lorsque la détectabilité d'un système est analysée de ne regarder que les pannes matérielles. Cette focalisation est pertinente car elle permet d'utiliser certaines caractéristiques auxquelles sont associées uniquement les défaillances de ce type. Nous trouvons par exemple la notion de *taux de défaillance* qui représente la probabilité d'occurrence d'une panne matérielle. Ce sont des informations que l'on retrouve typiquement dans des documents comme l'AMDEC (Analyse des Modes de Défaillances, de leurs Effets et de leur Criticité),

appelée aussi par sa traduction anglaise FMEA (*Failure Mode and Effects Analysis*), utilisés dans le domaine de la sûreté de fonctionnement notamment.

Il est donc possible de particulariser la métrique 5 en utilisant la fonction usuellement notée $\lambda : F_m \longrightarrow \mathbb{Q}$ qui caractérise la probabilité d'occurrence d'une défaillance matérielle en nombre d'occurrences par heure de vol.

Métrique 14 (Ratio pondéré de détectabilité matérielle forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio pondéré de détectabilité matérielle forte (resp. faible) *$RPDMF_{C'}$ (resp. $RPDMf_{C'}$) donne la proportion de défaillances matérielles fortement (resp. faiblement) détectables dans C', où ne sont considérées que les défaillances matérielles qui y sont déclenchables, et où les défaillances sont pondérées par leur taux de défaillance.*

$$RPDMF_{C'} = \frac{\sum\limits_{f \in F_m \cap F_{FD}^{C'}} \lambda(f)}{\sum\limits_{f \in F_m \cap F^{C'}} \lambda(f)} \tag{4.29}$$

$$RPDMf_{C'} = \frac{\sum\limits_{f \in F_m \cap F_{fD}^{C'}} \lambda(f)}{\sum\limits_{f \in F_m \cap F^{C'}} \lambda(f)} \tag{4.30}$$

Exemple:

Considérons $\lambda_{ex} : F_m \longrightarrow \mathbb{Q}$, la fonction qui donne le taux d'occurrence des défaillances :

f	e_1	e_2	e_3	e_4	e_5	e_7
$\lambda_{ex}(f)$	10^{-1}	10^{-3}	10^{-3}	10^{-3}	10^{-3}	10^{-9}

$RPDMF_{c_1} = \frac{4*10^{-3}+10^{-1}}{4*10^{-3}+10^{-1}+10^{-9}} \simeq 100\%$.
$RPDMF_{C_{ex}} = \frac{4*10^{-3}}{4*10^{-3}+10^{-1}+10^{-9}} \simeq 3,8\%$.
$\rightarrow$ *Le fait de ne pas détecter e_1 qui a un fort taux d'occurrence impacte fortement le résultat de la métrique.*

$RPDMf_{c_1} = \frac{0}{4*10^{-3}+10^{-1}+10^{-9}} = 0\%$.
$RPDMf_{C_{ex}} = \frac{10^{-1}}{4*10^{-3}+10^{-1}+10^{-9}} \simeq 96,2\%$.
$\rightarrow$ *Ici, l'événement e_1, qui a un fort poids, est faiblement détectable et permet donc d'obtenir un bon résultat.*

∘

Nous avons choisi d'exprimer une métrique de détectabilité utilisant le taux de défaillance car c'est une métrique très utilisée dans le monde avionique pour évaluer un système. Comme l'exprime la métrique 5, la fonction λ peut très bien être remplacée par une autre fonction.

Nous allons maintenant construire un ratio qui ne considère que les défaillances qui sont actives dans le système. Avec cette option, nous considérons qu'il n'est pas pertinent de prendre en compte la détectabilité des défaillances associées à des composants du système qui ne sont pas actifs.

Métrique 15 (Ratio de détectabilité matérielle active forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio de détectabilité matérielle active forte (resp. faible) *$RDMAF_{C'}$ (resp. $RDMAf_{C'}$) donne la proportion de défaillances matérielles actives fortement (resp. faiblement) détectables dans C', où ne sont considérées que les défaillances matérielles actives qui y sont déclenchables.*

$$RDMAF_{C'} = \frac{|F_m \cap F_{FD}^{C'} \cap F_a|}{|F_m \cap F^{C'} \cap F_a|} \tag{4.31}$$

$$RDMAf_{C'} = \frac{|F_m \cap F_{fD}^{C'} \cap F_a|}{|F_m \cap F^{C'} \cap F_a|} \tag{4.32}$$

Exemple:

$RDMAF_{c_1} = \frac{|\{e_1,...,e_5,e_7\} \cap \{e_1,...,e_5\} \cap \{e_1,...,e_5\}|}{|\{e_1,...,e_5,e_7\} \cap F_{ex} \cap \{e_1,...,e_5\}|} = \frac{5}{5} = 100\%$
$RDMAF_{C_{ex}} = \frac{|\{e_1,...,e_5,e_7\} \cap \{e_2,...,e_5\} \cap \{e_1,...,e_5\}|}{|\{e_1,...,e_5,e_7\} \cap F_{ex} \cap \{e_1,...,e_5\}|} = \frac{4}{5} = 80\%$

$RDMAf_{c_1} = \frac{0}{5} = 0\%$
$RDMAf_{C_{ex}} = \frac{|\{e_1,...,e_5,e_7\} \cap \{e_1\} \cap \{e_1,...,e_5\}|}{|\{e_1,...,e_5,e_7\} \cap F_{ex} \cap \{e_1,...,e_5\}|} = \frac{1}{5} = 20\%$

∘

Nous pouvons intégrer la pondération par le taux de défaillance λ dans la métrique précédente.

Métrique 16 (Ratio pondéré de détectabilité matérielle active forte (resp. faible)) *Soit C' un ensemble de configurations. Le* ratio pondéré de détectabilité matérielle active forte (resp. faible) *$RPDMIF_{C'}$ (resp. $RPDMIf_{C'}$) donne la proportion de défaillances matérielles actives fortement (resp. faiblement) détectables dans C', où ne sont considérées que les défaillances matérielles actives qui y sont déclenchables, et où les défaillances sont pondérées par la fonction λ.*

$$RPDMIF_{C'} = \frac{\sum_{f \in F_m \cap F_{FD}^{C'} \cap F_a} \lambda(f)}{\sum_{f \in F_m \cap F^{C'} \cap F_a} \lambda(f)} \tag{4.33}$$

$$RPDMIf_{C'} = \frac{\sum_{f \in F_m \cap F_{fD}^{C'} \cap F_a} \lambda(f)}{\sum_{f \in F_m \cap F^{C'} \cap F_a} \lambda(f)} \tag{4.34}$$

Exemple:

$RPDMIF_{c_1} = \frac{4*10^{-3}+10^{-1}}{4*10^{-3}+10^{-1}} = \frac{5}{5} = 100\%$
$RPDMIF_{C_{ex}} = \frac{4*10^{-3}}{4*10^{-3}+10^{-1}} \simeq 3,8\%$

$RPDMIf_{c_1} = \frac{0}{5} = 0\%$
$RPDMIf_{C_{ex}} = \frac{10^{-1}}{4*10^{-3}+10^{-1}} \simeq 96,2\%$

∘

Toutes les métriques précédentes définies sur la notion de détectabilité peuvent être appliquées à la notion de perceptibilité. De manière similaire nous pouvons obtenir le *ratio de perceptibilité matérielle forte $RPMF_{C'}$* ainsi que toutes les métriques dérivées.

Finalement, nous pouvons utiliser la notion combinée de détectabilité et perceptibilité pour définir la métrique suivante.

Métrique 17 (Ratio de détectabilité et perceptibilité matérielles forte) *Soit C' un ensemble de configurations. Le ratio de détectabilité et perceptibilité matérielles fortes $RDPMF_{C'}$ donne la proportion de défaillances matérielles fortement détectables et perceptibles dans C', où ne sont considérées que les défaillances matérielles qui y sont déclenchables.*

$$RDPMF_{C'} = \frac{|F_m \cap F_{FD}^{C'} \cap F_P^{C'}|}{|F_m \cap F^{C'}|} \tag{4.35}$$

Exemple:

$RDPM_{c_1} = \frac{|\{e_1,...,e_5,e_7\} \cap \{e_1,...,e_5\} \cap \{e_1,...,e_5\}|}{|\{e_1,...,e_5,e_7\} \cap F_{ex}|} = \frac{5}{6} \simeq 83\%.$
$RDPM_{C_{ex}} = \frac{|\{e_1,...,e_5,e_7\} \cap \{e_2,...,e_5\} \cap \{e_1,e_3,e_4,e_5\}|}{|\{e_1,...,e_5,e_7\} \cap F_{ex}|} = \frac{3}{6} = 50\%.$

∘

L'ensemble de ces métriques peut être étoffé par la création d'autres ratios en combinant les différents ensembles de défaillances (défaillances détectables, perceptibles, matérielles, actives...) définis dans la section 4.3.1.1 ainsi qu'en utilisant des fonctions de pondération.

4.3.2 Diagnosticabilité

4.3.2.1 Définitions

Les définitions générales sur la diagnosticabilité données précédemment (4.2.1) sont directement applicables pour le diagnostic de systèmes avioniques. Comme dans la partie correspondante pour la détectabilité (4.3.1.1), nous allons pouvoir affiner les définitions précédentes en utilisant les caractéristiques des systèmes avioniques. Les définitions spécifiques qui seront introduites concernent cette fois-ci plus particulièrement l'utilisation du diagnostic pour la maintenance avionique.

Pour un système avionique, en plus des précédentes définitions sur la détectabilité adaptées au contexte avionique (4.3.1.1), considérons que $RU \subseteq Comp_H$ est l'ensemble des *unités remplaçables*. Une unité remplaçable est un composant qui peut être remplacé pendant les opérations de maintenance. L'unité remplaçable représente un bon niveau d'abstraction d'un résultat de diagnostic pour la maintenance.

Exemple:

Pour l'exemple, considérons que $RU_{ex} = \{comp_1, comp_2, comp_3, comp_4, comp_7\}$.

∘

Avec la notion d'unité remplaçable (RU), nous pouvons raffiner la définition de la diagnosticabilité afin qu'elle soit plus adaptée au contexte du diagnostic pour la maintenance avionique. En effet, un opérateur de maintenance ne s'intéresse pas à savoir au final qu'elle est la défaillance exacte qui a impacté le système, mais plutôt quelle est l'unité remplaçable qu'il doit changer pour remettre le système en état opérationnel.

Définition 32 (RUs associées à un ensemble de défaillances) *Soit $F' \subseteq F \cup \{e_I\}$. La fonction $comps_{RU} : \mathcal{P}(F \cup \{e_I\}) \longrightarrow \mathcal{P}(RU \cup \{comp_I\})$ donne les unités remplaçables qui sont associées aux défaillances de F'.*

$$comps_{RU}(F') = \{ru \in RU \mid \exists f \in F', f \text{ est associée au } RU\ ru\} \tag{4.36}$$

où e_I est associée à $comp_I$.

Exemple:

Considérons $comp_{RU_{ex}} : F_{ex} \cup \{e_I\} \longrightarrow RU_{ex} \cup \{comp_I\}$ telle que :

f	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_I
$comp_{RU_{ex}}(f)$	$comp_1$	$comp_2$	$comp_3$	$comp_4$			$comp_7$	$comp_I$

Nous avons alors $comps_{RU_{ex}} : \mathcal{P}(F_{ex} \cup \{e_I\}) \longrightarrow \mathcal{P}(RU \cup \{comp_I\})$ telle que pour $F' \subseteq F_{ex}$, $comps_{RU_{ex}}(F') = \{comp_{RU_{ex}}(f) \mid f \in F'\}$.

Nous remarquons que le RU $comp_4$ contient les composants $comp_5$ et $comp_6$ auxquels sont associés e_5 et e_6 (cf. fonction comp définie en 4.3.1.1).

∘

L'introduction du composant $comp_I$ pour cette définition est dans la continuité de l'introduction de l'événement e_I pour la définition 28. Cela permet de continuer au niveau RU à prendre en compte l'ambiguité, portée par le groupe d'ambiguité pouvant contenir e_I, entre l'effet d'un événement et les configurations initiales.

Cette notion de RU nous permet de considérer non plus la notion de diagnosticabilité simple, mais la notion de RU-diagnosticabilité. Il est tout à fait logique, dans l'optique d'une utilisation pour la maintenance, par un agent de maintenance, de ne pas tenir compte de l'ambiguité entre deux défaillances qui sont associées à un même RU. Ce qu'il faut pour le diagnostic dans un but de maintenance c'est identifier le RU qui est à changer.

En intégrant cette notion de RU, nous pouvons reprendre et adapter certaines définitions que nous avons introduites en partie 4.2.1, à commencer par le degré de diagnosticabilité d'un événement qui devient le degré de RU-diagnosticabilité d'une défaillance.

Définition 33 (Degré de *RU*-diagnosticabilité d'une défaillance) *Soient C' un ensemble de configurations et f une défaillance déclenchable dans C'. La fonction $degDiag_{RU} : \mathcal{P}(C) \times F^{C'} \longrightarrow \mathbb{N}$ donne le degré d'ambiguité RU de f dans C'.*

$$degDiag_{RU}(C', f) = |comps_{RU}(groupeAmb(C', f))| \tag{4.37}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$degDiag_{RU}(C', e_1)$	1	1	nd	1	1
$degDiag_{RU}(C', e_2)$	1	nd	1	1	2
$degDiag_{RU}(C', e_3)$	1	nd	nd	nd	2
$degDiag_{RU}(C', e_4)$	1	nd	nd	nd	1
$degDiag_{RU}(C', e_5)$	1	nd	nd	nd	1
$degDiag_{RU}(C', e_6)$	1	nd	nd	nd	1
$degDiag_{RU}(C', e_7)$	2	nd	nd	nd	2

où nd *signifie* non défini.

∘

À partir de la définition du degré de RU-diagnosticabilité d'une défaillance, nous pouvons définir la notion de *n-RU*-diagnosticabilité d'une défaillance.

Définition 34 (n-RU-diagnosticabilité d'une défaillance) *Soient C' un ensemble de configurations et f une défaillance déclenchable dans C'. f est n-RU-diagnosticable si son degré de RU-diagnosticabilité est n.*

Exemple:
e_4 est 1-RU-diagnosticable pour c_1.
e_2 est 2-RU-diagnosticable pour C_{ex}.

∘

Nous pouvons maintenant introduire la relation de RU-isodiagnosticabilité.

Définition 35 (Relation de RU-isodiagnosticabilité) *Soit C' un ensemble de configurations. La relation de RU-isodiagnosabilité $\approx^{RU}_{C'}$ entre défaillances caractérise les défaillances qui ont le même degré de RU-diagnosticabilité dans C'.*

$$\forall (f, f') \in (F^{C'})^2, f \approx^{RU}_{C'} f' \iff degDiag_{RU}(C', f) = degDiag_{RU}(C', f') \tag{4.38}$$

Propriété 4 *La classe d'équivalence d'une défaillance f pour la relation $\approx^{RU}_{C'}$ représente la* classe de RU-diagnosticabilité *de f dans C' et est notée $[f]_{\approx^{RU}_{C'}}$.*

Si deux défaillances f et f' sont liées par $\approx^{RU}_{C'}$, alors on dit que f et f' sont *RU-isodiagnosticables* dans C'.

Exemple:

C'	$F^{C'} / \approx^{RU}_{C'}$
c_1	$\{\{e_1, e_2, e_3, e_4, e_5, e_6\}, \{e_7\}\}$
c_3	$\{\{e_1\}\}$
c_4	$\{\{e_2\}\}$
$\{c_3, c_4\}$	$\{\{e_1, e_2\}\}$
C_{ex}	$\{\{e_1, e_4, e_5, e_6\}, \{e_2, e_3, e_7\}\}$

où $F^{C'} / \approx^{RU}_{C'}$ représente l'ensemble quotient de $F^{C'}$ par la relation d'équivalence $\approx^{RU}_{C'}$.

∘

4.3.2.2 Métriques

Dans cette partie nous utilisons la notion d'unité remplaçable pour étendre quelques unes des précédentes métriques de diagnosticabilité (4.2.2), à commencer par le degré de diagnosticabilité moyen qui devient le *degré de RU-diagnosticabilité moyen.*

Métrique 18 (Degré de RU-diagnosticabilité moyen) *Soit C' un ensemble de configurations. Le degré de RU-diagnosticabilité moyen $RUDDM_{C'}$ caractérise la RU-diagnosticabilité du système dans C'.*

$$RUDDM_{C'} = \frac{\sum_{f \in F^{C'}} degDiag_{RU}(C', f)}{|F^{C'}|} \tag{4.39}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$RUDDM_{C'}$	$\frac{8}{7} \simeq 1.14$	$\frac{1}{1} = 1$	$\frac{1}{1} = 1$	$\frac{2}{2} = 1$	$\frac{10}{7} \simeq 1.43$

∘

Pour reprendre la métrique sur le degré de diagnosticabilité moyen pondéré et y intégrer la notion de *RU* comme dans la métrique 18 précédente, nous réutilisons le taux de défaillance λ introduit lors dans la métrique 14 de détectabilité. Pour cela, nous devons donc réduire le périmètre des défaillances aux seules défaillances matérielles.

Métrique 19 (Degré de RU-diagnosticabilité moyen pondéré) *Soit C' un ensemble de configurations. Le degré de diagnosticabilité moyen pondéré $RUDDMP_{C'}$ caractérise la diagnosticabilité du système dans C', où les défaillances matérielles sont pondérées par leur taux de défaillance λ.*

$$RUDDMP_{C'} = \frac{\sum_{f \in F^{C'} \cap F_m} degDiag_{RU}(C', f) * \lambda(f)}{\sum_{f \in F^{C'} \cap F_m} \lambda(f)} \tag{4.40}$$

Exemple:

Utilisons la fonction de pondération λ_{ex} définie pour la métrique 14.

$RUDDMP_{c_1} = \frac{10^{-1}+4*10^{-3}+2*10^{-9}}{10^{-1}+4*10^{-3}+10^{-9}} \simeq 1.$
$RUDDMP_{C_{ex}} = \frac{10^{-1}+6*10^{-3}+2*10^{-9}}{10^{-1}+4*10^{-3}+10^{-9}} \simeq 1.02.$

∘

Le degré de RU-diagnosticabilité moyen pondéré peut bien entendu, tout comme pour le ratio pondéré de détectabilité matérielle (mét. 14), être pondéré par une fonction autre que le taux de défaillance λ, sans devoir être ainsi restreint uniquement aux défaillances matérielles.

Nous pouvons finalement définir les ratios basés sur la relation d'isodiagnosticabilité.

Métrique 20 (Ratios de RU-diagnosticabilité) *Soient C' un ensemble de configurations et n un entier strictement positif. Un ratio de RU-diagnosticabilité $RURDiag^n_{C'}$ représente la proportion de défaillances n-RU-diagnosticables dans C', où ne sont considérées que les défaillances qui y sont déclenchables.*

$$RURDiag^n_{C'} = \frac{|[f]_{\approx^{RU}_{C'}}|}{|F^{C'}|} \text{ pour un } f \text{ tel que } degDiag_{RU}(C', f) = n \tag{4.41}$$

Exemple:

C'	c_1	c_3	c_4	$\{c_3, c_4\}$	C_{ex}
$RURDiag^1_{C'}$	$\frac{6}{7} \simeq 86\%$	$\frac{1}{1} = 100\%$	$\frac{1}{1} = 100\%$	$\frac{2}{2} = 100\%$	$\frac{4}{7} \simeq 57\%$
$RURDiag^2_{C'}$	$\frac{1}{7} \simeq 14\%$	nd	nd	nd	$\frac{3}{7} \simeq 43\%$
$RURDiag^3_{C'}$	$\frac{0}{7} = 0\%$	nd	nd	nd	$\frac{0}{7} = 0\%$

∘

Cette fois-ci, aucune défaillance ne sera plus que $|comps_{RU}(F^{C'})|$-diagnosticable et donc aucun ratio avec un n supérieur à $|comps_{RU}(F^{C'})|$ ne sera calculé.

4.4 Utilisation des relations de causes à effets pour évaluer un SUD

Lorsque nous considérons un système sous diagnostic à étudier, il n'est pas tout le temps possible de traiter directement les modèles de type *système à événements discrets*. Il est possible qu'un modèle par exemple soit d'une taille trop importante pour faire des analyses. Il arrive de ne pas avoir toutes les informations à travers un modèle complet du système mais uniquement une partie via des objets plus simples comme des séquences ou des coupes.

Il peut donc s'avérer intéressant de trouver d'autres moyens pour analyser les performances des systèmes sous diagnostic.

Dans cette section, nous introduisons tout d'abord les notions de *sous-effets* et de *sur-effets* et nous regarderons comment elles interagissent avec les métriques de diagnostic. Ensuite nous nous intéresserons à plusieurs relations de causes à effets qui peuvent être calculées à partir du système sous diagnostic et regarderons alors ce qu'il est possible de déduire de ces relations d'un point de vue performances de diagnostic.

4.4.1 Sur-effets et sous-effets

Les notions de *sur-effets* et de *sous-effets* sont relatives à la notion d'*effets* introduite en définition 15 page 88. Nous nous intéressons dans cette section à ce qu'il est possible de dire sur les performances du diagnostic d'un système lorsque nous n'avons pas sa relation d'effets mais que nous avons des relations dérivées que sont les sur-effets et les sous-effets.

Introduisons la relation de sur-effets à travers la définition qui suit.

Définition 36 (Sur-effets) *Une fonction* $\overline{effets} : \mathcal{P}(C) \times E^{C'} \longrightarrow \mathcal{V}(V_O)$ *est dite de* sur-effets *pour un système si elle contient tous les effets possibles des événements du système.*

$$effets \subseteq \overline{effets} \tag{4.42}$$

Avec la notion de sur-effets, nous allons pouvoir raisonner sur l'ensemble des effets possibles. Pour une propriété particulière, il sera possible d'affirmer qu'elle est vraie *pour tous les effets* du système en s'assurant qu'elle est vraie *pour tous les sur-effets*. Ceci est dû au fait que tous les effets sont des sur-effets.

De la même façon introduisons la notion de sous-effets.

Définition 37 (Sous-effets) *Une fonction* $\underline{effets} : \mathcal{P}(C) \times E^{C'} \longrightarrow \mathcal{V}(V_O)$ *est dite de* sous-effets *pour un système si elle ne contient que des effets possibles des événements du système.*

$$\underline{effets} \subseteq effets \tag{4.43}$$

Avec la notion de sous-effets, nous abordons la notion d'effets sous un autre angle. Avec ce type de relation, nous pourrons par exemple affirmer qu'*il existe un effet* pour lequel une propriété particulière est vraie lorsqu'*il existe un sous-effet* pour lequel elle est vérifiée. Ceci est dû au fait que tous les sous-effets sont des effets.

Notons que si les sur-effets sont les mêmes que les sous-effets, alors on obtient exactement les effets (on a à la fois tous les effets et uniquement des effets).

Nous souhaitons voir dans quelle mesure il est possible de caractériser les performances de diagnostic (définies relativement aux effets) avec des notions de sur-effets et de sous-effets. Pour cela, nous adressons les performances de diagnostic, et donc les notions de détectabilité et de diagnosticabilité, à travers les questions suivantes :

- Détectabilité :
  1. Peut-on affirmer qu'un événement est fortement détectable ?

2. Peut-on affirmer qu'un événement est non détectable ?
3. Peut-on affirmer qu'un événement est faiblement détectable ?
4. Peut-on affirmer qu'un événement est détectable ?

- Diagnosticabilité :

5. Peut-on affirmer qu'un événement est dans le groupe d'ambiguité d'un autre ?
6. Peut-on affirmer qu'un événement n'est pas dans le groupe d'ambiguité d'un autre ?

Pour répondre à ces questions, nous allons nous appuyer sur les notions de sur-effets et sous-effets que nous venons de définir. Nous allons prendre les questions une par une et voir dans quelle mesure nous pouvons y répondre.

4.4.1.1 Détectabilité forte

Dire qu'un événement est fortement détectable signifie qu'aucun de ses effets ne correspond à un effet de bon fonctionnement (déf. 18 page 90).

Comme il s'agit de faire une affirmation sur la totalité des effets d'un événement, l'important est d'avoir une relation qui comprend tous les effets. L'objet que nous considérons pour la *détectabilité forte* est donc l'objet *sur-effets*. On s'assure ici que pour tout événement e, il n'existe aucun effet associé de e qui est une observation de Obs_{OK}. Les sur-effets contenant tous les effets, nous sommes sûrs qu'en vérifiant cela pour les sur-effets, nous le vérifions aussi pour les effets.

Nous pouvons donc affirmer qu'un événement est fortement détectable si aucun des sur-effets qu'il provoque n'est une observation de bon fonctionnement. En revanche, l'implication inverse n'est pas vraie : il est tout à fait possible qu'un événement fortement détectable soit associé dans les sur-effets à une observation de bon fonctionnement. Cela est dû au fait qu'il peut exister des sur-effets qui ne correspondent pas à des effets dans le système.

4.4.1.2 Non détectabilité

On dit qu'un événement est non détectable si chacun de ses effets correspond à une observation de Obs_{OK}.

Nous nous intéressons une nouvelle fois à la vérification d'une propriété pour tous les effets des événements. Il convient donc d'utiliser à nouveau les *sur-effets*. Nous vérifions que si tous les sur-effets associés à un événement sont des observations de bon fonctionnement, alors l'événement est non détectable.

En revanche, et pour la même raison que pour la détectabilité forte, il est possible qu'un événement non détectable ait des sur-effets associés qui ne correspondent pas à une observation de bon fonctionnement, mais cela signifie alors que ces sur-effets ne sont pas des effets du système.

4.4.1.3 Détectabilité faible

Un événement est faiblement détectable s'il n'est ni fortement détectable, ni non détectable, i.e. s'il génère au moins un effet de bon fonctionnement (n'est pas fortement détectable) et au moins un effet qui n'est pas de bon fonctionnement (est détectable).

Cette fois-ci, nous nous intéressons donc à l'existence certaine des deux types d'effets. Nous nous appuyons logiquement dans ce cas sur la relation de *sous-effets*. Nous pouvons ainsi affirmer que si un événement est à la fois associé à un sous-effet qui est une observation de bon fonctionnement, et à un autre sous-effet qui n'en est pas une, alors l'événement est faiblement détectable. Nous pouvons affirmer cela car l'objet sous-effets ne contient que des observations qui sont des effets d'événement.

L'implication inverse est une nouvelle fois fausse. Du fait de l'absence potentielle d'effets dans la relation des sous-effets, il est possible qu'un événement faiblement détectable ne soit pas associé aux deux types d'observations alors même qu'en réalité, dans les effets, il l'est.

4.4.1.4 Détectabilité

Un événement est détectable s'il est soit faiblement détectable soit fortement détectable.

Il suffit qu'un événement ait pour effet une observation qui n'est pas de bon fonctionnement pour qu'il soit détectable. Il s'agit donc ici de s'assurer de la présence réelle d'un effet. Pour cela nous utilisons la notion de *sous-effets*. Si un événement est au moins associé à un sous-effet qui n'est pas dans Obs_{OK}, alors il est détectable.

L'implication inverse n'est pas vraie. En effet, un événement pourrait voir tous ses effets qui ne sont pas de bon fonctionnement disparaître dans la relation de sous-effets.

Une deuxième méthode permettant d'affirmer qu'un événement est détectable et utilisant la notion de *sur-effets* est possible. Il s'agit tout simplement de la méthode d'identification d'événements fortement détectables décrite en 4.4.1.1. En effet, nous l'avons précisé, tout événement faiblement ou fortement détectable est détectable.
En revanche, nous ne gagnons rien à utiliser la règle introduite pour les événements faiblement détectable car elle est couverte par celle que nous avons donnée quelques paragraphes plus haut dans cette partie.

4.4.1.5 Présence dans le groupe d'ambiguité

Un événement e_2 appartient au groupe d'ambiguité d'un événement e_1 si au moins un des effets de e_1 est un effet de e_2.

Pour pouvoir traiter cette propriété, il est nécessaire de s'assurer que les observations que l'on considère correspondent bien à des effets des événements du système. Pour cela, il est approprié d'utiliser la notion de *sous-effets*. Si deux événements partagent au moins une observation à laquelle ils sont associés dans les sous-effets, alors ils appartiennent chacun au groupe d'ambiguité de l'autre.

Par contre, une nouvelle fois, deux événements peuvent être dans le même groupe d'ambiguité sans pour autant partager une même observation dans la table des sous-effets. Tous les effets n'étant potentiellement pas présents dans la relation, il est possible que des effets que partagent deux événements ne soient pas dans les sous-effets.

4.4.1.6 Absence dans le groupe d'ambiguité

Un événement e_2 n'est pas dans le groupe d'ambiguité de e_1 (et vice et versa) si e_1 et e_2 n'ont aucun effet en commun.

Comme il s'agit de vérifier cette propriété sur l'ensemble de tous les effets, il est nécessaire pour ce faire de s'appuyer sur la notion de *sur-effets*. Si on vérifie que e_1 et e_2 ne partagent pas de sur-effet, on peut en déduire qu'ils n'ont pas d'effet commun, et donc que l'un n'apparaîtra pas dans le groupe d'ambiguité de l'autre. Cela revient à s'assurer de la discriminabilité forte des deux événements.

En revanche, et de manière logique une nouvelle fois à cause des imprécisions de la table des sur-effets (certaines associations *événement–observations* présentes dans la table des sur-effets ne correspondent pas à des effets), il est tout à fait possible que deux événements qui n'apparaissent pas dans le groupe d'ambiguité l'un de l'autre se voient partager des observations (qui ne correspondent donc pas à des effets réels) dans les sur-effets.

4.4.1.7 Synthèse

Dans cette section nous avons introduit les notions de sur-effets et sous-effets. À partir de ces notions, il est possible de caractériser la détectabilité et la diagnosticabilité des systèmes.

Pour la détectabilité nous pouvons tirer plusieurs conclusions :

- Il est possible de calculer pour le système une détectabilité forte minimum en utilisant les sur-effets.

- Il est possible de calculer pour le système une "non détectabilité" minimum en utilisant les sur-effets.
- Il est possible de calculer pour le système une détectabilité faible minimum en utilisant les sous-effets.
- Il est possible de calculer pour le système une détectabilité minimum en utilisant les sous-effets et les sur-effets.

Nous pouvons alors estimer les détectabilités forte, faible et globale minimum, ainsi qu'une détectabilité maximum (avec la valeur minimum de non détectabilité). Nous obtenons ainsi un encadrement de la valeur de détectabilité, et des détectabilités faible et forte.

Concernant la diagnosticabilité, nous avons introduit une règle permettant d'affirmer à partir des sur-effets qu'un événement n'appartient pas au groupe d'ambiguité d'un autre. Même si cette règle n'identifie pas tous ceux qui n'appartiennent pas au groupe d'ambiguité d'un événement, cela permet d'en éliminer quelques uns et ainsi de donner une taille maximale de groupe d'ambiguité pour un événement réduite. En appliquant cela à chaque événement, nous pouvons obtenir un degré de diagnosticabilité moyen maximum.

Le fait de pouvoir affirmer à partir des sous-effets que certains événements feront à coup sûr partie du groupe d'ambiguité d'un autre permet de donner une taille minimum pour les groupes d'ambiguité et donc une valeur minimum du degré de diagnosticabilité moyen. Nous obtenons ainsi un encadrement du degré de diagnosticabilité moyen.

4.4.2 Relations de causes à effets et performances de diagnostic

Nous venons d'introduire les notions de sur-effets et sous-effets. Nous avons vu leur impact sur les métriques de performances de diagnostic. Dans cette section, nous nous intéressons aux différents relations de causes à effets que l'on peut obtenir à partir d'un système sous diagnostic. Nous allons présenter dans quelle mesure nous pouvons atteindre les notions de sur-effets et sous-effets à partir de ces relations et allons en proposer des illustrations.

4.4.2.1 Système sous diagnostic réduit en profondeur

Une idée qui vient logiquement lorsque nous rencontrons des problèmes lors de la création de l'automate de comportement représentant le système sous diagnostic est de borner la profondeur du système que nous analysons. Cela correspond à la notion de profondeur introduite en section 3.1.

Cette considération, en plus d'être pratique d'un point de vue calculatoire, est aussi fréquente dans le domaine avionique dans lequel s'inscrit notre travail. En pratique, les effets générés par des événements multiples ne sont considérés que jusqu'à une certaine profondeur, souvent inférieure à trois.

Comme nous l'avons décrit en section 2.1.5, les techniques actuelles de diagnostic, à travers les tables de règles, associent à un événement l'ensemble des messages de monitoring qu'il génère. Ces messages correspondant aux valuations des variables d'observation de notre modèle, cela signifie que, dans les tables, un événement est associé à un ensemble de valuations de variables d'observation, i.e. à une observation. La notion d'effets que nous avons définie associe, elle, un événement à un ensemble d'observations représentant tous les effets possibles de cet événement.

Souvent, dans la pratique, les systèmes considérés sont totalement dysfonctionnels (ne contiennent que des événements de défaillances) et déterministes. L'observation associée à un événement correspond alors à l'effet généré par l'occurrence de cet événement depuis l'état nominal. Cela revient à travailler sur le système réduit à une profondeur maximale de 1 défaillance.

Considérons pour la suite de cette section 4.4.2.1 que $E_b = E_{nc}$. Vouloir appliquer nos métriques et définitions sur un système dont on a réduit la profondeur à n est simple. Nous pouvons faire cela de deux façons :

1. Pour les définitions et métriques relatives à un ensemble de configurations, il suffit de les paramétrer par l'ensemble de configurations C^{n-1}. Ainsi, choisissant les configurations sources dans C^{n-1}, cela revient à considérer les effets jusqu'à C^n.

2. Une autre solution est de considérer le système sous diagnostic projeté sur l'ensemble des configurations de C^n. Les définitions s'appliquent de la même manière sur le système restreint en profondeur que sur le système sous diagnostic de départ. Les effets sont à nouveau calculés jusqu'à C^n, ce qui rend cette solution équivalente à la première.

Nous pouvons illustrer ces méthodes avec l'exemple que nous suivons dans ce chapitre, en considérant une profondeur maximale de 1 événement. Pour avoir une application de la première méthode, il suffit de se rapporter aux exemples fournis dans ce chapitre, où l'ensemble de configurations pris en paramètres correspond aux états initiaux, i.e. $\{c_1\}$.

Pour la deuxième méthode, qui est donc équivalente, il suffit de considérer le système sous diagnostic A directement réduit à la profondeur 1. Nous retrouvons en figures 4.3 et 4.4 deux systèmes sous diagnostic, le système A_{ex} comme nous l'avons défini en début de chapitre (fig. 4.3), et le système qui correspond à la réduction du premier sur une profondeur maximale de 1 événement (fig. 4.4).

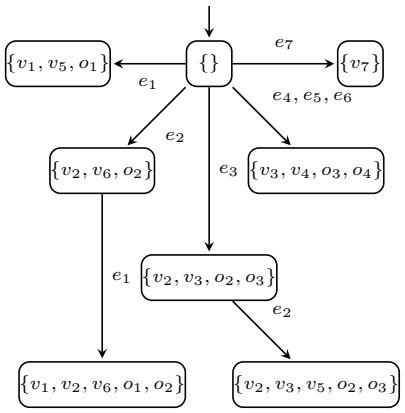


FIGURE 4.3 – Système sous diagnostic A_{ex}

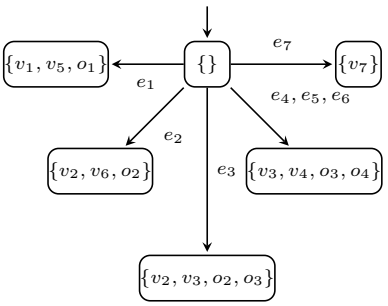


FIGURE 4.4 – A_{ex} réduit (prof. 1)

Nous avons vu que l'ensemble des définitions et métriques s'appliquent de la même manière que l'on considère un système sous diagnostic ou sa version réduite en profondeur. Néanmoins, les résultats obtenus peuvent être différents. En effet, en tronquant le système dans la profondeur, il est possible que les effets des événements changent. Plus exactement, certains effets d'un événement peuvent disparaître. Les effets calculés sur un système réduit en profondeur seront donc des sous-effets.

Pour avoir une illustration de ces propriétés, il suffit de mettre en comparaison les résultats des métriques obtenus sur le système sous diagnostic A_{ex} pour l'ensemble de toutes les configurations C_{ex} et pour l'ensemble de configurations $\{c_1\}$. Nous présentons en table 4.5 une comparaison des métriques obtenues pour le système réduit par rapport au système A_{ex} de départ.

D'après les remarques de le section précédente sur les sous-effets, nous pouvons dire qu'en évaluant les métriques de performances de diagnostic sur un système sous diagnostic réduit en profondeur, on obtiendra des valeurs de détectabilité (RD) et de diagnosticabilité (DDM) minimisant celles du système réel. Nous en avons d'ailleurs un exemple en table 4.5.

Ces valeurs minimum sont en pratique assez proche de la réalité dans notre contexte d'application, étant donné que les cas oubliés lors de la réduction de la profondeur maximale sont, comme nous l'avons précisé en 2.1.5, des pannes multiples, i.e. des cas de pannes moins probables.

Métriques pour C'	$\{c_1\}$	C_{ex}
Détectabilité		
$RD_{C'}$	$\frac{6}{7}$	$\frac{6}{7}$
Diagnosticabilité		
$DDM_{C'}$	2	$\frac{16}{7} \simeq 2.29$

TABLE 4.5 – Comparatif des résultats de métriques pour C_{ex} et $\{c_1\}$

4.4.2.2 Séquences

Une idée d'une méthode alternative pour exploiter les relations de causes à effets du système sous diagnostic dans le but d'évaluer sa détectabilité et diagnosticabilité est l'utilisation des séquences de défaillances.

Les séquences de défaillances, pour une observation donnée, représentent l'ensemble des séquences d'événements qui provoquent cette observation dans le système. Partant de là, une idée est de calculer les séquences pour chaque observation totalement valuée possible. Si l'observation pour laquelle on souhaite calculer les séquences n'est pas possible dans le système, alors nous n'obtiendrons aucune séquence en résultat.

Par la suite, nous exprimerons les observations totalement valuées par des ensembles de variables d'observation, faisant apparaître les variables valuées à vrai, comme nous le faisons avec l'ensemble des variables depuis le début de cette partie sur les performances de diagnostic. Nous ne ferons pas apparaître les observations pour lesquelles l'ensemble des séquences est vide, cela signifiant qu'elles n'existent pas dans le système.

Rapportons-nous à notre exemple du chapitre et considérons les tables 4.6 et 4.7. Considérons $Obs_1 = \mathcal{V}(V_O)$, l'ensemble des observations obtenues par valuation complète des variables de V_O, afin d'illustrer nos précédents propos.

e	$effets(C_{ex}, e)$
e_1	$\{\{o_1\}, \{o_1, o_2\}\}$
e_2	$\{\{o_2\}, \{o_2, o_3\}\}$
e_3	$\{\{o_2, o_3\}\}$
e_4	$\{\{o_3, o_4\}\}$
e_5	$\{\{o_3, o_4\}\}$
e_6	$\{\{o_3, o_4\}\}$
e_7	$\{\{\}\}$

TABLE 4.6 – Effets dans C_{ex}

obs	$Sequences(A_{ex}, C_{obs})$
$\{\}$	$\{\epsilon, e_7\}$
$\{o_1\}$	$\{e_1\}$
$\{o_2\}$	$\{e_2\}$
$\{o_1, o_2\}$	$\{e_2e_1\}$
$\{o_2, o_3\}$	$\{e_3, e_3e_2\}$
$\{o_3, o_4\}$	$\{e_4, e_5, e_6\}$

TABLE 4.7 – Séquences de A_{ex} pour Obs_{ex}

La table 4.6 représente les effets des événements dans C_{ex}. La table 4.7 représente elle les séquences qui provoquent certaines observations. Les observations qui apparaissent dans cette dernière table sont exactement celles qui existent dans le système et qui correspondent donc à l'ensemble $Obs_{ex} = C_{ex}[V_O]$. Notons que l'observation $\{o_1\}$ correspond à la formule booléenne $o_1 \wedge \overline{o_2} \wedge \overline{o_3} \wedge \overline{o_4}$.

Un effet d'un événement et une observation, telle que nous l'avons considérée, correspondent tous deux à la partie observable d'une configuration. Les événements qui apparaissent dans l'une et l'autre des tables sont issus de l'ensemble des événements du système E_{ex}.

On peut inverser la table des séquences et noter les observations générées en fonction de la séquence considérée. La table résultante est présentée en table 4.8.

Nous obtenons alors exactement pour chaque séquence l'ensemble des effets qu'elle génère. Et il s'agit d'effets dans le sens que nous avons donné en définition 15, sauf qu'au lieu de considérer des événements, on considère des séquences d'événements. Plus exactement, au lieu de considérer l'événement qui précède l'effet, on considère la séquence qui le produit.

seq	$\mathcal{P}(obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{o_1\}\}$
e_2e_1	$\{\{o_1, o_2\}\}$
e_2	$\{\{o_2, o_3\}\}$
e_3e_2	$\{\{o_2\}\}$
e_3	$\{\{o_2, o_3\}\}$
e_4	$\{\{o_3, o_4\}\}$
e_5	$\{\{o_3, o_4\}\}$
e_6	$\{\{o_3, o_4\}\}$
e_7	$\{\{\}\}$

TABLE 4.8 – Observations générées par les séquences

Dans ce cadre, la notion de séquences est plus précise pour expliquer ce qui implique un effet. Elle donne non seulement l'événement qui le précède mais aussi le contexte (la séquence en amont) dans lequel l'événement arrive. Il apparaît donc évident que la notion de séquence apporte plus d'informations et couvre même la notion d'effets comme nous l'avons introduite dans ce document.

Nous voyons facilement que pour retomber sur la notion d'effets, il suffit de grouper les séquences selon leur dernier événement. Cela apparaît clairement dans le cas de l'exemple sur la table 4.9.

e	$\mathcal{P}(obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{o_1\}, \{o_1, o_2\}\}$
e_2	$\{\{o_2\}, \{o_2, o_3\}\}$
e_3	$\{\{o_2, o_3\}\}$
e_4	$\{\{o_3, o_4\}\}$
e_5	$\{\{o_3, o_4\}\}$
e_6	$\{\{o_3, o_4\}\}$
e_7	$\{\{\}\}$

où ϵ correspond à l'événement virtuel e_I que nous introduisons en préambule de la définition 28.

TABLE 4.9 – Observations générées par les événements

Les métriques de détectabilité et de diagnosticabilité peuvent donc être calculées de la même façon à partir de la table des séquences (définie comme ci-dessus) qu'à partir du modèle du système. À partir des séquences, nous n'avons donc pas besoin d'utiliser les notions de sur-effets et sous-effets et d'encadrer les métriques de performances de diagnostic puisque nous sommes capables de reconstruire exactement l'ensemble des effets des événements du système.

4.4.2.3 Coupes

Un autre moyen qui permet d'obtenir des relations de causes à effets est l'exploitation des coupes de défaillances. Il est intéressant de regarder ce qu'il est possible de dire sur les performances du système sous diagnostic en s'appuyant sur les coupes.

4.4.2.3.1 Coupes et effets

Les coupes listent, pour une observation donnée, l'ensemble des groupes d'événements qui ont pu la provoquer. Les groupes d'événements dont il est question correspondent aux événements qui apparaissent dans les séquences de défaillances. Les coupes perdent donc deux informations par rapport aux séquences :

- L'ordre : on ne sait pas dans quel ordre les événements sont arrivés.

- Le nombre d'occurrences : on ne considère les événements qu'une seule fois, quel que soit le nombre de fois qu'ils apparaissent dans les séquences.

Nous nous demandons dans cette partie s'il est possible d'évaluer le système, vis-à-vis de ses performances de diagnostic telles que nous les avons définies dans ce chapitre, en utilisant la notion de coupes.

Tout comme pour les séquences, l'ensemble des coupes sera calculé pour chacune des valuations de variables d'observation possible.

Pour imager les propos qui vont suivre, nous rappelons l'ensemble des coupes pour les observations de Obs_{ex} en table 4.10, qui correspond exactement à l'ensemble des coupes pour toutes les observations totalement valuées, où les observations pour lesquelles aucune coupe n'est calculée n'apparaissent pas.

obs	$Cuts(A_{ex}, C_{obs})$
$\{\}$	$\{\{\}, \{e_7\}\}$
$\{o_1\}$	$\{\{e_1\}\}$
$\{o_2\}$	$\{\{e_2\}\}$
$\{o_1, o_2\}$	$\{\{e_1, e_2\}\}$
$\{o_2, o_3\}$	$\{\{e_3\}, \{e_2, e_3\}\}$
$\{o_3, o_4\}$	$\{\{e_4\}, \{e_5\}, \{e_6\}\}$

TABLE 4.10 – Coupes de A_{ex} pour Obs_{ex}

Une première idée est, comme nous l'avons fait pour les séquences, d'associer les observations à un événement, dans le but que celles-ci se rapprochent des effets des événements. Si nous inversons la table des coupes, nous obtenons une association entre un ensemble d'événements (une coupe) et un ensemble d'observations. La table 4.11 est le résultat de cette opération pour notre exemple.

cut	$\mathcal{P}(Obs)$
$\{\}$	$\{\{\}\}$
$\{e_1\}$	$\{\{o_1\}\}$
$\{e_1, e_2\}$	$\{\{o_1, o_2\}\}$
$\{e_2\}$	$\{\{o_2\}\}$
$\{e_2, e_3\}$	$\{\{o_2, o_3\}\}$
$\{e_3\}$	$\{\{o_2, o_3\}\}$
$\{e_4\}$	$\{\{o_3, o_4\}\}$
$\{e_5\}$	$\{\{o_3, o_4\}\}$
$\{e_6\}$	$\{\{o_3, o_4\}\}$
$\{e_7\}$	$\{\{\}\}$

TABLE 4.11 – Observations associées à une coupe

On se rend compte facilement à travers cet exemple en ayant une relation entre une coupe et un ensemble d'observations, que l'on ne sait pas quelles séquences exactement ont amené à ces observations. Ceci est dû à la notion d'ensemble qui perd l'ordre et le nombre d'occurrences des événements.

Nous pouvons affirmer que pour une coupe donnée et pour chaque élément de l'ensemble d'observations associé, au moins un des événements de la coupe a comme effet l'observation considérée ("avoir comme effet l'observation" au sens où nous l'avons défini signifie "est le dernier événement de la séquence provoquant l'observation"), mais nous ne pouvons pas affirmer lequel exactement. Considérons la table 4.11. Pour la coupe $\{e_1, e_2\}$, nous avons une seule observation générée $\{o_1, o_2\}$. Dans ce cas, nous n'avons pas assez d'informations pour décider si c'est la séquence e_1e_2 ou la séquence e_2e_1 ou tout autre séquence construite à partir de e_1 et e_2 (ou plusieurs d'entre elles) qui génère(nt) l'observation.

Nous comprenons ainsi que nous n'avons pas assez d'informations pour couvrir la notion d'effets telle que nous l'avons définie, et pour que les définitions et métriques qui en découlent puissent être applicables en l'état.

Nous pouvons néanmoins essayer de regarder s'il est possible d'obtenir des sur-effets ou sous-effets à partir des coupes, afin de pouvoir encadrer les métriques de détectabilité et de diagnosticabilité. Pour cela, nous allons nous intéresser à deux objets différents calculés à partir de la relation qui associe un ensemble d'observations à une coupe (table 4.11). Le premier objet jouera le rôle de sur-effets, et le deuxième celui de sous-effets.

Le premier objet représentant les sur-effets pour les coupes doit donc contenir plus d'effets qu'il n'y en a réellement dans le système. Pour calculer un tel objet à partir de la relation qui associe les observations à chaque coupe, il suffit de considérer, quand on ne sait pas quel événement de la coupe est réellement celui qui a provoqué l'observation (i.e. lorsque la coupe n'est pas réduite à un seul événement), que tous les événements de cette coupe peuvent le provoquer. Ainsi, nous associons potentiellement des événements à des observations qu'ils ne peuvent pas causer, mais en revanche, nous sommes sûrs que tous les effets des événements sont présents.

Afin de calculer l'objet représentant les sous-effets pour les coupes, nous appliquons un principe similaire. Lorsque nous rencontrons des cas pour lesquels il n'est pas possible d'affirmer quel événement a provoqué une observation de manière sûre, nous considérons que l'observation n'est associée à aucun de ces événements. Ainsi, nous ne conservons que l'ensemble des observations dont nous sommes certains qu'elles correspondent à des effets dans le système.

Pour résumer ces derniers paragraphes, nous pouvons dire qu'à partir des coupes, il est possible d'obtenir à la fois un objet représentant les sur-effets des événements du modèle et un objet représentant leurs sous-effets.

Nous représentons, pour notre exemple, en tables 4.12 et 4.13, les sur-effets et sous-effets des événements pour les coupes obtenues à partir de la table 4.11.

e	$\mathcal{P}(Obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{o_1\},\{o_1,o_2\}\}$
e_2	$\{\{o_2\},\{o_1,o_2\},\{o_2,o_3\}\}$
e_3	$\{\{o_2,o_3\}\}$
e_4	$\{\{o_3,o_4\}\}$
e_5	$\{\{o_3,o_4\}\}$
e_6	$\{\{o_3,o_4\}\}$
e_7	$\{\{\}\}$

TABLE 4.12 – Sur-effets dans C_{ex}

e	$\mathcal{P}(Obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{o_1\}\}$
e_2	$\{\{o_2\}\}$
e_3	$\{\{o_2,o_3\}\}$
e_4	$\{\{o_3,o_4\}\}$
e_5	$\{\{o_3,o_4\}\}$
e_6	$\{\{o_3,o_4\}\}$
e_7	$\{\{\}\}$

TABLE 4.13 – Sous-effets dans C_{ex}

Prenons l'exemple de l'association *coupe–observation* $\{e_1,e_2\} - \{o_1,o_2\}$ dans la table 4.11. Dans le cas des sur-effets, étant donné qu'il y a un doute sur la cause réelle de l'observation, nous ajoutons l'association *événement–observation* pour les deux événements. Pour les sous-effets, dans ce même cas, nous n'ajoutons aucune association *événement–observation* du fait de l'incertitude qu'il existe. Finalement dans la relation d'effets, nous voyons que la réalité est entre les deux, c'est e_1 qui génère l'observation $\{o_1,o_2\}$.

Nous pouvons faire la remarque, en comparant ces tables et celle correspondant aux effets des événements dans C_{ex} (rappelée en table 4.6 page 115), que nous avons bien sous-effets $\subseteq$ effets $\subseteq$ sur-effets. Cela signifie que nous retrouvons toutes les associations *événement–observation* des sous-effets dans les effets, et toutes celles des effets dans les sur-effets.

Nous pouvons finalement représenter sur la figure 4.5 la relation entre effets, sur-effets et sous-effets.

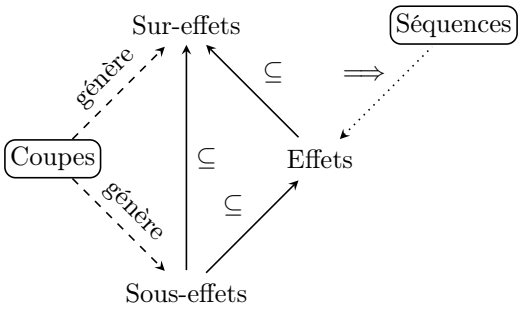


FIGURE 4.5 – Effets, sur-effets et sous-effets (coupes et séquences)

Remarque *La relation d'inclusion est transitive. Tous les arcs ne sont pas représentés sur le graphe dans un soucis de visibilité. De même nous n'avons pas représenté les arcs entrant au niveau des séquences, qui sont les mêmes que pour les effets.*

Nous avons fait apparaître les séquences sur la figure pour exprimer le fait qu'à partir des séquences, nous pouvons obtenir exactement les effets (ce qui n'est pas vrai dans l'autre sens).

4.4.2.3.2 Illustrations des métriques de performances

Dans cette partie nous allons illustrer l'utilisation des sur-effets et sous-effets calculés à partir des coupes pour caractériser les propriétés de détectabilité et de diagnosticabilité. Nous verrons des exemples qui vérifient les propriétés énoncées en section 4.4.1.

4.4.2.3.2.1 Détectabilité forte

Illustrons ce que nous avons introduit en section 4.4.1.1 : *Si aucun des sur-effets d'un événement ne correspond à un effet de bon fonctionnement, alors il est fortement détectable.* L'implication inverse n'est pas vraie.

Considérons notre exemple à travers la table des sur-effets 4.12. Rappelons que $Obs_{OK_{ex}} = \{\{\}, \{o_1, o_2\}\}$. Prenons l'événement e_3 que l'on sait fortement détectable dans C_{ex} (cf exemple en déf. 18 page 90). L'ensemble des observations associé à e_3 dans les sur-effets est $\{\{o_2, o_3\}\}$. Nous avons donc ici l'exemple d'un événement dont l'ensemble des observations associées n'intersecte pas $Obs_{OK_{ex}}$, et qui est fortement détectable.

Regardons maintenant l'événement e_2. Les observations auxquelles est associé e_2 dans les sur-effets sont $\{o_2\}$, $\{o_1, o_2\}$ et $\{o_2, o_3\}$, et l'une d'entre elles appartient à $Obs_{OK_{ex}}$. Nous ne pouvons donc pas déduire à partir des sur-effets que e_2 est fortement détectable dans C_{ex} bien qu'il le soit en réalité. Cela illustre le fait que l'implication dans l'autre sens n'est donc pas vraie.

4.4.2.3.2.2 Non détectabilité

Illustrons ce que nous avons introduit en section 4.4.1.2 : *Si chacun des sur-effets d'un événement correspond à un effet de bon fonctionnement, alors il est non détectable.* L'implication inverse n'est pas vraie.

Si nous revenons sur notre exemple nous pouvons voir que l'événement e_7, non détectable dans notre système, est typiquement dans le cas où les observations qui lui sont associées (ici uniquement $\{\}$) sont toutes éléments de l'ensemble des observations de bon fonctionnement.

Dans notre exemple nous n'avons pas de cas d'événement non détectable dont toutes les observations associées ne sont pas dans $Obs_{OK_{ex}}$. Nous allons donc introduire un petit exemple ponctuel pour illustrer cette propriété. Pour cela, nous proposons la figure 4.6 représentant le système sous diagnostic α_{ex_1}.

Les coupes de ce système sous diagnostic sont présentées en table 4.14.

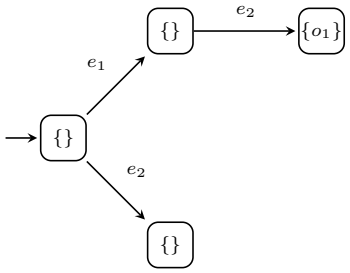


FIGURE 4.6 – SUD exemple α_{ex_1} (réduit aux variables d'observation)

obs	$Cuts(A_{ex}, C_{obs})$
$\{\}$	$\{\{\}, \{e_1\}, \{e_2\}\}$
$\{o_1\}$	$\{\{e_1, e_2\}\}$

TABLE 4.14 – Coupes du SUD α_{ex_1}

Nous pouvons inverser la table des coupes pour obtenir les observations associées à chaque coupe pour ce SUD exemple. La table 4.15 est le résultat de cette opération.

cut	$\mathcal{P}(Obs)$
$\{\}$	$\{\{\}\}$
$\{e_1\}$	$\{\{\}\}$
$\{e_2\}$	$\{\{\}\}$
$\{e_1, e_2\}$	$\{\{o_1\}\}$

TABLE 4.15 – Observations associées à une coupe pour le SUD α_{ex_1}

Finalement, nous obtenons la relation de sur-effets en table 4.16.

e	$\mathcal{P}(Obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{\}, \{o_1\}\}$
e_2	$\{\{\}, \{o_1\}\}$

TABLE 4.16 – Sur-effets pour α_{ex_1}

L'événement e_1 a comme observations associées $\{\}$ et $\{o_1\}$. Avec $Obs_{OK} = \{\{\}\}$ pour cet exemple, nous avons le cas d'un événement e_1, non détectable, dont une des observations associées ($\{o_1\}$) n'est pas une observation de bon fonctionnement. Cela est dû au fait que l'association e_1–$\{o_1\}$ dans les sur-effets n'est pas un effet réel du système.

4.4.2.3.2.3 Détectabilité faible

Illustrons ce que nous avons introduit en section 4.4.1.3 : *Si un événement est à la fois associé à un sous-effet qui est une observation de bon fonctionnement, et à un autre sous-effet qui n'en est pas une, alors il est faiblement détectable.* L'implication inverse n'est pas vraie.

Prenons l'exemple de la figure 4.7 représentant la partie observable d'un système sous diagnostic.

Les coupes de ce système sous diagnostic sont présentées en table 4.17.

L'événement e_1 a comme sous-effets pour les coupes les observations $\{\}$ et $\{o_1\}$. Si $Obs_{OK} = \{\{\}\}$, on peut en déduire que l'événement est faiblement détectable.

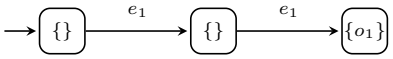


FIGURE 4.7 – SUD exemple α_{ex_2} (réduit aux variables d'observation)

obs	$Cuts(A_{ex}, C_{obs})$
$\{\}$	$\{\{\}, \{e_1\}\}$
$\{o_1\}$	$\{\{e_1\}\}$

TABLE 4.17 – Coupes du SUD α_{ex_2}

Si nous considérons maintenant l'exemple du SUD A_{ex} et que nous nous reportons à la table des sous-effets 4.13, nous remarquons que l'événement e_1, que nous savons faiblement détectable dans C_{ex} (cf. exemple en déf. 19 page 91), a comme sous-effets uniquement $\{o_1\}$. L'implication inverse ne se vérifie pas à cause de l'absence de l'effet $\{\}$ pour e_1 dans notre cas.

4.4.2.3.2.4 Détectabilité

Illustrons ce que nous avons introduit en section 4.4.1.4 : *Si un événement est associé à un effet qui n'est pas de bon fonctionnement, alors il est détectable.*. L'implication inverse n'est pas vraie. La deuxième méthode d'identification des événements détectables étant la même que pour la détectabilité forte, l'illustration pour la détectabilité forte est aussi valable pour la détectabilité.

La première règle définie pour la détectabilité couvrant celle de détectabilité faible, nous pouvons reprendre l'exemple d'illustration de la règle de détectabilité faible pour illustrer notre règle, à savoir le cas de l'événement e_1 dans le système α_{ex_2}.

Reprenons l'exemple α_{ex_1} que nous avons proposé en figure 4.6 page 120. Les sous-effets obtenus (à partir de la table 4.15 page 120) sont présentés en figure 4.18. Étant donné qu'$Obs_{OK} = \{\{\}\}$

e	$\mathcal{P}(Obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{\}\}$
e_2	$\{\{\}\}$

TABLE 4.18 – Sous-effets pour α_{ex_1}

dans ce cas, nous voyons que le seul sous-effet associé à e_2 ($\{\}$) ne permet d'identifier e_2 comme étant détectable, alors qu'il l'est en réalité puisqu'il provoque aussi l'effet $\{o_1\}$.

4.4.2.3.2.5 Présence dans le groupe d'ambiguité

Illustrons ce que nous avons introduit en section 4.4.1.5 : *Si deux événements partagent au moins une observation à laquelle ils sont associés dans les sous-effets, alors ils appartiennent chacun au groupe d'ambiguité de l'autre.* L'implication inverse n'est pas vraie.

Considérons notre système A_{ex} et la table des sous-effets correspondante 4.13. Les événements e_4, e_5, e_6 sont une illustration de la règle énoncée. Ils sont tous associés à l'observation $\{o_3, o_4\}$ dans la table des sous-effets, et sont aussi ambigus dans le système (cf. exemple de la définition 28).

En revanche, si nous considérons les événements e_2 et e_3 que nous savons ambigus, nous remarquons que dans la table des sous-effets ils ne partagent pas d'observation. C'est parce que l'observation $\{o_1, o_2\}$, qu'ils génèrent tous les deux, a disparu de cette table.

4.4.2.3.2.6 Absence dans le groupe d'ambiguité

Illustrons ce que nous avons introduit en section 4.4.1.6 : *Si deux événements ne partagent aucun sur-effet, alors aucun des deux n'apparaît dans le groupe d'ambiguité de l'autre.* L'implication inverse n'est pas vraie.

Nous pouvons utiliser l'exemple du chapitre pour illustrer cette règle. Référons-nous donc à la table des sur-effets 4.12. Nous observons par exemple que les ensembles d'observations associés aux événements e_1, e_3, e_4 et e_7 sont tous disjoints et remarquons qu'en effet tous ces événements sont fortement discriminables entre eux (cf. table 4.1). Ceci est une illustration de la règle rappelée ci-dessus.

En revanche, nous observons que les ensembles d'observations associés aux événements e_1 et e_2 ne sont pas disjoints puisqu'ils partagent l'observation $\{o_1, o_2\}$. Or en réalité, ces deux événements sont bien fortement discriminables. Ceci est une fois de plus dû à la présence de $\{o_1, o_2\}$, qui n'est pas un effet de e_1 parmi les observations qui lui sont associées. Ce cas infirme bien l'implication inverse de la règle.

4.4.2.4 Séquences et coupes minimales

Nous nous intéressons dans cette section aux versions minimales des objets précédemment calculés que sont les coupes et les séquences. Il est intéressant de regarder ces versions minimales car elles peuvent notamment être plus faciles à obtenir d'un point de vue calculatoire.

Le fait de minimiser les séquences et les coupes implique que des effets peuvent disparaître des tables. Cela signifie que les observations associées aux séquences ou coupes d'événements ne sont pas complètes et que tous les effets ne sont assurés d'être présents dans la table. Une notion pertinente de sur-effets (autre que celle considérant toutes les observations possibles) est donc impossible à obtenir dans ce cadre.

4.4.2.4.1 Coupes minimales

La minimisation a un double effet pour les coupes. Tout d'abord, elle invalide la notion de sur-effets puisqu'il est impossible de s'assurer que l'on a tous les effets des événements du système. Ensuite, elle rend moins précise (par rapport aux coupes) la notion de sous-effets, des associations *événement – effets* présentes avec les coupes ayant pu être minimisées. Les coupes minimales formant un sous-ensemble des coupes, la notion de sous-effets est toujours valable et est obtenue de la même façon que pour les coupes.

Pour essayer d'évaluer la détectabilité et la diagnosticabilité du système, nous n'avons donc à notre disposition que la notion de sous-effets. Les règles basées sur l'analyse des sur-effets (la détectabilité forte, la non détectabilité, une partie de la détectabilité et l'absence du groupe d'ambiguité) sont donc invalidées.

Dans la suite de cette partie, nous allons donner des exemples illustrant d'une part que les règles dépendant des sur-effets ne sont plus valables et d'autre part que celles utilisant les sous-effets fonctionnent (comme nous l'avons fait en 4.4.2.3.2) toujours mais qu'elles sont moins précises que dans le cas des coupes.

4.4.2.4.1.1 Détectabilité forte

Considérons la règle affirmant qu'un événement est fortement détectable pour les coupes. Nous proposons un cas problématique très simple en figure 4.8.

Les coupes minimales obtenues à partir de ce système sont exprimées en table 4.19.

Considérons que pour cet exemple $Obs_{OK} = \{\{\}\}$. On voit qu'en se rapportant à la règle d'identification d'événements fortement détectables à partir des sur-effets, on considérerait que l'événement e_1 est fortement détectable, puisqu'il n'est associé qu'à l'observation $\{o_1\}$ qui n'est pas une observation de Obs_{OK}. Or ici, un de ses effets qui correspond à un effet de bon fonctionnement (l'effet $\{\{\}\}$) a été minimisé.

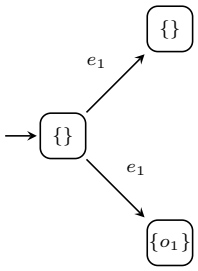


FIGURE 4.8 – SUD exemple α_{ex_4} (réduit aux variables d'observation)

obs	*MinCuts*
$\{\}$	$\{\{\}\}$
$\{o_1\}$	$\{\{e_1\}\}$

TABLE 4.19 – Coupes minimales pour α_{ex_4}

4.4.2.4.1.2 Non détectabilité

Pour la règle qui permet d'affirmer qu'un événement est non détectable, le problème est similaire. Regardons le système sous diagnostic exemple représenté en figure 4.9 qui illustre bien ce problème.

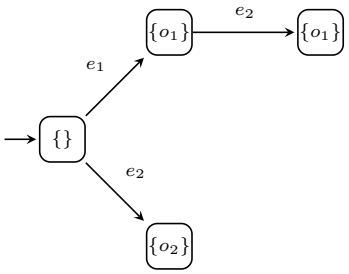


FIGURE 4.9 – SUD exemple α_{ex_5} (réduit aux variables d'observation)

Les coupes minimales obtenues à partir de ce système sont exprimées en table 4.20.

obs	*MinCuts*
$\{\}$	$\{\{\}\}$
$\{o_1\}$	$\{\{e_1\}\}$
$\{o_2\}$	$\{\{e_2\}\}$

TABLE 4.20 – Coupes minimales pour α_{ex_5}

Considérons que l'ensemble des observations de bon fonctionnement est $\{\{\}, \{o_2\}\}$. En utilisant la règle identifiant des événements non détectables, on se retrouverait à dire ici que l'événement e_2 est non détectable alors qu'en réalité il l'est faiblement car il a aussi pour effet $\{o_1\}$. Seulement cet effet est supprimé pour e_2 du fait de la minimisation puisque la coupe $\{e_1\}$ le provoque déjà.

4.4.2.4.1.3 Détectabilité faible

Si les deux règles précédentes concernant la détectabilité sont invalidées par la minimisation, ce n'est pas le cas de la règle identifiant une partie des événements faiblement détectables. Prenons un exemple simple pour illustrer cela. Le SUD exemple α_{ex_6} a sa partie observable présentée en figure 4.10.

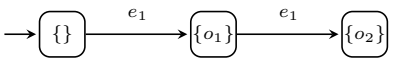


FIGURE 4.10 – SUD exemple α_{ex_6} (réduit aux variables d'observation)

Les coupes minimales et les sous-effets obtenus pour ce système sont présentés en tables 4.21 et 4.22.

obs	*MinCuts*
$\{\}$	$\{\{\}\}$
$\{o_1\}$	$\{\{e_1\}\}$
$\{o_2\}$	$\{\{e_1\}\}$

TABLE 4.21 – Coupes min. de α_{ex_6}

e	$\mathcal{P}(Obs)$
ϵ	$\{\{\}\}$
e_1	$\{\{o_1\},\{o_2\}\}$

TABLE 4.22 – Sous-effets (cm) pour α_{ex_6}

En considérant que $Obs_{OK} = \{\{\},\{o_2\}\}$, nous trouvons bien que l'événement e_1 est faiblement détectable grâce à la règle définie sur les coupes. Il a en effet pour sous-effets à la fois une observation de bon fonctionnement ($\{o_2\}$) et une observation de défaillance ($\{o_1\}$).

Il est évident que l'implication inverse n'est toujours pas vraie, et l'exemple donné pour les coupes (exemple de l'événement e_1 dans A_{ex}) est toujours valable pour les coupes minimales. Mais nous pouvons illustrer le fait que la notion de sous-effets est moins précise pour les coupes minimales que pour les coupes en reprenant l'exemple α_{ex_2} présenté en figure 4.7 page 121. Avec cet exemple, nous étions capable de déterminer que e_1 était faiblement détectable à partir des coupes. Avec les coupes minimales, l'observation $\{\{\}\}$ ne fait plus partie des sous-effets (car e_1 est minimisé par ϵ sur cette observation) et nous ne pouvons plus appliquer la règle.

4.4.2.4.1.4 Détectabilité

La seconde méthode permettant d'identifier des événements détectables n'est plus valide car elle s'appuie sur les sur-effets. Comme elle correspond à la méthode d'identification des événements fortement détectables, le cas problématique présenté en 4.4.2.4.1.1 s'applique aussi à cette règle.

La première méthode proposée pour la détectabilité s'appuie elle sur les sous-effet. Nous pouvons donc toujours l'utiliser. De manière logique, l'exemple d'illustration présenté pour la détectabilité faible (en 4.4.2.4.1.3) s'applique de la même manière pour la règle de détectabilité (car elle couvre celle de détectabilité faible).

Bien que la règle utilisée pour identifier des événements détectables soit tout autant valable dans le cadre des coupes que dans le cadre des coupes minimales (puisqu'elle s'appuie sur la notion de sous-effets), elle est moins précise dans ce dernier cas. Considérons l'exemple α_{ex_5} (fig. 4.9 page 123) introduit pour la non détectabilité. L'ensemble des coupes pour cet exemple et la table de sous-effets pour ces coupes sont respectivement présentées en tables 4.23 et 4.24.

obs	*Cuts*
$\{\}$	$\{\{\}\}$
$\{o_1\}$	$\{\{e_1\},\{e_2\}\}$
$\{o_2\}$	$\{\{e_2\}\}$

TABLE 4.23 – Coupes de α_{ex_5}

e	$\mathcal{P}(Obs)$
ϵ	$\{\}$
e_1	$\{\{o_1\}\}$
e_2	$\{\{o_1\},\{o_2\}\}$

TABLE 4.24 – Sous-effets pour les coupes de α_{ex_5}

La table des sous-effets calculés à partir des coupes minimales (présentées en table 4.20) est présentée en table 4.25.

e	$\mathcal{P}(Obs)$
ϵ	$\{\}$
e_1	$\{\{o_1\}\}$
e_2	$\{\{o_2\}\}$

TABLE 4.25 – Sous-effets pour les coupes min. de α_{ex_5}

Rappelons que $Obs_{OK} = \{\{\}, \{o_2\}\}$. Nous voyons que nous ne pouvons pas affirmer avec les sous-effets pour les coupes minimales que l'événement e_2 est détectable, alors que c'est tout à fait possible avec les sous-effets obtenus à partir des coupes. Ceci est dû à la minimisation de l'événement e_2 pour l'observation $\{o_1\}$.

4.4.2.4.1.5 Présence dans le groupe d'ambiguité

La règle qui affirme la présence d'un événement dans le groupe d'ambiguité d'un autre est toujours valide après minimisation des coupes car elle est basée sur la notion de sous-effets.

L'exemple de A_{ex} des événements e_4, e_5 et e_6 est toujours valable pour illustrer l'utilisation de cette règle : les trois événements sont toujours associés à la même observation $\{o_3, o_4\}$ dans la table des sous-effets pour les coupes minimales, et sont donc ambigus.

Pour illustrer à la fois que l'implication inverse n'est pas vraie et que la règle est moins efficace que pour les coupes, nous considérons l'exemple α_{ex_5} décrit en figure 4.9, ainsi que les tables 4.23, 4.24 et 4.25. Nous voyons que l'événement e_2, que l'on peut bien identifier comme appartenant au groupe d'ambiguité de e_1 dans le cas des coupes, ne peut pas l'être à partir des sous-effets calculés depuis les coupes minimales. Cela invalide donc l'implication inverse de la règle et illustre le fait que les sous-effets pour les coupes sont plus précis que ceux pour les coupes minimales.

4.4.2.4.1.6 Absence dans le groupe d'ambiguité

Finalement on peut s'intéresser à la règle affirmant qu'un événement n'appartient pas au groupe d'ambiguité d'un autre. Le raisonnement pour ce type de caractéristique nécessite que l'on soit sûr que tous les effets sont pris en compte. C'est pourquoi nous nous sommes appuyés sur la notion de sur-effets pour traiter cela pour les coupes. Mais la minimisation supprimant potentiellement des effets parmi les sur-effets, ces derniers ne sont pas forcément complets et nous ne pouvons plus raisonner dessus.

Considérons un exemple pour illustrer l'invalidité de la règle. Reprenons le système sous diagnostic présenté précédemment en figure 4.9 ainsi que ses sous-effets pour les coupes minimales associées en table 4.25.

Les deux ensembles d'observations associés à e_1 ($\{\{o_1\}\}$) et à e_2 ($\{\{o_2\}\}$) sont disjoints. Pourtant, les événements ne sont pas fortement discriminables car ils partagent l'effet $\{o_1\}$. Ceci est une nouvelle fois dû à la disparition de l'effet $\{o_1\}$ pour e_2 dans la table.

4.4.2.4.1.7 Synthèse

Dans cette partie, nous avons considéré les coupes minimales associées à un système sous diagnostic et, à partir d'elles, nous avons donné des informations concernant la détectabilité et la diagnosticabilité du système.

Nous avons vu que la minimisation peut enlever des effets, ce qui rend impossible le calcul de sur-effets pertinents dans ce contexte. Toutes les règles utilisant la propriété assurant que tous les effets sont présents dans les sur-effets deviennent invalides pour les coupes minimales.

Pour la détectabilité, nous conservons les possibilités d'affirmer qu'un événement est faiblement détectable ainsi que détectabilité. En revanche, il n'existe plus de règle pour identifier les événements fortement détectables et non détectables.

Pour la diagnosticabilité, nous pouvons toujours affirmer qu'un événement appartient au groupe d'ambiguité d'un autre, mais nous ne pouvons plus dire qu'il n'en fait pas partie.

Nous pouvons reprendre la figure 4.5 et y intégrer la notion de sous-effets calculés à partir de la table des coupes minimales à travers la figure 4.11.

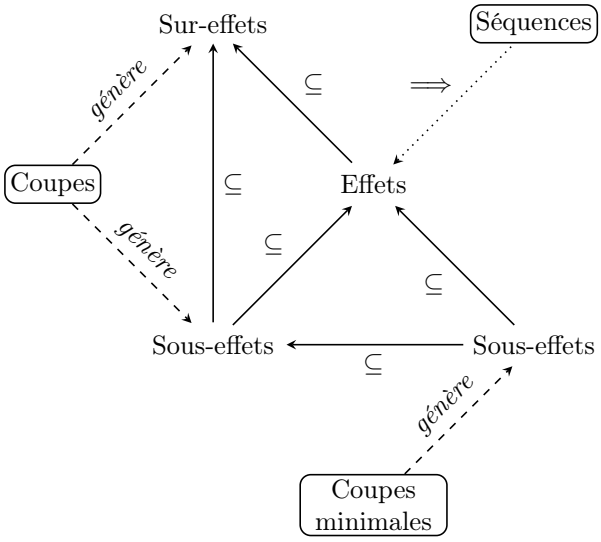


FIGURE 4.11 – Effets, sur-effets et sous-effets (coupes [minimales] et séquences)

Dans ce schéma, nous avons ajouté une information. Elle caractérise le fait que les sous-effets calculés à partir des coupes minimales sont moins précis que ceux calculés à partir des coupes d'origine. Cela est dû au fait que certaines associations *événement–observation* sont potentiellement perdues pendant la minimisation.

4.4.2.4.2 Séquences minimales

Dans la section 4.4.2.2, nous avons vu que les séquences contiennent toutes les informations nécessaires, et même plus, pour couvrir l'ensemble des définitions et métriques introduites qui dépendent des notions de détectabilité et de diagnosticabilité. Nous avons donc avec l'ensemble des séquences tous les effets des événements, et pas plus (pas d'observation supplémentaire qui ne correspond pas à un effet). C'est d'ailleurs cette dernière caractéristique qui distingue les séquences des coupes, ceci étant dû à la perte de l'ordre sur les coupes.

Raisonnons maintenant sur les séquences minimales. Nous avons vu dans la section 4.4.2.4.1 qui précède que le fait d'appliquer une minimisation peut entraîner la perte de certains effets dans la table. Cela vaut de la même manière pour les séquences. Il n'est donc plus possible de raisonner sur le fait d'avoir tous les effets des événements dans la table (sur-effets) de manière pertinente (sans considérer toutes les observations possibles).

Pour les séquences minimales, comme pour les coupes minimales, il nous reste la caractéristique de l'exactitude des observations présentes dans la table : chaque observation associée à un événement correspond à un effet réel de cet événement.

Nous obtenons alors des sous-effets de la même façon que pour obtenir les effets à partir des séquences (cf. tables 4.8 et 4.9 page 116), i.e. en ne conservant que le dernier événement de chaque séquence minimale et en exprimant pour chaque événement à quelles observations il est associé dans les séquences minimales. Nous n'avons que des associations *événement–observation* qui correspondent à des effets réels, mais nous ne les avons pas forcément tous à cause de la minimisation.

Comme nous ne pouvons pas obtenir une relation de sur-effets pertinente, nous ne regarderons pas les propriétés de détectabilité et de diagnosticabilité basées uniquement sur les sur-effets. Cela vaut avec les séquences minimales pour les règles décrivant que :

- Un événement est fortement détectable (impossible d'affirmer qu'un événement n'a aucun effet qui est de bon fonctionnement puisque la table ne contient potentiellement pas tous ses effets).
- Un événement est non détectable (impossible d'affirmer qu'un événement n'a que des effets qui sont dans Obs_{OK} puisque la table ne contient potentiellement pas tous ses effets).
- Un événement n'est pas dans le groupe d'ambiguité d'un autre (impossible de s'assurer que les événements n'ont pas perdu un effet commun dans les tables à cause de la minimisation).

Les exemples d'illustration présentés en 4.4.2.4.1.1, 4.4.2.4.1.2 et 4.4.2.4.1.6 pour les coupes minimales sont tout aussi appropriés pour montrer l'invalidité de ces règles pour les séquences minimales.

Il nous reste donc trois caractéristiques pour lesquelles nous pouvons exploiter les séquences minimales.

4.4.2.4.2.1 Détectabilité faible

La règle permettant d'affirmer qu'un événement est faiblement détectable s'appuie sur la notion de sous-effets et est donc toujours valable dans le cas des séquences minimales.
Illustrons la règle avec l'exemple représenté en figure 4.12.

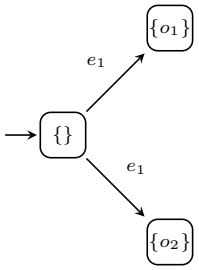


FIGURE 4.12 – SUD exemple α_{ex_7} (réduit aux variables d'observation)

Les séquences minimales pour le système α_{ex_7} sont représentées en table 4.26.

obs	$MinSeq$
$\{\}$	$\{\epsilon\}$
$\{o_1\}$	$\{e_1\}$
$\{o_2\}$	$\{e_1\}$

TABLE 4.26 – Séquences minimales pour α_{ex_7}

En considérant que $Obs_{OK} = \{\{\}, \{o_1\}\}$, nous voyons, e_1 étant associé aux observations $\{o_1\}$ et $\{o_2\}$, que e_1 est faiblement détectable. Ceci est une illustration de l'application de la règle.
Néanmoins, dans l'exemple du système présenté en figure 4.9, l'événement e_2 est l'exemple typique d'un événement qui ne répond pas à l'implication inverse de la règle pour la détectabilité faible.

4.4.2.4.2.2 Détectabilité

Nous ne considérons pour la détectabilité à partir des séquences minimales que la règle utilisant les sous-effets. L'illustration utilisée pour la détectabilité faible est évidemment une nouvelle fois valable pour la détectabilité.
Pour illustrer l'invalidité de l'implication inverse, nous pouvons utiliser le même exemple que nous avons utilisé pour la détectabilité dans le cadre des coupes minimales (4.4.2.4.1.4), à savoir

l'exemple α_{ex_5} (fig. 4.9 page 123). L'événement e_2 n'est pas vu comme détectable avec les séquences minimales non plus car la séquence e_1e_2 pour l'observation $\{o_1\}$ est minimisé par la séquence e_1.

4.4.2.4.2.3 Présence dans le groupe d'ambiguité

La seconde règle qui est conservée concerne l'appartenance d'un premier événement au groupe d'ambiguité d'un deuxième événement. L'exemple des événements e_4, e_5 et e_6 dans A_{ex} est une illustration de cette propriété (cf. les séquences minimales pour A_{ex} en 3.3).

Une nouvelle fois, la minimisation ayant pu supprimer des effets communs, cette règle ne couvre pas tous les événements dans ce cas. L'exemple du système présenté en figure 4.9 montre que l'on ne peut pas voir que l'événement e_2 appartient au groupe d'ambiguité de l'événement e_1 (et vice et versa) à partir des séquences minimales (la séquence e_1e_2 a été minimisée par la séquence e_1 pour l'observation $\{o_1\}$).

4.4.2.4.2.4 Synthèse

Dans cette partie nous avons montré que l'utilisation de la minimisation réduit l'analyse que l'on peut faire à partir des séquences pour caractériser la détectabilité et la diagnosticabilité d'un système à travers les définitions et métriques que nous avons définies dans le chapitre. Avec les séquences minimales, nous conservons la notion de sous-effets, et ainsi nous pouvons encore affirmer que des événements sont faiblement détectables ainsi que détectables et assurer qu'un événement appartient au groupe d'ambiguité d'un autre.

Nous pouvons reprendre la figure 4.11 et y intégrer la notion de sous-effets représentée par les séquences minimales à travers la figure 4.13.

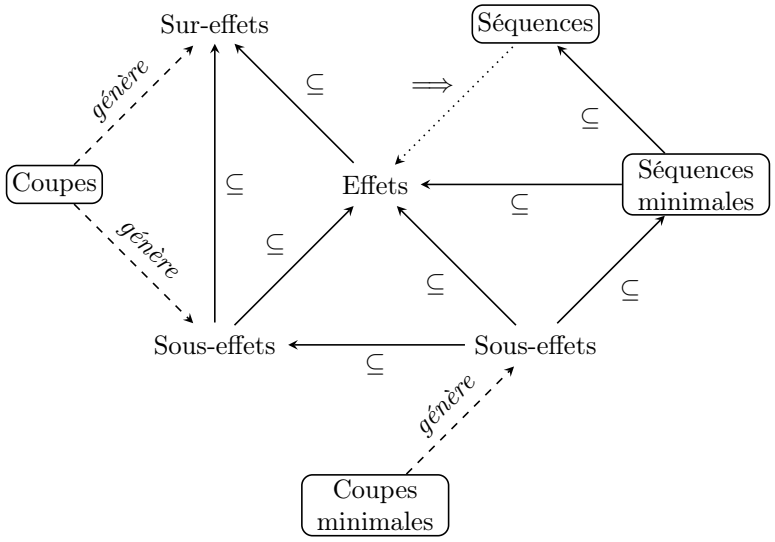


FIGURE 4.13 – Effets, sur-effets et sous-effets (coupes [minimales] et séquences [minimales])

Dans ce schéma nous avons ajouté deux informations. La première est que les séquences minimales, qui représentent des sous-effets, sont plus précis que les sous-effets des coupes minimales.

En effet, les associations *événement–observation* perdues par les séquences minimales lors de la minimisation sont obligatoirement aussi perdues par les coupes minimales. L'ordre sous-mot sur les séquences est couvert par l'ordre sous-ensemble sur les coupes dérivées des séquences.

Comme les seules informations perdues par les séquences minimales sont issues de la minimisation, et que les séquences sont par essence plus précises que les coupes, alors les séquences minimales sont plus précises que les coupes minimales.

La seconde information est que les séquences minimales en tant que sous-effets ne sont pas comparables aux sous-effets obtenus pour les coupes non minimales. Ceci est justifiée par le fait que du côté des coupes, c'est la perte de l'ordre qui fait perdre de l'information, alors que du côté des séquences minimales, c'est la minimisation. L'impact de l'une n'est pas tout le temps plus fort que l'impact de l'autre, donc on ne peut pas affirmer que l'une fait perdre moins d'information que l'autre.

4.4.2.5 Objet "dernier événement"

Nous avons vu que la relation nécessaire pour couvrir les notions importantes de détectabilité et de diagnosticabilité est la relation *événement – effets*. La quasi-totalité des définitions et métriques introduites par la suite en découlent (hors mis la notion de perceptibilité qui nécessite de connaître les effets précédant l'occurrence d'un événement).

Nous avons vu aussi que les objets que sont le système sous diagnostic réduit en profondeur, les séquences, les coupes, et leur version minimale, ne correspondent pas exactement à cette notion d'effets. Le système réduit en profondeur enlève les effets des événements dont l'occurrence a lieu après une certaine profondeur. Les coupes, elles, ajoutent des observations dans la relation qui ne correspondent pas à des effets, cela à cause de la perte de l'ordre d'occurrence des événements. Les séquences conservent pour leur part toutes les informations nécessaires à l'établissement de la relation *événement – effets*, et comporte même plus d'informations. Finalement, les versions minimisées des coupes et séquences suppriment des effets, impactant les notions de détectabilité et de diagnosticabilité comme nous l'avons vu dans la section précédente.

Nous pouvons finalement, avec l'analyse que nous avons fait des objets de coupes et de séquences, définir l'objet qui couvre exactement la relation *événement – effets*. Cet objet correspond à celui calculé pour une illustration dans la partie sur les séquences à travers la table 4.9 page 116. Il s'agit en réalité de ne conserver que le dernier événement de la séquence lorsque nous cherchons à identifier la cause d'une observation.

Nous conservons avec cet objet la notion d'ordre, mais réduite au dernier événement, et nous groupons les observations si le dernier événement est le même. Nous n'avons ainsi pas besoin de garder en mémoire l'ensemble de la séquence précédant l'occurrence du dernier événement, ce qui peut améliorer sensiblement les méthodes de calcul de cet objet par rapport à celles pour les séquences.

Nous avons donc identifié que la relation *événement – effets* est la même que la relation entre le dernier événement et les observations qu'il provoque. C'est l'objet à la base de la majorité des définitions et métriques introduites dans le chapitre, dont la méthode de calcul peut être moins coûteuse que celle pour calculer les séquences.

L'objet "dernier événement" est donc l'objet minimal à calculer si l'on veut conserver l'entière correction de la majorité de définitions et métriques (hors perceptibilité).

4.4.2.6 Synthèse

Nous proposons en table 4.27 un récapitulatif de ce qu'il est possible de dire sur les performances de diagnostic que nous avons définies dans ce chapitre à partir des différentes relations introduites dans cette section.

Dans cette section, nous n'avons pas caractérisé toutes les relations de causes à effets qu'il est possible d'obtenir à partir d'un système à événements discrets. Néanmoins en donnant les notions de sur-effets et sous-effets, nous avons donné la clé du raisonnement.

Par exemple, si nous considérons les séquences bornées ou les séquences minimales bornées, nous comprenons que borner la taille des séquences de défaillances invalide les règles liées aux sur-effets. Ainsi nous savons que seules les propriétés de détectabilité faible, de détectabilité et de présence dans le groupe d'ambiguité sont conservées, et donc à partir de là que nous pouvons donner des détectabilités faible et globale minimum ainsi qu'un degré de diagnostic moyen minimum pour le système.

Nous n'avons pas décrit dans cette partie les séquences bornées et les séquences minimales bornées car elles correspondent respectivement à l'application des séquences et séquences minimales

Relations	Séquences	Effets	Coupes	Effets bornés	Séq. min.	Coupes min.
Détectabilité forte	X	X	X			
Non détectabilité	X	X	X			
Détectabilité faible	X	X	X	X	X	X
Détectabilité	X	X	X	X	X	X
Présence dans le GA	X	X	X	X	X	X
Absence dans le GA	X	X	X			
Métriques	RDF RDf RD DDM	RDF RDf RD DDM	$RDF_{min/max}$ $RDf_{min/max}$ $RD_{min/max}$ $DDM_{min/max}$	RDf_{min} RD_{min} DDM_{min}	RDf_{min} RD_{min} DDM_{min}	RDf_{min} RD_{min} DDM_{min}

TABLE 4.27 – Comparatif des résultats de métriques pour C_{ex} et $\{c_1\}$

dans un système à profondeur borné (les événements bornant la profondeur correspondant aux événements visibles dans les séquences).

4.5 Bilan

Dans ce chapitre nous avons introduit différentes notions permettant caractériser un système sous diagnostic et plus particulièrement ses performances.

Nous avons tout d'abord abordé la notion de détectabilité d'événements. Cette performance centrale d'un système sous diagnostic décrit la capacité du système à détecter l'occurrence des événements qu'il contient. Ainsi nous assurons qu'un événement qui est détectable ne peut pas arriver sans que cela soit remarqué.

Nous avons ensuite décrit la deuxième notion centrale des performances de diagnostic qu'est la diagnosticabilité des événements d'un système sous diagnostic. La diagnosticabilité d'un événement caractérise son ambiguité (au niveau de ses effets) avec les autres événements. Plus un événement sera ambigu, plus il sera confondu avec d'autres événements lors du diagnostic et plus il sera difficile de le localiser lors d'une occurrence.

Dans la section suivante nous avons utilisé différentes caractéristiques des systèmes spécifiques au contexte avionique pour raffiner les définitions et métriques de détectabilité et de diagnosticabilité.

Finalement, nous nous sommes intéressés à différents objets représentant les relations de causes à effets du système sous diagnostic comme les coupes et les séquences, et nous avons analysé, à partir d'eux, dans quelle mesure il est possible d'évaluer les performances du système sous diagnostic.

Dans cette dernière partie traitant les relations de causes à effets, nous avons mis en avant le fait que l'objet décrivant les séquences de défaillances pour chaque observation du système est un objet qui comporte plus d'informations que celui, exprimant les effets des événements, que nous avons utilisé pour donner l'ensemble des définitions et métriques de diagnostic.

Définir les effets pour une séquence d'événements plutôt que pour un événement seul aurait permis d'avoir des analyses plus précises. Nous aurions cherché à discriminer directement les séquences d'événements et non les événements. Cela aurait amené des notions plus précises de discriminabilité d'événements et donc de détectabilité et de diagnosticabilité.

Néanmoins nous avons estimé que l'utilisation des séquences de défaillances impacterait de manière trop importante l'applicabilité industrielle d'une telle approche. La première difficulté est que les séquences peuvent être infinies. En outre, même lorsqu'elles ne sont pas infinies, calculer l'ensemble des séquences de défaillances d'un système se révèle être une opération très complexe (au sens combinatoire) dès lors que nous souhaitons traiter des systèmes autres que des petits systèmes. D'ailleurs, en pratique, on n'utilise pas l'ensemble des séquences, mais plutôt les séquences bornées en profondeur.

Ayant la volonté de ne pas nous écarter de la réalité et du contexte pratique des travaux dans lesquels s'inscrivent la thèse, nous avons préféré considérer les événements et leurs effets qui forment

un objet plus adapté à l'application sur des cas pratiques. En revanche, d'un point de vue purement théorique ou dans le cadre d'application à des systèmes sous diagnostic de faible taille, il peut être intéressant d'utiliser les séquences de défaillances pour définir la détectabilité et la diagnosticabilité de séquences et ainsi obtenir une évaluation des performances de ces systèmes plus précise.

Chapitre 5

Comparaison et optimisation du monitoring

Dans le chapitre 4 précédent, nous avons caractérisé les performances d'un système sous diagnostic à travers les notions de détectabilité et de diagnosticabilité. Mais le travail que nous souhaitons effectuer ne s'arrête pas à la mesure des performances de diagnostic. L'étape suivante est d'améliorer nos systèmes sous diagnostic. Nous pouvons les améliorer d'un point de vue performances en les modifiant, et nous pouvons aussi chercher à optimiser le monitoring en réduisant l'ensemble des capteurs utilisés tout en s'assurant que l'on conserve les performances de diagnostic.

Pour savoir si les modifications que nous avons appliqué à un système l'ont rendu meilleur, il est nécessaire d'introduire la notion de comparaison de systèmes. La première section de ce chapitre s'intéresse donc à la comparaison des systèmes sous diagnostic du point de vue des performances de diagnostic.

Ensuite, nous nous intéressons à la possibilité qu'une partie du monitoring du système sous diagnostic puisse être supprimé tout en maîtrisant les performances de diagnostic du système. La notion de superfluité de monitoring qui décrit cela est introduite dans la seconde section de ce chapitre.

5.1 Comparaison de systèmes sous diagnostic

Dans cette section, nous utiliserons ces définitions et métriques préalablement définies pour comparer différents systèmes du point de vue du diagnostic. Nous donnerons donc des définitions permettant de caractériser le fait qu'un système sous diagnostic est meilleur qu'un autre, puis nous introduirons quelques métriques utilisant ces définitions qui permettront d'évaluer les évolutions des systèmes.

5.1.1 Définitions

Lorsque nous exprimons la volonté de comparer des systèmes sous diagnostic du point de vue du diagnostic, ce que nous souhaitons faire de manière plus fréquente en réalité, c'est comparer pour un même système différentes stratégies de monitoring. Cela revient à comparer pour un même système S les qualités de différents couples $(O, \mathcal{R}_O)$ où O est l'observateur et $\mathcal{R}_O$ la relation d'observation.

Nous avons vu dans le chapitre précédent différentes notions permettant de caractériser la qualité d'un système pour le diagnostic. La détectabilité donne la qualité d'un système à détecter

les événements, la diagnosticabilité évalue la qualité du système à bien les isoler. Les deux notions fondamentales pour caractériser la qualité d'un système sous diagnostic sont donc la détectabilité et la diagnosticabilité. Ce sont sur elles que nous allons nous appuyer pour comparer différents systèmes sous diagnostic.

L'idée maintenant est de définir une relation de comparaison basée sur ces notions. Le problème est qu'il est difficile de combiner les notions de détectabilité et de diagnosticabilité pour comparer deux systèmes sous diagnostic en toute objectivité. En effet, il faudrait définir l'importance de la détectabilité par rapport à la diagnosticabilité. Il n'y a pas réellement de vérité objective là-dessus, par contre, les pratiques actuelles dans le monde avionique peuvent aiguiller certains choix concernant l'importance de ces deux notions.

Aujourd'hui, la détectabilité et notamment le ratio de "couverture" est une mesure connue et fréquemment utilisée dans le domaine avionique. C'est une mesure clé pour caractériser les systèmes sous diagnostic, et les systèmes sous surveillance en général. Cette notion apparaît même dans les documents de description des systèmes. Lors de la spécification de systèmes avioniques, les avionneurs incluent souvent la détectabilité comme une qualité du système à atteindre dans le contexte de la maintenance.

La détectabilité est une notion primordiale pour le diagnostic, qui est, dans sa version la plus simple, déjà utilisée dans la maintenance avionique. Elle doit prendre une part importante dans la comparaison de systèmes sous diagnostic. En parallèle, la diagnosticabilité n'a pas encore franchi ces étapes dans le même contexte. Il n'y a pour l'instant pas de mesures similaires intégrées dans les processus industriels pour la diagnosticabilité. Le seul type de mesure se rapprochant de la diagnosticabilité que l'on voit apparaître est la notion de "diagnostic en trois coups", que nous pourrions énoncer comme représentant la proportion des pannes qui peuvent être isolées en au maximum trois tentatives. Mais la diagnosticabilité dans son ensemble comme nous l'avons définie reste une mesure encore confidentielle.

Nous pouvons néanmoins remarquer que les notions de détectabilité et de diagnosticabilité ont une certaine proximité, et que, dans la pratique, un système sous diagnostic dont la détectabilité est médiocre aura une qualité de diagnosticabilité médiocre elle aussi. En effet, généralement en pratique, les événements non détectables sont les événements qui vont n'avoir aucun effet visible, cela signifiant bien souvent qu'ils se confondent avec l'état nominal (initial) du système sous diagnostic. De ce fait, tous ces événements non détectables vont être non discriminables entre eux. Ainsi le degré de diagnostic de tous ces événements sera élevé, le degré de diagnosticabilité moyen sera élevé, et la qualité de diagnosticabilité du système sera médiocre.

Malgré cela, de manière générale, il est faux de dire que si un système sous diagnostic a une performance de détectabilité meilleure qu'un autre système sous diagnostic, alors il aura aussi forcément une meilleure performance de diagnosticabilité. La seule chose que l'on peut dire est que dans la pratique, un système qui a une performance de détectabilité médiocre alors qu'un deuxième en a une bonne n'aura pas une performance de diagnosticabilité meilleure que celle du deuxième.

Néanmoins, cela ne permet pas de trouver comment combiner détectabilité et diagnosticabilité dans une seule mesure de performances de manière objective.

Dans un premier temps, nous allons donc conserver les deux comparaisons dissociées, celle du point de vue de la détectabilité et celle du point de vue de la diagnosticabilité.

Définition 38 (Relation d'ordre de détectabilité pour SUD) *Soient* $A_1 = \langle C_1, I_1, E_1, T_1 \rangle$ *et* $A_2 = \langle C_2, I_2, E_2, T_2 \rangle$ *deux systèmes sous diagnostic. La relation d'ordre* $<_{de}$ *compare* A_1 *et* A_2 *selon leur détectabilité.*

$$A_1 <_{de} A_2 \iff A_1.RDF_{C_1} < A_2.RDF_{C_2} \tag{5.1}$$

Remarque *Si ni* $A_1 <_{de} A_2$ *ni* $A_2 <_{de} A_1$ *alors on a* $A_1 =_{de} A_2$.

Exemple:

En utilisant la table A.3 on obtient : $\{\gamma_1, \gamma_2\} <_{de} \{\gamma_3, \gamma_4, A_{ex}\}$.

◦

On peut bien entendu, selon le contexte dans lequel on est, effectuer une comparaison de la détectabilité de deux systèmes sous diagnostic en regardant non pas le ratio de détectabilité forte, mais le ratio de détectabilité globale (forte et faible) ou les autres ratios définis dans les métriques de détectabilité (4.1.2), ou bien dans un contexte avionique, les métriques qui y sont adaptées (4.3.1.2).

Les métriques les plus représentatives de la qualité de diagnosticabilité d'un système sous diagnostic sont le degré de diagnosticabilité moyen (mét. 10) et les autres métriques qui en dérivent (pondération et adaptées au contexte avionique). Ainsi nous pouvons introduire une relation d'ordre pour systèmes sous diagnostic pour la diagnosticabilité.

Définition 39 (Relation d'ordre de diagnosticabilité pour SUD) *Soient* $A_1 = \langle C_1,\ I_1,\ E_1,\ T_1\rangle$ *et* $A_2 = \langle C_2,\ I_2,\ E_2,\ T_2\rangle$ *deux systèmes sous diagnostic. La relation d'ordre* $<_{di}$ *compare* A_1 *et* A_2 *selon leur diagnosticabilité.*

$$A_1 <_{di} A_2 \iff A_1.DDM_{C_1} > A_2.DDM_{C_2} \tag{5.2}$$

Remarque *Si ni* $A_1 <_{di} A_2$ *ni* $A_2 <_{di} A_1$ *alors on a* $A_1 =_{di} A_2$.

Exemple:

En utilisant la table A.5, on obtient : $\gamma_1 <_{di} \gamma_2 <_{di} \{\gamma_3, \gamma_4, A_{ex}\}$.

∘

Comme nous l'avons exprimé en début de section, les relations d'ordre de détectabilité et de diagnosticabilité pour SUD ne sont pas compatibles. Il n'y a donc pas d'ordre logiquement objectif que l'on peut déduire qui combine les deux notions de détectabilité et de diagnosticabilité.

Prenons l'exemple de deux systèmes simples $A_1 = \langle C_1, I_1, E_1, T_1\rangle$ et $A_2 = \langle C_2, I_2, E_2, T_2\rangle$ en figures 5.1 et 5.2 pour illustrer la non-compabilité de ces deux ordres que nous venons de définir.

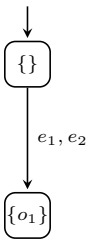


FIGURE 5.1 – Système A_1

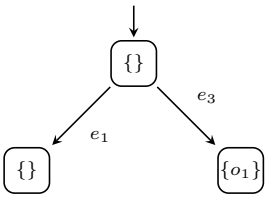


FIGURE 5.2 – Système A_2

En considérant que $Obs_{OK} = \{\{\}\}$, nous avons $A_2 <_{de} A_1$ puisque tous les événements de A_1 sont détectables ($A_1.RDF_{C_1} = 100\%$) alors que pour A_2, e_1 n'est pas détectable ($A_2.RDF_{C_2} = 50\%$).

Or, pour A_1, e_1 et e_2 ayant les mêmes effets, ils sont tous les deux dans le même groupe d'ambiguité et sont de degré 2, alors que pour A_2, l'événement e_1, qui partage ses effets avec l'état initial, est de degré 2 (groupe d'ambiguité $\{e_1, e_I\}$) quand l'événement e_2 est lui diagnosticable. Nous avons donc $A_1.DDM_{C_1} > A_2.DDM_{C_2}$ et ainsi $A_1 <_{di} A_2$.

Nous remarquons à travers cet exemple que les relations d'ordre de détectabilité et de diagnosticabilité ne sont pas compatibles. Une simple combinaison des deux ordres n'est pas possible, et il faudra donc départager ces deux notions pour départager deux systèmes sous diagnostic.

De la même façon que précédemment avec la détectabilité, selon le contexte, les métriques de degré de diagnosticabilité pondérées et adaptées à l'avionique peuvent être utilisées à la place de la métrique de degré de diagnosticabilité moyen simple.

Nous pouvons aussi utiliser la notion de profondeur de systèmes pour définir une relation d'ordre sur les systèmes sous diagnostic pour la diagnosticabilité. Cette relation compare la diagnosticabilité des systèmes pour une profondeur de 1 événement bornant, puis pour une profondeur de 2 événements bornants, etc.

Définition 40 (Relation d'ordre de diagnosticabilité pour SUD avec profondeur) *Soient $A_1 = \langle C_1, I_1, E_1, T_1\rangle$, $A_2 = \langle C_2, I_2, E_2, T_2\rangle$ deux systèmes sous diagnostic et n un entier positif représentant la profondeur dans le système. La relation d'ordre $<^n_{di}$ compare A_1 et A_2 selon leur diagnosticabilité dans le système réduit en profondeur.*

$$A_1 <^n_{di} A_2 \iff \bigvee_{i=0}^{n} \left(A_1.DDM_{C_1^i} > A_2.DDM_{C_2^i} \wedge \bigwedge_{j=0}^{i-1} A_1.DDM_{C_1^j} = A_2.DDM_{C_2^j} \right) \tag{5.3}$$

Remarque *Si ni $A_1 <^n_{di} A_2$ ni $A_2 <^n_{di} A_1$ alors on a $A_1 =^n_{di} A_2$.*

Exemple:

En utilisant les tables A.5 et A.7, on obtient : $\gamma_1 <^2_{di} \gamma_2 <^2_{di} \gamma_3 <^2_{di} \{\gamma_4, A_{ex}\}$.

∘

Une façon qui semble répondre de manière la plus objective possible à la mise en commun des deux mesures de détectabilité et de diagnosticabilité pour comparer des systèmes, s'appuie sur l'utilisation actuelle de ces notions que nous avons décrite précédemment. En utilisant le fait que la détectabilité est utilisée lors de la spécification d'un système comme une qualité à atteindre, pour définir une comparaison globale entre systèmes sous diagnostic, on peut fixer une valeur minimale pour la détectabilité et comparer ensuite les valeurs de diagnosticabilité. La valeur de détectabilité minimale correspond alors à cette valeur de couverture fournie dans le cadre de la spécification du système sous diagnostic.

Introduisons donc la *relation d'ordre pour systèmes sous diagnostic* utilisée pour comparer des systèmes sous diagnostic.

Définition 41 (Relation d'ordre pour systèmes sous diagnostic) *Soient $A_1 = \langle C_1, I_1, E_1, T_1\rangle$, $A_2 = \langle C_2, I_2, E_2, T_2\rangle$ deux systèmes sous diagnostic et K le pourcentage de détectabilité minimum à atteindre pour les SUD. La relation d'ordre $<_K$ compare les systèmes sous diagnostic A_1 et A_2.*

$$\begin{aligned} A_1 <_K A_2 \iff\ & A_1.RDF_{C_1} < K \wedge A_2.RDF_{C_2} \geq K \vee \\ & A_1.RDF_{C_1} < K \wedge A_2.RDF_{C_2} < K \wedge A_1 <_{di} A_2 \vee \\ & A_1.RDF_{C_1} \geq K \wedge A_2.RDF_{C_2} \geq K \wedge A_1 <_{di} A_2 \end{aligned} \tag{5.4}$$

Remarque *Si ni $A_1 <_K A_2$ ni $A_2 <_K A_1$ alors on a $A_1 =_K A_2$.*

Exemple:

En utilisant les tables A.3 et A.5 on obtient : $\gamma_1 <_{70\%} \gamma_2 <_{70\%} \{\gamma_3, \gamma_4, A_{ex}\}$.

∘

Il est possible d'affiner la comparaison de systèmes en utilisant la notion de profondeur.

Définition 42 (Relation d'ordre pour systèmes sous diagnostic avec profondeur) *Soient $A_1 = \langle C_1, I_1, E_1, T_1\rangle$, $A_2 = \langle C_2, I_2, E_2, T_2\rangle$ deux systèmes sous diagnostic, K le pourcentage de détectabilité minimum à atteindre pour les SUD, et n un entier positif représentant une profondeur du système. La relation d'ordre $<^n_K$ compare les systèmes sous diagnostic A_1 et A_2 en utilisant leur profondeur.*

$$\begin{aligned} A_1 <^n_K A_2 \iff\ & A_1.RDF_{C_1} < K \wedge A_2.RDF_{C_2} \geq K \vee \\ & A_1.RDF_{C_1} < K \wedge A_2.RDF_{C_2} < K \wedge A_1 <^n_{di} A_2 \vee \\ & A_1.RDF_{C_1} \geq K \wedge A_2.RDF_{C_2} \geq K \wedge A_1 <^n_{di} A_2 \end{aligned} \tag{5.5}$$

Remarque *Si ni $A_1 <^n_K A_2$ ni $A_2 <^n_K A_1$ alors on a $A_1 =^n_K A_2$.*

Exemple:

En utilisant les tables A.3, A.5 et A.7 on obtient : $\gamma_1 <^2_{70\%} \gamma_2 <^2_{70\%} \gamma_3 <^2_{70\%} \{\gamma_4, A_{ex}\}$.

∘

Comme nous l'évoquions en début de section, les objets qui ont une réelle raison d'être comparés, ce sont différentes stratégies de monitoring (observateur et relation d'observation) pour un même système, afin de savoir quelle est celle qui permet les meilleures performances au système sous diagnostic.

La considération de la comparaison de différents couples $(O, \mathcal{R}_O)$ pour un même système S plutôt que la comparaison plus globale de systèmes sous diagnostic différents apporte une spécificité qui permet de donner des versions supplémentaires des définitions précédentes. En effet, dans ce cas-là, nous pouvons poser ces définitions pour un ensemble de configurations, et non pour l'ensemble des configurations total des systèmes sous diagnostic. Donnons l'exemple avec la version adaptée depuis la définition 39.

Définition 43 (Relation d'ordre de détectabilité pour observateurs) *Soient $S = \langle Q, Q_i, E, \rightarrow\rangle$ un système, Q' un ensemble de configurations de Q, et $m_1 = (O_1, \mathcal{R}_{O_1})$ et $m_2 = (O_2, \mathcal{R}_{O_2})$ deux stratégies de monitoring pour S. Considérons C'_1 et C'_2, deux ensembles de configurations respectivement de $A_1 = \langle S, O_1, \mathcal{R}_{O_1}\rangle$ et $A_2 = \langle S, O_2, \mathcal{R}_{O_2}\rangle$ tels que $C'_1[V] = C'_2[V] = Q'$. La relation d'ordre $\overset{Q'}{<}_{de}$ compare m_1 et m_2 pour S selon leur détectabilité forte respectivement dans C'_1 et C'_2.*

$$m_1 \overset{Q'}{<}_{de} m_2 \iff A_1.DDM_{C'_1} > A_2.DDM_{C'_2} \tag{5.6}$$

Remarque *Si ni $m_1 \overset{Q'}{<}_{de} m_2$ ni $m_2 \overset{Q'}{<}_{de} m_1$ alors on a $m_1 \overset{Q'}{=}_{de} m_2$.*

Exemple:

Pour les $\gamma_j \in \Gamma$, $\gamma_j.Z^1_{ex_j}[V] = Q^1_{ex}$ pour toute profondeur n. Considérons pour $j \in [\![1 \dots 4]\!]$, les couples $(O_j, \mathcal{R}_{O_j})$; l'observateur O_j est défini par l'ensemble de variables $V_O \setminus \{o_j\}$ et la relation d'observation $\mathcal{R}_{O_j} \subseteq BF(V^{\gamma_j})$ est telle que $[\![\mathcal{R}_{O_j}]\!] = [\![\mathcal{R}_O]\!][V^{\gamma_j}]$.

En utilisant la table A.8, on obtient :

$\{(O_1, \mathcal{R}_{O_1}), (O_2, \mathcal{R}_{O_2})\} \overset{Q^1_{ex}}{<}_{de} \{(O_3, \mathcal{R}_{O_3}), (O_4, \mathcal{R}_{O_4}), (O, \mathcal{R}_O)\}$.

∘

On peut de la même façon dériver les versions pour observateurs et dans un ensemble de configurations particulier pour les définitions 39, 40, 41 et 42.

5.1.2 Métriques

Dans cette partie nous allons donner quelques métriques qui peuvent caractériser les différences entre deux systèmes. Ces métriques sont le reflet de l'évolution de la détectabilité et de la diagnosticabilité entre deux systèmes.

Introduisons tout d'abord le degré d'évolution de la détectabilité entre deux systèmes. La détectabilité étant déclinée en versions forte et faible, on peut identifier deux métriques associées. Nous n'explicitons ici que la version "forte" de la métrique.

Métrique 21 (Degré d'évolution de détectabilité forte) *Soient $A_1 = \langle C_1, I_1, E_1, T_1\rangle$ et $A_2 = \langle C_2, I_2, E_2, T_2\rangle$ deux systèmes sous diagnostic. Le* degré d'évolution de détectabilité forte *entre A_1 et A_2, noté $DEDF^{A_1 \to A_2}$, représente l'évolution (positive ou négative) du ratio de détectabilité forte de A_1 vis-à-vis de celui de A_2.*

$$DEDF^{A_1 \to A_2} = \frac{A_2.RDF_{C_2}}{A_1.RDF_{C_1}} - 1 \tag{5.7}$$

Exemple:

A'	$DEDF^{\gamma_1 \to A'}$	$DEDF^{\gamma_2 \to A'}$	$DEDF^{\gamma_3 \to A'}$	$DEDF^{\gamma_3 \to A'}$	$DEDF^{A_{ex} \to A'}$
γ_1	0%	$\frac{4/7}{4/7} - 1 = 0\%$	$\frac{5/7}{4/7} - 1 = 25\%$	$\frac{5/7}{4/7} - 1 = 25\%$	$\frac{5/7}{4/7} - 1 = 25\%$
γ_2	$\frac{4/7}{4/7} - 1 = 0\%$	0%	$\frac{5/7}{4/7} - 1 = 25\%$	$\frac{5/7}{4/7} - 1 = 25\%$	$\frac{5/7}{4/7} - 1 = 25\%$
γ_3	$\frac{4/7}{5/7} - 1 = -20\%$	$\frac{4/7}{5/7} - 1 = -20\%$	0%	$\frac{5/7}{5/7} - 1 = 0\%$	$\frac{5/7}{5/7} - 1 = 0\%$
γ_4	$\frac{4/7}{5/7} - 1 = -20\%$	$\frac{4/7}{5/7} - 1 = -20\%$	$\frac{5/7}{5/7} - 1 = 0\%$	0%	$\frac{5/7}{5/7} - 1 = 0\%$
A_{ex}	$\frac{4/7}{5/7} - 1 = -20\%$	$\frac{4/7}{5/7} - 1 = -20\%$	$\frac{5/7}{5/7} - 1 = 0\%$	$\frac{5/7}{5/7} - 1 = 0\%$	0%

∘

Le degré d'évolution de détectabilité est un ratio qui s'il est négatif, caractérise une perte de détectabilité en passant de A_1 à A_2. Si au contraire il est positif, il représente le gain de détectabilité qu'entraîne le fait de passer du système A_1 au système A_2.

Nous pouvons caractériser de la même manière l'évolution de la diagnosticabilité entre deux systèmes.

Métrique 22 (Degré d'évolution de diagnosticabilité) *Soient* $A_1 = \langle C_1, I_1, E_1, T_1 \rangle$ *et* $A_2 = \langle C_2, I_2, E_2, T_2 \rangle$ *deux systèmes sous diagnostic. Le* degré d'évolution de diagnosticabilité *entre* A_1 *et* A_2*, noté* $DED^{A_1 \to A_2}$*, représente l'évolution (positive ou négative) du degré de diagnosticabilité moyen de* A_1 *vis-à-vis de celui de* A_2*.*

$$DED^{A_1 \to A_2} = \frac{A_1.DDM_{C_1}}{A_2.DDM_{C_2}} - 1 \tag{5.8}$$

Exemple:

A'	$DED^{\gamma_1 \to A'}$	$DED^{\gamma_2 \to A'}$	$DED^{\gamma_3 \to A'}$	$DED^{\gamma_3 \to A'}$	$DED^{A_{ex} \to A'}$
γ_1	0%	$\frac{21/7}{19/7} - 1 \simeq 11\%$	$\frac{21/7}{16/7} - 1 \simeq 31\%$	$\frac{21/7}{16/7} - 1 \simeq 31\%$	$\frac{21/7}{16/7} - 1 \simeq 31\%$
γ_2	$\frac{19/7}{21/7} - 1 \simeq -10\%$	0%	$\frac{19/7}{16/7} - 1 \simeq 19\%$	$\frac{19/7}{16/7} - 1 \simeq 19\%$	$\frac{19/7}{16/7} - 1 \simeq 19\%$
γ_3	$\frac{16/7}{21/7} - 1 \simeq -24\%$	$\frac{16/7}{19/7} - 1 \simeq -16\%$	0%	$\frac{16/7}{16/7} - 1 = 0\%$	$\frac{16/7}{16/7} - 1 = 0\%$
γ_4	$\frac{16/7}{21/7} - 1 \simeq -24\%$	$\frac{16/7}{19/7} - 1 \simeq -16\%$	$\frac{16/7}{16/7} - 1 = 0\%$	0%	$\frac{16/7}{16/7} - 1 = 0\%$
A_{ex}	$\frac{16/7}{21/7} - 1 \simeq -24\%$	$\frac{16/7}{19/7} - 1 \simeq -16\%$	$\frac{16/7}{16/7} - 1 = 0\%$	$\frac{16/7}{16/7} - 1 = 0\%$	0%

∘

De même que précédemment, un résultat positif représentera un gain, et un résultat négatif, une perte dans le passage de A_1 à A_2 au niveau de la diagnosticabilité.

Nous n'expliciterons pas ces versions, mais il est possible, de la même manière que pour la définition 43, de baser les définitions sur les observateurs plutôt que sur les systèmes sous diagnostic et d'ajouter le paramètre de l'ensemble de configurations considéré.

5.2 Superfluité de monitoring

Dans les deux sections 4.1 et 4.2 sur les performances de diagnostic, nous nous sommes intéressés à la détectabilité et la diagnosticabilité de système, qui sont des notions centrées sur les événements.

Dans cette section, nous regardons la notion de superfluité de monitoring, centrée sur l'observateur. La superfluité permet de caractériser le résultat d'un monitoring comme n'étant pas utile pour le diagnostic.

Cette section est organisée en deux parties. La première sera dédiée aux définitions de superfluité, et la seconde aux métriques qui peuvent évaluer la superfluité du système.

5.2.1 Définitions

La méthode utilisée pour définir la superfluité d'un ensemble de messages de monitoring (ou variables d'observation) consiste à évaluer les caractéristiques du système avec et sans ces messages, autrement dit, à comparer les capacités de détectabilité et de diagnosticabilité du système sous diagnostic de départ A avec celles du système sous diagnostic γ obtenu en retirant ces variables d'observation de l'observateur de A.

Nous utilisons donc le système sous diagnostic γ obtenu par retrait d'un ensemble de variables d'observation V_s. Il est issu de la définition 7.

En outre, nous trouverons en Annexe A différentes caractéristiques évaluées (détectabilité, diagnosticabilité, etc.) des systèmes de Γ introduits dans la section d'illustration, en 3.7.4. Ces systèmes seront utilisés comme exemples pour illustrer les définitions à venir.

Nous avons montré à travers le théorème 3 que le système obtenu en retirant un ensemble de variables d'observation d'un système sous diagnostic est aussi un système sous diagnostic sur le même système de départ. Les définitions appliquées sur le SUD A sont donc applicables sur le SUD γ.

Par la suite, nous noterons γ.P l'application de la propriété P pour γ. Par exemple, $\gamma.DDM_{Z'}$ représente le degré de diagnosticabilité moyen de γ pour un ensemble de configurations Z'.

Nous pouvons désormais nous intéresser à la notion de superfluité d'un ensemble de variables d'observation.

Définition 44 (Superfluité d'un ensemble de var. d'observation pour un événement) *Soient C' un ensemble de configurations, e un événement et V_s un ensemble de variables d'observation. V_s est superflu pour e dans C' si le système sous diagnostic $\gamma = A \setminus V_s$ conserve les qualités de détectabilité (forte et faible), ainsi que de diagnosticabilité de A pour e.*

$$\begin{cases} \textit{(détectabilité forte)} & [e \in A.E_{FD}^{C'}] = [e \in \gamma.E_{FD}^{Z'}] \\ \textit{(détectabilité faible)} & [e \in A.E_{fD}^{C'}] = [e \in \gamma.E_{fD}^{Z'}] \\ \textit{(diagnosticabilité)} & A.degDiag(C', e) = \gamma.degDiag(Z', e) \end{cases} \tag{5.9}$$

où $\gamma = \langle Z, Z_i, E, \rightarrow_\gamma \rangle$, V^γ sont les variables de γ et $Z' = C'[V^\gamma]$.

Exemple:

En utilisant les tables A.3 et A.5, nous pouvons illustrer la superfluité des ensembles de variables $\{o_1\}$, $\{o_2\}$, $\{o_3\}$ et $\{o_4\}$.

e	e_1	e_2	e_3	e_4	e_5	e_6	e_7
$\{o_1\}$ *fortement superflu pour e dans C_{ex} ?*	N	N	O	O	O	O	N
$\{o_2\}$ *fortement superflu pour e dans C_{ex} ?*	N	N	O	O	O	O	N
$\{o_3\}$ *fortement superflu pour e dans C_{ex} ?*	O	O	O	O	O	O	O
$\{o_4\}$ *fortement superflu pour e dans C_{ex} ?*	O	O	O	O	O	O	O

∘

On s'assure à travers la notion de superfluité d'un ensemble de variables pour un événement que le système amputé de ces variables conserve la même qualité de détectabilité (forte et faible) et de diagnosticabilité pour cet événement, sachant qu'il est impossible d'améliorer la détectabilité globale ou la diagnosticabilité du système en lui enlevant des variables d'observation. Les variables répondant positivement à la définition ci-dessus sont donc déclarées comme superflues pour l'événement visé d'un point de vue diagnostic.

Regardons maintenant la superfluité pour un système.

Définition 45 (Superfluité d'un ensemble de variables d'observation) *Soient C' un ensemble de configurations et V_s un ensemble de variables d'observation. V_s est superflu dans C' s'il est superflu dans C' pour tous les événements du système déclenchables dans C'.*

$$\forall e \in E^{C'},\ V_s \textit{ est fortement superflu pour } e \textit{ dans } C' \tag{5.10}$$

Exemple:
$\{o_1\}$ *n'est pas superflu dans* C_{ex}.
$\{o_2\}$ *n'est pas superflu dans* C_{ex}.
$\{o_3\}$ *est superflu dans* C_{ex}.
$\{o_4\}$ *est superflu dans* C_{ex}.

◦

5.2.2 Métriques

Les métriques de superfluité présentées dans cette partie sont deux ratios (une version classique, une version pondérée) donnant la proportion des événements déclenchables pour lesquels un ensemble de variables d'observation est superflu.

Notons $E^{V_s}_{C'}$ l'ensemble des événements déclenchables pour lesquels, dans un ensemble de configurations C', l'ensemble des variables d'observation V_s est superflu.

$$E^{V_s}_{C'} = \{e \in E^{C'} \mid V_s \text{ est superflu pour } e \text{ dans } C'\} \tag{5.11}$$

Exemple:

V_s	$\{o_1\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$
$E^{V_s}_{C_{ex}}$	$\{e_3, e_4, e_5, e_6\}$	$\{e_3, e_4, e_5, e_6\}$	E_{ex}	E_{ex}

◦

Cette notation introduite, nous pouvons définir les différentes métriques de superfluité.

Métrique 23 (Ratio de superfluité d'un ensemble de var. d'observation) *Soient* C' *un ensemble de configurations et* V_s *un ensemble de variables d'observation. Le* ratio de superfluité $RS^{V_s}_{C'}$ *donne la proportion d'événements pour lesquels* V_s *est superflu dans* C'*, où on ne considère que les événements qui y sont déclenchables.*

$$RS^{V_s}_{C'} = \frac{|E^{V_s}_{C'}|}{|E^{C'}|} \tag{5.12}$$

Exemple:

V_s	$\{o_1\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$
$RS^{V_S}_{C_{ex}}$	$\frac{4}{7} \simeq 57\%$	$\frac{4}{7} \simeq 57\%$	$\frac{7}{7} = 100\%$	$\frac{7}{7} = 100\%$

◦

Tout comme pour les métriques de détectabilité (mét. 5) et de diagnosticabilité (mét. 11), on peut introduire une pondération des événements à travers la fonction w. La superfluité pour un événement important (au sens de la pondération) sera de ce fait plus prise en compte dans le résultat de la métrique.

Métrique 24 (Ratio pondéré de superfluité d'un ensemble de var. d'observation) *Soient* C' *un ensemble de configurations et* V_s *un ensemble de variables d'observation. Le* ratio pondéré de superfluité $RPS^{V_s}_{C'}$ *donne la proportion d'événements pour lesquels* V_s *est superflu dans* C'*, où on ne considère que les événements qui y sont déclenchables, et où les événements sont pondérés par une fonction* w.

$$RPS^{V_s}_{C'} = \frac{\sum_{e \in E^{V_s}_{C'}} w(e)}{\sum_{e \in E^{C'}} w(e)} \tag{5.13}$$

Exemple:

Utilisons la fonction de pondération w_{ex} définie pour la métrique 5.

V_s	$\{o_1\}$	$\{o_2\}$	$\{o_3\}$	$\{o_4\}$
$RPS_{C_{ex}}^{V_S}$	$\frac{40}{91} \simeq 44\%$	$\frac{40}{91} \simeq 44\%$	$\frac{91}{91} = 100\%$	$\frac{91}{91} = 100\%$

$\circ$

Troisième partie

Expérimentations et intégration industrielle

Chapitre 6

Approche et méthode

Dans ce chapitre nous présentons la méthode que nous utilisons en pratique pour effectuer l'analyse d'un système que l'on souhaite observer. L'approche générale est une approche à base de modèle et se décompose de manière basique en une procédure de trois volets :

1. Obtention d'un modèle du système observé.
2. Analyse du modèle et calcul pour l'aide au diagnostic.
3. Utilisation des résultats.

Dans le cadre de cette thèse, nous avons choisi de réaliser ces différents volets de la manière suivante :

- Obtention d'un modèle : Création d'un modèle AltaRica décrivant le comportement du système observé.
- Analyse du modèle : Utilisation de l'outil ARC associé à AltaRica pour effectuer des calculs sur le modèle concernant notamment la détectabilité et la diagnosticabilité du système sous diagnostic.
- Utilisation des résultats de calcul pour définir les performances de diagnostic du modèle.

Dans ce chapitre nous expliquons donc comment en pratique nous analysons les systèmes sous diagnostic. Pour ce faire, nous commençons en présentant le langage AltaRica et ses outils qui nous serviront à la modélisation et au calcul de performances, puis nous donnerons la correspondance avec AltaRica des définitions et métriques de performances de diagnostic de la partie précédente, et finalement, nous illustrerons cette procédure sur l'exemple simple utilisé dans l'ensemble de la partie II pour illustrer les définitions et métriques.

6.1 Le langage AltaRica et ses outils

Dans cette section, nous présentons AltaRica, le langage de modélisation choisi pour représenter les systèmes sous diagnostic.

Nous présenterons les points importants pour la compréhension du langage. Le but n'est pas d'expliciter toutes les caractéristiques du langage AltaRica et de ses outils associés dans le détail. Pour avoir une description exhaustive, nous pourrons nous référer aux articles [PTL+98, PR99, AGPR00] qui définissent le langage et les outils, ainsi qu'aux thèses [Poi00, Vin03a, Ber09] sur lesquelles nous nous appuyons notamment pour la rédaction de cette section.

6.1.1 Préambule

Cette partie devient sur les origines du langage AltaRica.

6.1.1.1 Origine du projet AltaRica

AltaRica est né au LaBRI à la fin des années 90 de besoins industriels pour analyser le comportement de systèmes critiques. Originellement, il s'agissait de pouvoir traiter des problèmes de sûreté de fonctionnement en utilisant des méthodes formelles.

AltaRica a alors pour but de permettre :

- La création de modèles formels pour représenter les systèmes critiques.
- L'utilisation des méthodes formelles pour faire des analyses sur les modèles.
- La description à la fois de systèmes fonctionnels et dysfonctionnels.

6.1.1.2 Choix de représentation

AltaRica est un langage de modélisation formelle basée sur la notion d'automate à contraintes [PR99]. On trouve aussi cette notion de système de transitions avec contraintes dans [BR94, CR97]. Cette notion est proche de celle d'automate gardé (*guarded automaton*). Un automate à contraintes est un automate fini pour lequel les variables et les transitions sont soumises à des contraintes.

Les automates à contraintes sont utilisés pour représenter des systèmes à événements discrets. Les nœuds (ou configurations) composent les différents états du système et les événements qui étiquettent les transitions permettent de passer d'une configuration à une autre.

6.1.1.3 Historique

Nous présentons dans cette partie les principales étapes de la vie du langage et citons quelques travaux basés sur AltaRica. Pour avoir un historique complet détaillé, on pourra se référer par exemple à la thèse de Romain Bernard [Ber09].

Comme nous l'avons dit précédemment, les premiers travaux au LaBRI de recherche pour la création d'un langage de description de système complexes ont commencé vers 1997. Les travaux ont été dès le départ effectués en partenariat avec des industriels parmi lesquels on peut citer IXI et Elf Aquitaine pour l'analyse des besoins. Suite à l'analyse de besoins, il est apparu comme intéressant de créer ce qui deviendra par la suite le langage AltaRica.

Les années nous séparant de la décennie suivante furent consacrées à la formalisation du langage [PTL$^+$98, PR99, AGPR00], de sa sémantique et à l'élaboration d'outils-prototypes comme un simulateur ou un compilateur de modèles AltaRica vers les outils Aralia [RD97] pour la génération d'arbres de fautes ou Mec[GV04] pour la vérification de modèles. D'autres entreprises et laboratoire comme Dassault Aviation, Schneider Electric, l'Institut de Protection et de Sûreté Nucléaire (IPSN), Thomson Detexis, ou encore l'Office National d'Études et de Recherche Aérospatiales (ONERA) se sont intéressés au langage et sont devenus utilisateurs d'AltaRica.

Le projet AltaRica connaîtra ensuite un tournant avec la création de sous-langages à partir de l'AltaRica d'origine (AltaRica LaBRI). Ces sous-langages, tels que l'AltaRica Data Flow [Rau02, BDRS06] ou l'AltaRica OCAS sont des restrictions du langage pour répondre au mieux aux besoins spécifiques de certaines entreprises comme par exemple EADS Apsys avec l'outil *SIMFIA* utilisant l'AltaRica Data Flow et Dassault Aviation avec l'outil *Cecilia OCAS* utilisant l'AltaRica OCAS. Aujourd'hui, on retrouve de nombreux outils développés par les entreprises pour leur utilisation à base d'AltaRica, parmi lesquels les plus connus :

- *BPA-DAS Safety Designer* par Dassault Systèmes
- *Cecilia OCAS* par Dassault Aviation
- *RAMSES* par EADS Airbus
- *SIMFIA* par EADS Apsys

Ces utilisations spécifiques sont l'occasion de voir se développer différentes interfaces graphiques pour la modélisation de systèmes, et d'étendre par la même occasion le parc d'utilisateurs

d'AltaRica. Si des grandes entreprises comme EADS Airbus et Astrium, Thales Avionics, Dassault Systèmes n'utilisent pour l'instant AltaRica que dans le cadre de recherches pour réaliser des expérimentations dans le domaine de la sûreté de fonctionnement, Dassault Aviation a déjà utilisé AltaRica dans le cadre de la certification.

Le langage AltaRica a été l'objet de nombreux travaux de thèse :

- Gérald Point [Poi00] a formalisé la sémantique d'AltaRica LaBRI.
- Aymeric Vincent [Vin03a] a conçu un vérificateur de modèles AltaRica, Mec 5.
- Claire Pagetti [Pag04] a travaillé sur une extension temps réel d'AltaRica.
- Christophe Kehren [Keh05] a identifié des motifs d'architecture de systèmes pour la sûreté de fonctionnement.
- Sophie Humbert [Hum08] s'est intéressée à la définition d'exigences de sécurité pour les systèmes.
- Minh-Thang Khuu [Khu08] a travaillé sur l'accélération de la simulation stochastique pour la sûreté de fonctionnement à partir de modèles décrits en AltaRica Data Flow.
- Laurent Sagaspe [Sag08] a proposé une méthodologie et un outil pour la réalisation de l'allocation de ressources dans le cadre de l'aéronautique tout en s'assurant du respect des exigences de sûreté de fonctionnement.
- Hayssam Soueidan [Sou09] a défini BioRica, une extension d'AltaRica pour la modélisation de processus biologiques stochastiques en biologie des systèmes.
- Romain Bernard [Ber09] s'est intéressé à la sûreté de fonctionnement des systèmes critiques, et plus particulièrement quand on considère un système à des niveaux de détails différents, en utilisant la notion de raffinement de modèles AltaRica.
- Fares Chucri [Chu12] a travaillé sur la vérification de modèles AltaRica par l'approche CEGAR (*Counter-Example Guided Abstraction Refinement* [CGJ+00]).
- Romain Adeline [Ade11] a proposé une méthodologie pour la modélisation AltaRica de systèmes physiques et pour la validation de ces modèles.

AltaRica a aussi été présent dans le projet européen MISSA (*More Integrated and cost efficient System Safety Assessment*) d'avril 2008 à mars 2011 et qui avait pour but de proposer des méthodes et outils pour aider les ingénieurs sûreté de fonctionnement à effectuer leurs analyses.

Une conférence autour du langage AltaRica a déjà eu plusieurs éditions. Il s'agit du AltaRica *Workshop* ou désormais *MBSAW (Model-Based Safety Assessment Workshop)* :

- Octobre 2002 : *1st* AltaRica *Workshop* à Toulouse, France.
- Octobre 2003 : *2nd* AltaRica *Workshop* à Marseille, France.
- Novembre 2007 : *3rd* AltaRica *Workshop* à Bordeaux, France.
- Mars 2011 : *MBSAW 2011* à Toulouse, France.
- Septembre 2012 : *MBSAW 2012* à Bordeaux, France.

6.1.2 Le langage AltaRica

Nous donnons dans cette partie un aperçu du langage AltaRica.

Nous ne présenterons pas tous les détails du langage AltaRica, d'autres documents le faisant déjà très bien. Nous pourrons notamment nous référer aux thèses de Gérald Point [Poi00] et Aymeric Vincent [Vin03a] qui donnent une description formelle de la syntaxe et de la sémantique du langage. La présentation du langage proposée sur le site internet dédié au langage AltaRica [1] est aussi intéressante et présente l'ensemble des notions traitées par le langage.

1. `http://altarica.labri.fr/`

En outre, il est intéressant de se référer à la thèse de Romain Bernard [Ber09] qui donne le pendant des définitions d'AltaRica en AltaRica OCAS. Le langage que nous décrivons et que nous utiliserons dans cette thèse est le langage AltaRica d'origine, l'AltaRica LaBRI.

Nous ne donnerons pas la description précise de la syntaxe AltaRica, et proposons de se référer aux documents cités précédemment pour aborder les éléments basiques du langage comme les *mots-clés*, les *identifiants*, les *expressions*, la *visibilité* ou encore les *domaines*. Nous ne décrirons pas non plus les éléments spécifiques tels que les *priorités* ou les *types de données abstraits*.
Nous ne reprenons dans cette partie que les éléments nécessaires pour la réalisation et la compréhension de modèles AltaRica tels que nous les manipulons dans cette thèse.

Une description AltaRica est composé de nœuds. La définition d'un nœud commence par le mot-clé `node` et se termine par le mot-clé `edon`. Un exemple est proposé en figure 6.1.

```
node Composant
  ...
edon
```

FIGURE 6.1 – Déclaration d'un nœud AltaRica

Un nœud simple (appelé aussi nœud feuille) est composé des plusieurs champs, chacun débutant par un mot-clé et se terminant au mot-clé suivant. Dans un nœud simple nous retrouvons donc les champs :

- `state` et `init` définissant les variables d'états.
- `flow` définissant les variables de flux.
- `event` définissant les événements.
- `trans` définissant les transitions.
- `assert` définissant les assertions.
- `extern` permettant la définition de commandes externes au langage mais qui pourront être prises en comptes dans des outils complémentaires.

Le langage intègre la notion de hiérarchie de nœuds. Un nœud à un niveau de hiérarchie peut être un sous-nœud au niveau supérieur. De nouveaux champs spécifiques à la hiérarchie sont alors utilisés :

- `sub` pour exprimer les sous-nœuds.
- `sync` pour la synchronisation d'événements.

Chaque champ a son rôle dans le nœud. Nous allons dans la suite de cette partie sur le langage AltaRica présenter plus en détails les différents champs. Commençons par la description d'un nœud simple avant d'introduire la notion de hiérarchie.
Nous illustrerons les définitions introduites au fur et à mesure avec l'exemple simple d'un interrupteur et de l'état de ses tensions en entrée et en sortie selon qu'il est *on* (fermé) ou pas (cf. figure 6.2).

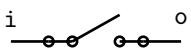


FIGURE 6.2 – Exemple Interrupteur

6.1.2.1 Description d'un nœud simple

Nous présentons les différents champs que l'on peut retrouver dans un nœud simple.

6.1.2.1.1 Les champs `state` et `init`

Le champ `state` permet de déclarer les variables d'état d'un nœud. Chaque variable est associée à un identificateur et un type. L'identificateur désignera la variable pour l'ensemble du nœud. Le type de la variable peut être de plusieurs formes :

- Un type prédéfini, tel que `bool` (`true` ou `false`) ou `integer`.
- Un intervalle d'entiers, tel que `[-5,5]`.
- Un ensemble de valeurs, tel que `couleur={jaune,vert,bleu}`.

Il est ensuite possible de composer ces types dans des tableaux. Par exemple, `matrice = bool[10][10]` définit une matrice de booléens de taille 10 par 10.

De même, AltaRica permet de définir des types structurés comme on les trouve dans de nombreux langages de programmation. Un exemple typique que l'on peut voir en figure 6.3 est celui d'un point défini par deux coordonnées entières `x` et `y`.

```
domain Point = struct
                 x : integer;
                 y : integer;
               tcurts;
```

FIGURE 6.3 – Définition d'un type structuré en AltaRica

Le type structuré est ensuite utilisable comme n'importe quel autre type. On accédera au champ `x` d'une variable `p` de type `Point` en faisant `p.x`.

Il est possible d'associer à une variable une *visibilité*. La visibilité permet de définir comment les nœuds extérieurs ont accès à la variable. Pour plus de détails, nous nous référerons aux documents cités en début de partie.

Une variable d'état est définie dans le champ `state` et sa valeur initiale est donnée dans le champ `init`. Si aucune valeur initiale n'est donnée, alors la variable sera libre dans les configurations initiales, i.e. toutes les valeurs de son domaine seront possibles. Notons qu'une configuration AltaRica représente l'état d'un nœud et correspond à une valuation de l'ensemble des variables (d'état et de flux) du nœud.

Ces variables portent l'état d'un composant et sont modifiés lors d'occurrence d'événements à travers les transitions.

Considérons l'exemple de notre interrupteur. Nous représentons ses états possibles par une variable d'état comme représenté en figure 6.4

```
node Switch
  state
    on : bool;
  init
    on := true;
  ...
edon
```

FIGURE 6.4 – Exemple Interrupteur : définition d'une variable d'état

La variable d'état `on` est une variable booléenne qui exprime que l'interrupteur est fermé lorsque la variable `on` est vraie (vaut `true`) et ouvert sinon. Au départ, nous considérons que notre interrupteur est fermé.

6.1.2.1.2 Le champ `flow`

Les variables de flux sont définies dans le champ `flow`. On donne un identificateur et un type de la même manière pour les variables de flux que pour les variables d'état (de même, on peut lui associer une visibilité et des attributs).

Le rôle d'une variable de flux est différent de celui d'une variable d'état. Les valeurs des variables de flux dépendent de celles des variables d'état et de l'environnement du nœud. C'est aussi le cas pour leur valeur initiale, qui ne rentre pas dans le champ `init`. Les relations de dépendances entre variables de flux et variables d'état sont exprimées à travers des contraintes appelées *assertions* que nous définirons par la suite.

Nous intégrons dans l'exemple de notre interrupteur deux variables de flux qui représentent les flux de tensions d'entrée (`i` pour *in*) et de sortie (`o` pour *out*) de l'interrupteur, comme nous pouvons le voir en figure 6.5.

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  ...
edon
```

FIGURE 6.5 – Exemple Interrupteur : ajout d'une variable de flux

Nous définissons les variables `i` et `o` comme pouvant prendre leur valeur entre 0 et 1, ce qui est équivalent à les définir booléennes. La valeur 0 exprime qu'aucune tension n'est présent contrairement à la valeur 1.

6.1.2.1.3 Le champ `event`

Les événements ALTARICA sont définis dans le champ `event`. Un événement représente une action qui va faire basculer le système d'une configuration dans une autre. L'événement correspondra donc à l'étiquette d'une transition.

On peut, comme pour les variables, associer à l'événement une visibilité et on peut aussi lui associer des attributs. Les attributs peuvent servir à faire des catégories. Il est possible par exemple de distinguer ainsi les événements qui sont contrôlables de ceux qui ne le sont pas en les marquant d'un attribut.

Pour notre exemple, nous pouvons définir l'événement `push` qui correspond à une action d'appui sur l'interrupteur. Le code ALTARICA de l'exemple intégrant cet événement est proposé en figure 6.6.

6.1.2.1.4 Le champ `assert`

Les assertions ALTARICA sont définies dans le champ `assert`. Elles sont exprimées à travers des expressions booléennes qui représentent des contraintes sur les variables. Toutes les configurations d'un nœud doivent satisfaire ses assertions.

Les assertions peuvent être utilisées pour décrire les relations de dépendance entre les variables d'états du système et les variables de flux, mais aussi pour décrire la valeur de variables de flux

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  ...
edon
```

FIGURE 6.6 – Exemple Interrupteur : ajout d'un événement

par rapport à d'autres variables de flux. Un exemple classique de ce dernier type d'utilisation est la définition de fonctions de transfert.

Nous ajoutons à notre exemple d'interrupteur en figure 6.7 la contrainte décrivant la valeur en sortie de l'interrupteur en fonction de son état et de la valeur d'entrée.

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  assert
    on => (i = o);
    not on => (o = 0);
  ...
edon
```

FIGURE 6.7 – Exemple Interrupteur : ajout d'une assertion

Les deux assertions `on => (i = o)` et `not on => (o = 0)` peuvent être exprimées en une seule contrainte `on => (i = o) and not on => (o = 0)`. Cela décrit le comportement des flux de tension en entrée et en sortie selon que l'interrupteur est ouvert ou fermé. Ici on exprime le fait que si l'interrupteur est `on`, alors les tensions en entrée et en sortie sont les mêmes (`i = o`). En revanche, si l'interrupteur est ouvert, la tension en sortie est nulle (`o = 0`).

Nous aurions évidemment pu écrire des formules booléennes différentes mais équivalentes à la place de celle que nous avons écrite, par exemple `on and i = o or not on and o = 0`. Le modèle ALTARICA est totalement équivalent.

ALTARICA permet aussi d'exprimer ces contraintes booléennes en utilisant les expressions `if-then-else`. Les assertions suivantes sont donc équivalentes aux deux présentées dans le code en figure 6.7 :

- `o = (if on and i = 0 then 1 else 0);`
- `(if on and i = 0 then 1 else 0) = o;`
- `(if on and i = 0 then o = 1 else o = 0);`

6.1.2.1.5 Le champ `trans`

Le dernier champ utilisé pour les nœuds simples est le champ `trans`. Il est utilisé pour décrire les conditions d'occurrence d'un événement et les changements d'état du système qu'il provoque.

Une transition a trois composantes :

- une garde qui est une expression booléenne décrivant la condition à vérifier pour que la transition puisse être tirée.
- un événement, déclaré dans le champ `event`, qui sera l'étiquette de la transition.
- une liste d'affectations qui décrit les nouvelles valeurs des variables d'états du système après le tirage de la transition. (pour plus de détails concernant la forme des affectations, on pourra se référer à [Vin03a] qui donne en annexe la grammaire complète du langage, et donc en particulier celle décrivant les affectations).

Lorsqu'un événement est déclenché, chaque transition qu'il étiquette peut être tirée si les deux conditions suivantes sont remplies :

- la garde est satisfaite.
- les assertions sont satisfiables une fois les variables d'états modifiées par les affectations.

Si au moins une transition étiquetée par un événement e est tirable, alors e est dit *déclenchable*. Une fois la transition tirée, les variables d'état sont modifiées d'après les affectations et les variables de flux ont comme valeurs possibles celles qui satisfont les assertions.

Une transition a la forme suivante :

```
garde |- événement -> affectations;
```

Un événement peut très bien étiqueter plusieurs transitions. De plus, les gardes peuvent ne pas être exclusives, AltaRica gérant le non-déterminisme.

Nous pouvons finalement intégrer les transitions à notre exemple d'interrupteur, en figure 6.8.

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  assert
    on => (i = o);
    not on => (o = 0);
  trans
    true |- push -> on := not on;
edon
```

FIGURE 6.8 – Exemple Interrupteur : modèle complet

Notre modèle, à travers la transition que nous venons d'ajouter, stipule qu'il est possible d'effectuer l'appui sur l'interrupteur (événement `push`) à n'importe quel moment (garde à `true`), et que cela le fait basculer de ouvert à fermé et inversement (`on := not on`).

Il est une nouvelle fois possible de décrire différemment le même comportement. Par exemple, nous aurions pu le décomposer en deux transitions :

- `on |- push -> on := false;`
- `not on |- push -> on := true;`

Avec l'ajout des transitions dans notre modèle d'interrupteur, nous l'obtenons de manière complète.

Nous pouvons donc désormais nous intéresser à la notion de hiérarchie.

6.1.2.2 Description de la hiérarchie

AltaRica donne la possibilité de décrire le modèle d'un système par niveau de hiérarchie.

Le principe est tout d'abord de construire les nœuds de base comme nous l'avons montré dans la partie précédente, puis de les utiliser à un niveau supérieur. Les notions qui apparaissent sont celles de *sous-nœuds*, de *connexion de flux* et de *synchronisation d'événements*.

Pour illustrer la notion de hiérarchie en AltaRica nous proposons de modéliser un système comportant deux interrupteurs en série, comme représenté sur le schéma en figure 6.9.

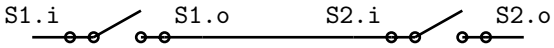


Figure 6.9 – Exemple Système d'interrupteurs

6.1.2.2.1 Sous-nœuds

La première notion à définir pour gérer la hiérarchie est la notion de sous-nœud. Elle apparaît en AltaRica dans le champ `sub`.

Un nœud AltaRica peut contenir plusieurs sous-nœud. On dit alors que c'est le nœud *parent*. Les droits de regard d'un nœud parent sur ses enfants et leurs attributs sont définis par la notion de visibilité. Nous ne regarderons pas ces notions que l'on retrouve dans [Poi00] et [Vin03a], mais considérons le comportement par défaut, à savoir :

- Un sous-nœud n'a connaissance d'aucun événement ou variable de son nœud parent (par déclaration).
- Un nœud parent peut seulement agir sur les variables de flux (à travers des contraintes) et sur les événements (à travers les synchronisations) de ses enfants.
- Un nœud peut en revanche définir ou redéfinir les valeurs initiales des variables d'état de tous ses descendants.

La création du nœud système de notre exemple peut être réalisée comme sur code AltaRica présenté en figure 6.10.

Pour notre système, nous créons deux sous-nœuds `S1` et `S2` qui représentent les deux interrupteurs du système.

6.1.2.2.2 Connexion de flux

Les flux de plusieurs sous-nœuds peuvent être connectés au niveau du nœud parent. Cela correspond typiquement au cas de communication entre sous-nœuds qui ne peut être défini qu'au niveau supérieur.

La connexion de flux se fait à travers des contraintes dans le champ `assert`. Pour vérifier l'assertion globale du système, il s'agira désormais de vérifier toutes les contraintes, à la fois celles du nœud courant et celles de tous ses descendants.

Nous pouvons illustrer la connexion de flux sur notre exemple en figure 6.11.

Nous avons connecté le flux de sortie `S1.o` du premier interrupteur `S1` au flux d'entrée `S2.i` du second interrupteur `S2`.

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  assert
    on => (i = o);
    not on => (o = 0);
  trans
    true |- push -> on := not on;
edon

node System
  sub
    S1, S2 : Switch;
  ...
edon
```

FIGURE 6.10 – Exemple Système d'interrupteurs : création de sous-nœuds

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  assert
    on => (i = o);
    not on => (o = 0);
  trans
    true |- push -> on := not on;
edon

node System
  sub
    S1, S2 : Switch;
  assert
    S1.o = S2.i;
  ...
edon
```

FIGURE 6.11 – Exemple Système d'interrupteurs : connexion de flux

6.1.2.2.3 Synchronisation d'événements

La synchronisation d'événements en ALTARICA donne la possibilité de forcer des événements à

se produire simultanément. Les synchronisations sont exprimées dans le champ `sync`.

Une synchronisation s'applique à des événements de nœuds différents et est définie dans le nœud le plus global (assez haut dans la hiérarchie pour avoir la visibilité des événements qu'il veut synchroniser).

Une synchronisation d'événements a la forme suivante :

```
< event1, event2, ..., eventN >
```

Une synchronisation n'est possible que si les modifications induites par l'occurrence de tous les événements de manière simultanée sont réalisables. Cela signifie que la modification de l'ensemble des variables d'état et la satisfaction de l'assertion globale sont possibles.

Un autre type de synchronisation, moins contrainte que les synchronisations fortes, est possible : les synchronisations faibles.

Lorsqu'un événement d'une synchronisation forte n'est pas déclenchable, la synchronisation ne peut pas avoir lieu. Dans les synchronisations faibles, certains événements, précédés d'un symbole " ?", ne sont pas bloquants pour la synchronisation. Cela se traduit en deux clauses :

- Un événement qui est marqué n'est pas nécessaire pour la synchronisation.
- Si un événement marqué est déclenchable lors d'une synchronisation, alors il peut être ou ne pas être déclenché.

Considérons la synchronisation faible suivante :

```
< a, b?, c? >
```

Cette synchronisation stipule que les événements `b` et `c` sont non bloquants pour la synchronisation. On peut réécrire la synchronisation faible en plusieurs synchronisations fortes :

```
  < a >
 < a, b >
 < a, c >
< a, b, c >
```

Avec le langage AltaRica, il est possible en plus de donner une contrainte sur le nombre d'événements qui doivent être synchronisés dans une synchronisation faible. Prenons l'exemple suivant pour illustrer cela :

```
< a, b?, c? > >= 1
```

Ici nous forçons à ce qu'au moins un des deux événements marqués (`b` ou `c`) soit déclenché. La synchronisation avec cette contrainte supplémentaire peut être réécrite en cet ensemble de synchronisations fortes :

```
 < a, b >
 < a, c >
< a, b, c >
```

Nous pouvons finalement considérer notre exemple de système d'interrupteurs auquel nous avons intégré la notion de synchronisation en figure 6.12.

À travers la synchronisation forte `< S1.push, S2.push >;` nous obligeons les événements `push` des deux interrupteurs à se produire simultanément.

Ce comportement, qui ne correspond pas à une interaction classique de deux interrupteurs, est définie à des fins d'illustration et n'a pas de réalité particulière.

```
node Switch
  state
    on : bool;
  init
    on := true;
  flow
    i, o : [0,1];
  event
    push;
  assert
    on => (i = o);
    not on => (o = 0);
  trans
    true |- push -> on := not on;
edon

node System
  sub
    S1, S2 : Switch;
  assert
    S1.o = S2.i;
  sync
    < S1.push, S2.push >;
edon
```

FIGURE 6.12 – Exemple Système d'interrupteurs : Modèle complet

6.1.2.3 Automate de comportement AltaRica

Dans cette dernière section de la partie sur le langage AltaRica, nous allons décrire dans les grandes lignes l'automate de comportement du système modélisé.

Le langage AltaRica, comme la majorité des langages de modélisation formelle, calcule un objet formel à partir du fichier AltaRica décrivant le comportement du système. Dans le cas d'AltaRica, l'objet manipulé, qui représente le graphe d'accessibilité du modèle, peut être décrit comme un automate à contraintes [PR99].

Avant de définir ce qu'est un automate à contraintes, nous introduisons ou rappelons quelques notations. Pour la suite, $T(V)$ et $BF(V)$ représenteront respectivement l'ensemble des termes et l'ensemble des formules booléennes construites à partir d'un ensemble de variables V. De plus, $\mathcal{F}(A, B)$ désigne l'ensemble des fonctions de A vers B.

Définition 46 (Automate à contraintes) *Un* automate à contraintes $\mathcal{A} = \langle V_S, V_F, E, T, A, I \rangle$ *est un n-uplet où :*

- V_S *et* V_F *sont des ensembles de variables disjoints. Les éléments de* V_S *(resp.* V_F*) sont les variables* d'état *(resp.* de flux*).*
- E *est l'ensemble des* événements.
- $T \subseteq BF(V_S \cup V_F) \times E \times \mathcal{F}(V_S, T(V_S \cup V_F))$ *est l'ensemble des* macro-transitions. *Si* $\langle g, e, \alpha \rangle$ *appartient à* T*,* g *est appelée la* garde*,* e *l'*événement *et* α *l'*affectation *de la transition.* α *n'est pas nécessairement totale.*
- $A \subseteq BF(V_S \cup V_F)$ *contient l'ensemble des* assertions *du modèle i.e. les invariants qui doivent être satisfaits par les affectations de variables.*
- $I \subseteq BF(V_S)$ *contient la* contrainte initiale.

Dans le cas de l'automate à contraintes, une configuration de $\mathcal{A}$ est une valuation de ses variables qui satisfait A. On obtient ainsi Q, l'ensemble des configurations de $\mathcal{A}$.

Un automate à contraintes est donc un système de transitions étiquetées. On peut le rapprocher des définitions données en section 3.1. En particulier, les notions de successeurs et d'accessibles sont totalement applicables ici.

Nous n'entrerons pas plus dans les détails concernant la représentation formelle du modèle. Pour approfondir la notion d'automate à contraintes en AltaRica, on pourra se référer au rapport de recherche [GPKV11] qui décrit tout cela plus en détail.

Nous pouvons maintenant représenter l'automate à contraintes qui correspond au comportement de notre exemple de système d'interrupteurs (cf. figure 6.12).

Représentons tout d'abord l'automate à contraintes pour un interrupteur en figure 6.13. Pour cela, nous représentons les configurations comme un triplet (on, i, o) qui contient les valuations des variables on, i et o.

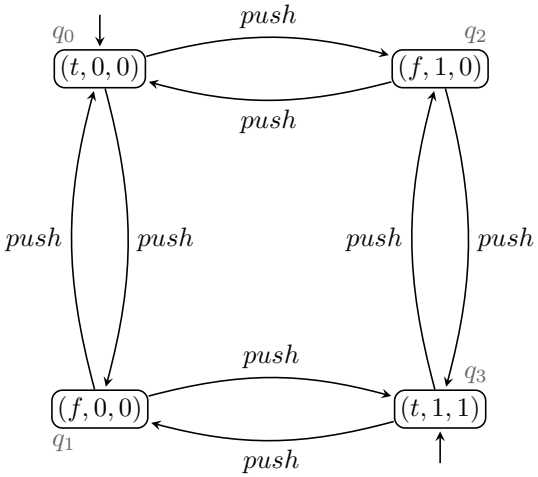


FIGURE 6.13 – Automate de comportement pour un interrupteur

Nous représentons l'automate de comportement du système d'interrupteurs en figure 6.14. Pour cela, une configuration de l'automate correspond à deux triplets $\{(S1.on, S1.i, S1.o), (S2.on, S2.i, S2.o)\}$.

6.1.3 L'outil ARC

L'outil ARC (**A**lta**R**ica **C**hecker) est un outil de calcul et d'analyse de modèle AltaRica basé sur du model-checking [CGP99].

Le but d'ARC est de rassembler un ensemble d'outils pour l'analyse et la compilation de modèles AltaRica. ARC est d'ailleurs le fruit de l'union de deux suites d'outils : AltaTools et Mec 5 [GV04].

Nous pouvons lister quelques utilisations possibles de l'outil pour donner une idée de ce qu'il propose :

- Support du langage AltaRica.

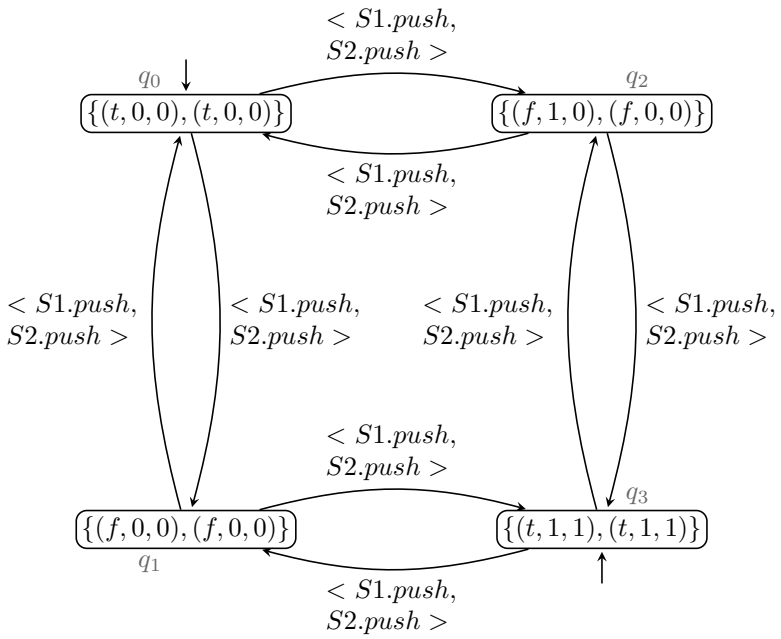


FIGURE 6.14 – Automate de comportement pour le système d'interrupteurs

- Intégration de l'outil ACHECK présent dans la suite AltaTools qui permet principalement de gérer l'automate de comportement d'un modèle ALTARICA de manière symbolique ou explicite.
- Model-checking à partir de diverses logiques (logique de Dicky [Dic86], CTL*).
- Intégration des spécifications de type MEC 5, qui est un model-checker qui gère aussi les modèles ALTARICA.
- Simulateur graphique.
- Traducteurs du langage ALTARICA vers LUSTRE [PG06].

Plus récemment, un logiciel appelé ALTARICA-STUDIO et basé sur ARC propose de manipuler des modèles ALTARICA de manière graphique.

L'outil est très complet et ce n'est pas le but ici de le présenter de manière détaillée. De plus, une très bonne description en est faite dans le document *Arc Handbook*[2]. Nous allons donc présenter seulement quelques possibilités de l'outil qui correspondent à celles que nous utiliserons par la suite dans ce document.

6.1.3.1 Calculs sur les modèles

Nous allons présenter dans cette partie comment effectuer des calculs sur les modèles en utilisant les commandes ACHECK.

Les commandes ACHECK sont appliquées sur un nœud. Pour cela, les commandes que l'on souhaite appliquer sont contenues dans une structure `with ... do ... done`. Entre les mots `with` et `do`, nous listons les nœuds auxquels on veut voir appliquées les commandes. Entre `do` et `done` se retrouvent les commandes. On peut voir une illustration de cela en figure 6.15.

Nous pouvons en outre préciser à l'outil si nous souhaitons effectuer les calculs sur le modèle de manière symbolique (mot-clé `symbolically`) ou de manière explicite (mot-clé `exhaustively`). On place alors le mot-clé correspondant juste après le `do`.

2. `ftp://altarica.labri.fr/pub/publications/arc-handbook.pdf`

```
with Switch do
  commands;
done
```

FIGURE 6.15 – Structure pour les commandes ACHECK

Par défaut, les calculs sont fait de manière symbolique.

Dans la suite de cette partie, nous allons présenter une partie des commandes qu'il est possible de réaliser avec ARC. Plus précisément, nous ne présenterons que des commandes relatives à l'utilisation des commandes ACHECK. ARC compte de nombreuses commandes que nous ne présenterons pas. Pour avoir une idée des possibilités globales de l'outil, on pourra donc se référer à son manuel.

6.1.3.1.1 Ensembles pré-calculés

Les objets du modèle que nous manipulons lorsque nous utilisons ARC pour faire des calculs sont des ensembles de configurations et des ensembles de transitions. Nous trouvons des ensembles pré-calculés pour les deux.

Il faut noter que certains ensembles ne sont disponibles que lorsque nous effectuons des calculs en utilisant la méthode symbolique, d'autres uniquement avec la méthode explicite, et enfin d'autres avec les deux. Ceci est dû à l'encodage utilisé qui est différent pour les deux méthodes. Nous préciserons pour chacun des ensembles pré-calculés présenté pour quelle méthode il est disponible.

Nous ne serons une nouvelle fois pas exhaustifs, et nous préférerons nous reporter au manuel ARC pour avoir l'ensemble des ensembles pré-calculés. Les définitions données ci-après sont d'ailleurs traduites depuis ce manuel.

6.1.3.1.1.1 Ensembles de configurations

any_c Cet ensemble contient toutes les configurations valides du nœud considéré. Les configurations vérifient l'assertion globale du nœud. À noter que les configurations ne sont pas forcément accessibles depuis les états initiaux.
Utilisation : *Symbolique.*

any_s Cet ensemble contient les configurations accessibles depuis les états initiaux.
Utilisation : *Symbolique, Explicite.*

initial Cet ensemble contient les configurations initiales, i.e. les valuations des variables qui vérifient à la fois la contrainte `init` et l'assertion globale.
Utilisation : *Symbolique, Explicite.*

6.1.3.1.1.2 Ensembles de transitions

any_t Cet ensemble contient les transitions entre les configurations accessibles (éléments de *any_s*).
Utilisation : *Symbolique, Explicite.*

any_trans Cet ensemble est la relation de transitions où les pre- et post-conditions ne sont pas appliquées..
Utilisation : *Symbolique.*

not_ deterministic
Cet ensemble identifie les transitions non déterministes.
Utilisation : *Explicite.*

self Cet ensemble identifie les boucles élémentaires, i.e. les transitions dont la source et la destination sont la même configuration..
Utilisation : *Symbolique, Explicite.*

6.1.3.1.2 Opérateurs

Il est possible d'effectuer des opérations avec les ensembles de configurations et de transitions. Pour cela, AltaRica fournit un ensemble d'opérateurs. Nous en présentons une partie issue du manuel.

X_1 `and` X_2
X_1 `&` X_2 Calcule l'intersection des deux ensembles X_1 et X_2. Les deux ensembles doivent être du même type.
Utilisation : *Symbolique, Explicite.*

X_1 `or` X_2
X_1 `|` X_2 Calcule l'union des deux ensembles X_1 et X_2. Les deux ensembles doivent être du même type.
Utilisation : *Symbolique, Explicite.*

X_1 `-` X_2 Calcule la différence des deux ensembles X_1 et X_2. Les deux ensembles doivent être du même type.
Utilisation : *Symbolique, Explicite.*

`[`ϕ`]` Cet opérateur renvoie les valuations de variables qui satisfont ϕ, où ϕ est une expression AltaRica booléenne sur les variables du nœud considéré.
Selon la méthode de calcul, le résultat n'a pas la même sémantique. Lorsque la méthode est explicite, les valuations résultats sont, par construction, des configurations accessibles. Avec l'encodage symbolique, les valuations ne sont pas contraintes et n'appartiennent pas forcément à *any_ c*.
Utilisation : *Symbolique, Explicite.*

`attribute` *att*
`attribute` *att* renvoie l'ensemble des transitions étiquetées par un événement d'attribut *att*.
Si la méthode de calcul est symbolique, alors les transitions renvoyées ne sont pas contraintes par les pre- et post-conditions, i.e. `attribute` *att* est un sous-ensemble de *any_ trans*.
Utilisation : *Symbolique, Explicite.*

`label` e e est l'identificateur d'un événement. `label` e renvoie l'ensemble des transitions dont l'événement global contient e.
Si la méthode de calcul est symbolique, alors les transitions renvoyées ne sont pas contraintes par les pre- et post-conditions, i.e. `label` e est un sous-ensemble de *any_ trans*.
Utilisation : *Symbolique, Explicite.*

`not` X Cet opérateur renvoie le complément de l'ensemble X, qui est soit un ensemble de configurations, soit un ensemble de transitions. Si la méthode de calcul est explicite, le complément de l'ensemble de configurations (resp. transitions) est fait dans les configurations accessibles *any_ s* (resp. transitions *any_ t*). Dans le cas d'une méthode symbolique, le complément est pris dans l'ensemble des configurations et transitions possibles, i.e. *any_ c* ou *any_ trans*.
Utilisation : *Symbolique, Explicite.*

`reach(`S`,`T`)`
: Cet opérateur retourne l'ensemble des configurations valides qui sont accessibles depuis S en utilisant des transitions de T. Notons que dans le cas d'une méthode de calcul symbolique, les configurations calculées ne sont pas forcément accessibles depuis l'état initial.
Utilisation : *Symbolique, Explicite.*

`rsrc(`S`)`
: Cet opérateur retourne l'ensemble des transitions qui partent d'une configuration de S. Notons que dans le cas d'une méthode de calcul symbolique, les transitions ne sont pas contraintes par les pre- et post-conditions.
Utilisation : *Symbolique, Explicite.*

`rtgt(`S`)`
: Cet opérateur retourne l'ensemble des transitions qui mènent dans une configuration de S. Notons que dans le cas d'une méthode de calcul symbolique, les transitions ne sont pas contraintes par les pre- et post-conditions.
Utilisation : *Symbolique, Explicite.*

`src(`T`)`
: Cet opérateur retourne l'ensemble des configurations qui sont la source d'au moins une transition de T. Notons que les configurations ne sont pas contraintes par les assertions.
Utilisation : *Symbolique, Explicite.*

`tgt(`T`)`
: Cet opérateur retourne l'ensemble des configurations qui sont la cible d'au moins une transition de T. Notons que les configurations ne sont pas contraintes par les assertions.
Utilisation : *Symbolique, Explicite.*

6.1.3.1.3 Commandes

Nous présentons finalement quelques unes des commandes ACHECK fournies par ARC. Elles permettent de manipuler les ensembles de configurations et de transitions calculés à travers les deux sections précédentes.

`display(`id_1`,...,`id_n`)`
: Cette commande liste tous les éléments des ensembles id_1 à id_n. Dans le cas d'une méthode symbolique, les ensembles sont calculés à la demande.
Utilisation : *Symbolique, Explicite.*

`dot(`S`,`T`)`
: Cette commande génère, au format `dot`, le graphe d'accessibilité restreint à l'ensemble de configurations S et l'ensemble de transitions T.
Utilisation : *Symbolique, Explicite.*

`show(`id_1`,...,`id_n`)`
: Cette commande affiche la cardinalité des ensembles id_1 à id_n.
Utilisation : *Symbolique, Explicite.*

`test(`X`,`n`)`
: Cette commande est utilisée pour vérifier que la cardinalité de l'ensemble X est n. Elle affiche simplement le résultat du test.
Utilisation : *Symbolique, Explicite.*

Avec ce que nous avons dans ces sections dédiées aux calculs sur les modèles, nous pouvons donner un exemple de fichier de type ACHECK, en figure 6.16, qui effectue des calculs sur les modèles de nos interrupteur et système de deux interrupteurs.

```
with Switch, System do
  // Affiche les configurations.
  display(any_s);
  // Donne le nombre d'états initiaux.
  show(initial);
  // Génère l'automate à contraintes du modèle
  dot(any_s,any_t) > '$NODENAME.dot';
  // Automate à contraintes du modèle, sans epsilon.
  dot(any_s,any_t-epsilon) > '$NODENAME_sans_eps.dot';
done
```

FIGURE 6.16 – Spécifications pour les nœuds `Switch` et `System`

Dans cet exemple, pour chacun des nœuds `Switch` et `System`, nous effectuons plusieurs calculs.
Le premier calcul affiche simplement les configurations initiales. Nous obtenons alors, à travers ARC, les résultats suivants :

- Pour la commande qui affiche les configurations :
  - Pour `Switch` :
    ```
    set 'any_s' contains
    o = 0, on = false, i in [0, 1]
    o = 0, on = true, i = 0
    o = 1, on = true, i = 1
    ```
  - Pour `System` :
    ```
    set 'any_s' contains
    S2.o = 0, S2.on = false, S1.o = 0, S1.on = false, S1.i in [0, 1]
    S2.o = 0, S2.on = true, S1.o = 0, S1.on = true, S1.i = 0
    S2.o = 1, S2.on = true, S1.o = 1, S1.on = true, S1.i = 1
    ```
- Pour la commande qui donne le nombre d'états initiaux :
  - Pour `Switch` :
    ```
    /*
    * Properties for node : Switch
    * # state properties : 1
    *
    * initial = 2
    *
    * # trans property : 0
    *
    */
    ```
  - Pour `System` :
    ```
    /*
    * Properties for node : System
    * # state properties : 1
    *
    * initial = 2
    *
    * # trans property : 0
    *
    */
    ```

En outre, à travers la commande `dot`, nous avons généré plusieurs graphes.

Le premier graphe que nous générons contient toutes les configurations et toutes les transitions. Le résultat est écrit dans le fichier `$NODENAME.dot`, i.e. `Switch.dot` pour `Switch` et `System.dot` pour `System`. Les deux graphes sont représentés en figures 6.17 et 6.18.

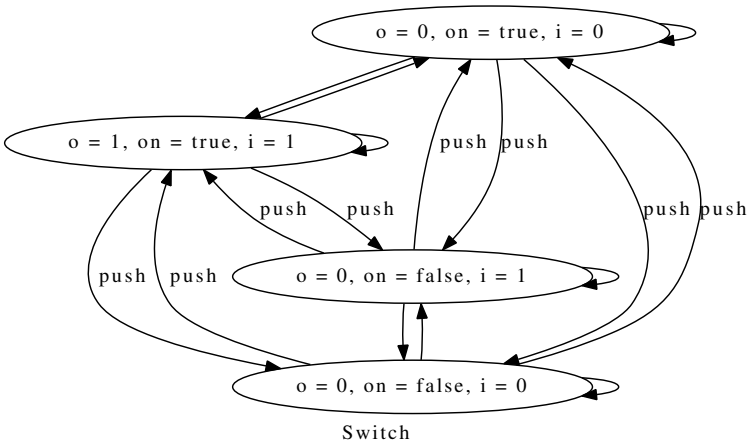


FIGURE 6.17 – Automate de comportement pour `Switch`

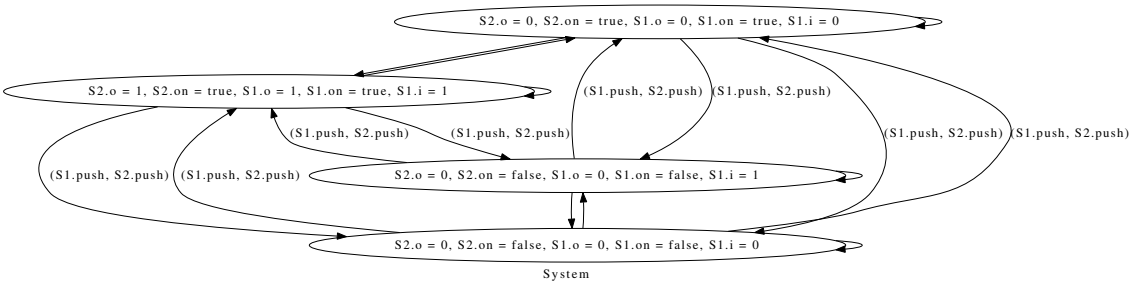


FIGURE 6.18 – Automate de comportement pour `System`

Dans ces graphes nous voyons apparaître des transitions non étiquetées. Ce sont ce que l'on appelle en ALTARICA des *epsilon-transitions*. ϵ est un événement ALTARICA qui est ajouté à chaque nœud et qui signifie qu'aucun événement ne se passe, i.e. qu'aucune affectation n'a lieu. Ce que l'on voit ici, c'est que le système étant non déterministe au niveau de ses variables de flux dans certains cas, les ϵ-transitions peuvent relier deux configurations différentes.

Nous n'en dirons pas plus concernant les ϵ-transitions, leur raison d'être et leurs effets, nous nous reporterons pour cela aux documents déjà cités [Poi00, Vin03a] qui décrivent le langage de manière exhaustive.

La deuxième commande `dot` calcule le même graphe mais duquel on a enlevé les ϵ-transitions. Nous obtenons les deux nouveaux automates de comportement présentés en figures 6.19 et 6.20.

Les graphes que nous obtenons sont exactement ceux que nous avons présentés en figures 6.13 et 6.14 pages 157–158, dans la partie concernant l'automate à contraintes ALTARICA (6.1.2.3).

Nous pouvons cependant remarquer que toutes les variables valuées n'apparaissent pas dans tous les nœuds du graphe. Cela n'est le cas que lors de l'utilisation de la méthode symbolique.

Dans le nœud le plus en haut du graphe en figure 6.19 par exemple, nous remarquons que la valeur de la variable `o` n'est pas notée. Les variables dont la valeur ne change pas par rapport tous ses voisins n'apparaissent pas dans les nœuds. Dans le cas que nous observons, la variable `o` est à 0 dans le nœud considéré ainsi que dans ses deux voisins.

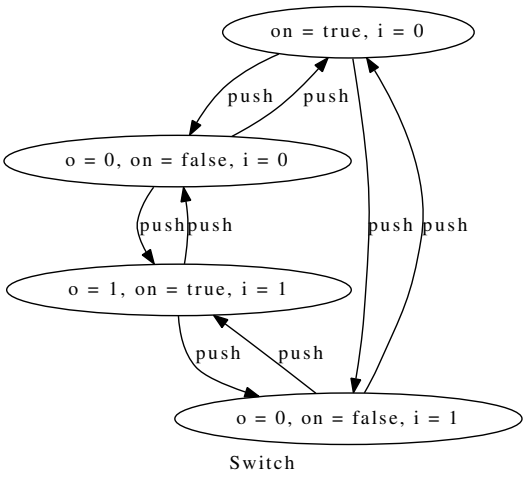


FIGURE 6.19 – Automate de comportement pour `Switch` sans les ϵ-transitions

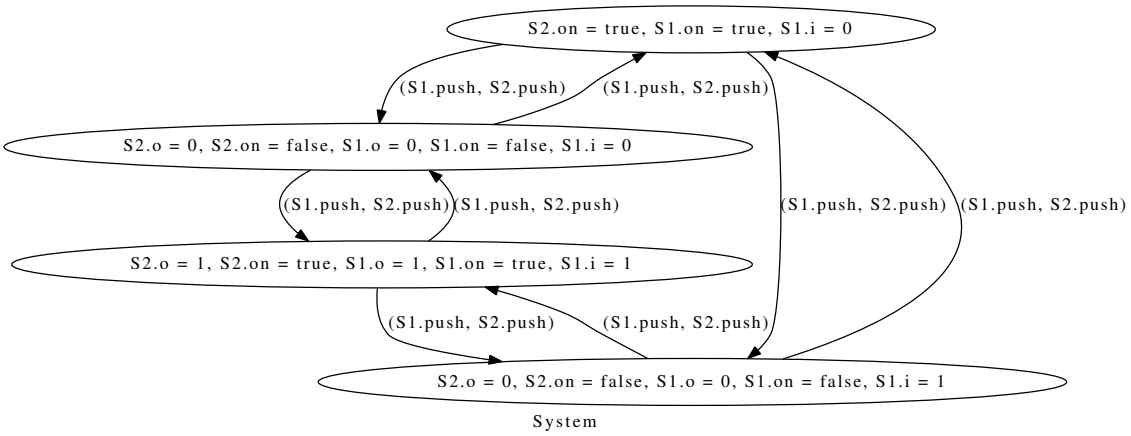


FIGURE 6.20 – Automate de comportement pour `System` sans les ϵ-transitions

Il est important de comprendre cela. Il ne faut pas penser que la variable n'apparaissant pas a une valeur indéterminée. Chaque nœud du graphe obtenu par la commande `dot` correspond à une configuration qui est, par définition, totalement valuée.

Nous avons vu les opérations disponibles à travers les commandes de type ACHECK. La prochaine étape est donc de présenter comment interagir avec ARC, et ceci pour notamment qu'il effectue les calculs sur les modèles.

6.1.3.2 Interaction avec ARC

L'interaction avec ARC se fait de trois façons :

- En utilisant des fichiers passés en paramètres à l'outil ARC.
- Directement avec l'invite de commande d'ARC.
- À travers ALTARICA-STUDIO.

Par la suite, nous pourrons illustrer nos propos avec des exemples de commandes. La partie écrite en *`italique`* correspondra à la réponse du système alors que la partie `non italique` correspondra aux commandes de l'utilisateur.

6.1.3.2.1 Fichiers en paramètre

La première méthode pour interagir avec ARC et analyser les modèles consiste à manipuler des fichiers puis à les fournir à ARC.

Le premier fichier à fournir à ARC est le modèle AltaRica. L'outil le charge alors et vérifie sa cohérence syntaxique, remontant des alertes dans le cas où le modèle ne serait pas écrit correctement.

Si on considère `modele.alt` le fichier contenant notre modèle AltaRica, il suffit de lancer ARC avec le chemin du fichier en paramètre pour le charger.

```
$ arc modele.alt
...
Loading file 'modele.alt'.
arc>
```

Une fois le modèle chargé, il est possible d'effectuer des calculs. On peut les spécifier dans un deuxième fichier, dit *de spécification*. Dans nos exemples, nous attribuerons l'extension "*.ack*" à ce type de fichier.

Pour le charger, nous pouvons le passer en paramètre avec le fichier modèle.

```
$ arc modele.alt modele.ack
...
Loading file 'modele.alt'.
Loading file 'modele.ack'.
arc>
```

Cette opération de chargement dans ARC est faisable avec la commande ARC `load`. On peut donc aussi charger le modèle et le fichier de spécifications en les passant en paramètres lors du lancement de ARC, ou bien une fois ARC lancé, avec la commande `load`.

```
$ arc
arc>load modele.alt modele.ack
Loading file 'modele.alt'.
Loading file 'modele.ack'.
arc>
```

Il est aussi possible de charger d'abord le modèle puis le fichier par deux commandes successives. Mais l'inverse n'est pas possible puisque les analyses contenues dans le fichier de spécifications sont exécutées immédiatement, et qu'ARC ne connaît pas le modèle à ce moment-là.

6.1.3.2.2 Ligne de commande avec l'outil ARC

Une deuxième méthode pour interagir avec arc est d'utiliser directement ARC en ligne de commandes. Par là, il faut comprendre qu'au lieu d'écrire les spécifications dans un fichier et de le charger ensuite dans ARC, il est possible de décrire ces spécifications directement dans ARC avec la commande `eval`.

Pour cela, il suffit de lancer la commande `eval`, de donner les spécifications sous la forme `with ... do ... done` et de marquer la fin de la saisie avec le symbole `^D`, qui correspond souvent à `CTRL-D` dans les systèmes de type UNIX.

La méthode est donc très similaire à la première présentée. On peut reprendre l'exemple de fichier de spécifications de la figure 6.16 :

```
$ arc
arc>load modele.alt
Loading file 'modele.alt'.
arc>eval
eval>with Switch, System do
eval>display(any_s);
eval>show(initial);
eval>dot(any_s,any_t);
eval>dot(any_s,any_t - epsilon);
eval>done
eval>[... ]
arc>
```

En faisant la commande de fin de fichier `^D` après le `done`, on obtient les réponses d'ARC aux commandes entrées. C'est ce que nous avons représenté par *[...]*, et qui correspond aux résultats obtenus en section 6.1.3.1.3 après l'exécution des spécifications.

6.1.3.2.3 AltaRica-Studio

AltaRica-Studio est un outil doté d'une interface graphique permettant de visualiser et d'analyser les modèles. Il est basé sur ARC et automatise certaines commandes récurrentes lors de la manipulation de modèles AltaRica.

AltaRica-Studio se présente comme montré sur la figure 6.21.

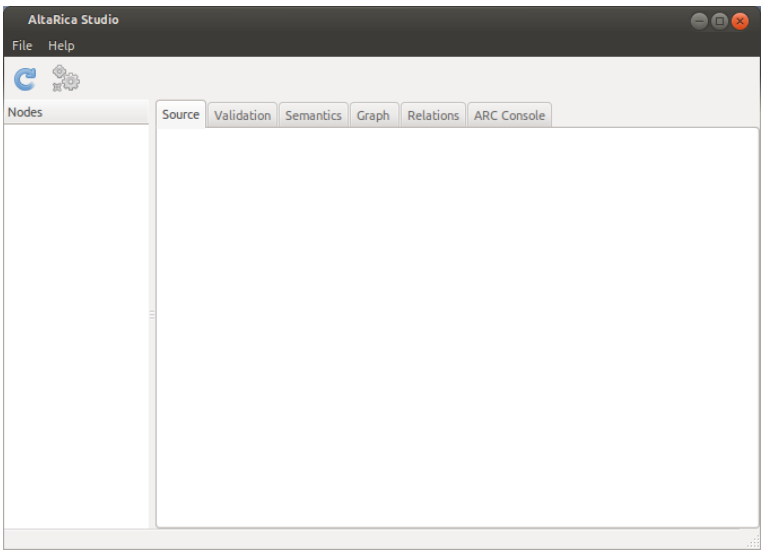


Figure 6.21 – AltaRica-Studio : vue d'ensemble

AltaRica-Studio présente plusieurs volets :

- Un menu permettant de charger un modèle, afficher des informations sur l'outil, ou le quitter.
- Un panneau à gauche comprenant la liste des nœuds chargés.
- Deux boutons, l'un permettant de rafraîchir la liste des nœuds, l'autre permettant de simuler pas à pas le comportement d'un nœud.
- Un panneau à droite comprenant plusieurs onglets.

Les onglets du panneau droit représentent plusieurs calculs et services que permet ARC.

6.1.3.2.3.1 Source

L'onglet `Source` permet d'afficher le code source du nœud sélectionné dans la liste. Nous pouvons voir le code source de notre interrupteur en figure 6.22.

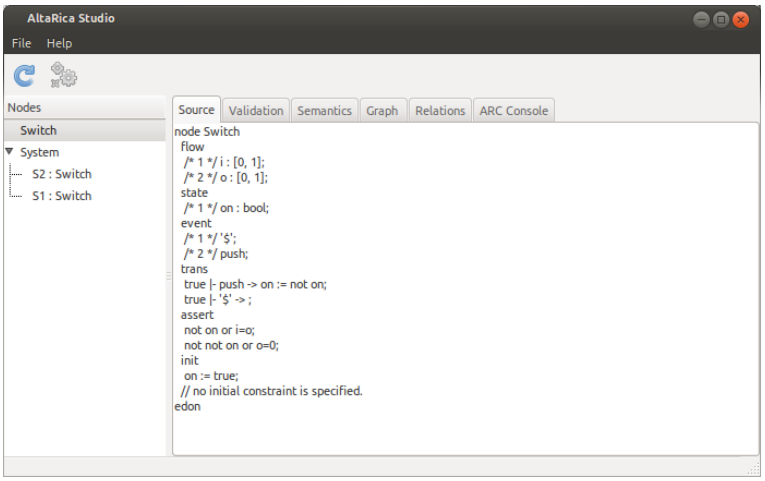


FIGURE 6.22 – ALTARICA-STUDIO : Source d'un nœud

On remarque que quelques informations ont été ajoutées lors du chargement du modèle par l'outil :

- Les identificateurs du modèle ont été numérotés.
- L'événement ϵ, ici représenté par le symbole $ a été ajouté, ainsi que sa transition.
- Les assertions ont été réécrites.

6.1.3.2.3.2 Validation

L'onglet `Validation` donne les résultats d'un ensemble de tests classiques faits sur un nœud. C'est l'équivalent de la commande ARC `validate`. Nous voyons les résultats de cette commande sur le nœud `Switch` en figure 6.23.

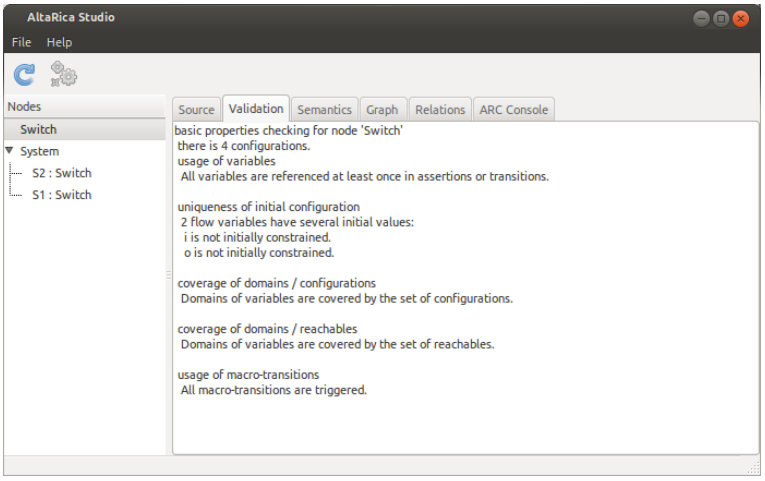


FIGURE 6.23 – ALTARICA-STUDIO : Validation d'un nœud

Les propriétés basiques effectuées dans cet onglet sont :

- Combien y a-t-il de configurations ?
- Les variables déclarées sont-elles bien toutes utilisées dans le modèle ?
- L'état initial est-il unique ?
- Les variables prennent-elles toutes les valeurs de leur domaine dans l'ensemble des configurations ?
- Les variables prennent-elles toutes les valeurs de leur domaine dans l'ensemble des configurations accessibles ?
- Toutes les transitions sont-elles tirables ?

Lorsqu'une de ces propriétés n'est pas vérifiée, ARC présente une explication.

6.1.3.2.3.3 Semantics

L'onglet `Semantics` présente le graphe de comportement du modèle sélectionné. Nous retrouvons celui du modèle d'interrupteur en figure 6.24.

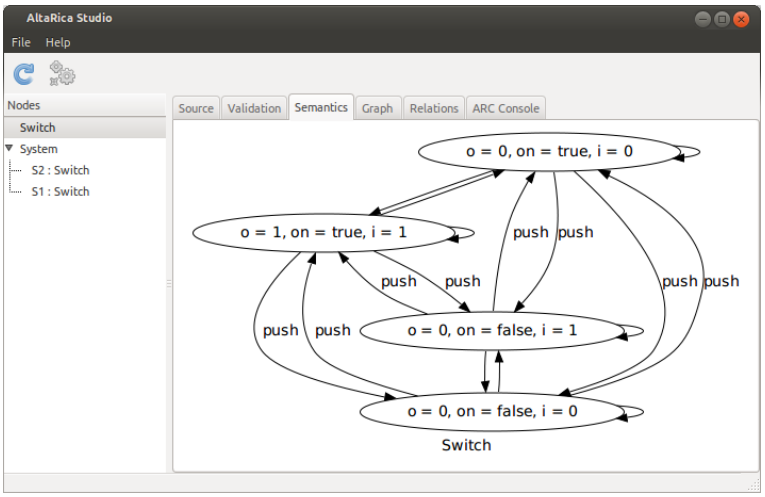


FIGURE 6.24 – ALTARICA-STUDIO : Graphe de comportement d'un nœud

Le graphe affiché correspond à la commande ACHECK `dot(any_s,any_t)`.

6.1.3.2.3.4 Graph

L'onglet `Graph` permet d'utiliser des commandes ACHECK qui génèrent des graphes. C'est le cas typiquement de la commande `dot` que nous avons présentée. Nous pouvons par exemple générer le graphe de comportement qui ne contient pas les ϵ-transitions, comme nous l'avons fait dans le fichier de spécifications. Le résultat est visible en figure 6.25.

6.1.3.2.3.5 Relations

L'onglet `Relations` liste l'ensemble des ensembles de configurations et de transitions déjà calculés. Sur la figure 6.26, nous pouvons voir les résultats suite aux précédents calculs.

6.1.3.2.3.6 ARC Console

Finalement, le dernier onglet ARC `Console` permet de gérer les commandes ARC et d'afficher leurs résultats. C'est l'équivalent de l'invite de commandes *`arc>`* qui apparaît au lancement d'ARC. Ici, nous pouvons par exemple charger le fichier de spécifications (Switch.ack) que nous avons présenté en figure 6.16. Le résultat est présenté en figure 6.27.

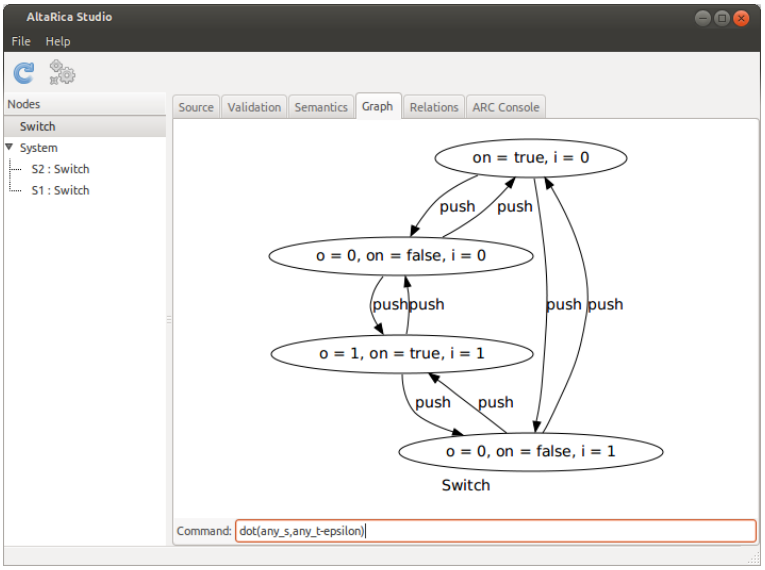


FIGURE 6.25 – ALTARICA-STUDIO : Graphe de comportement personnalisé d'un nœud

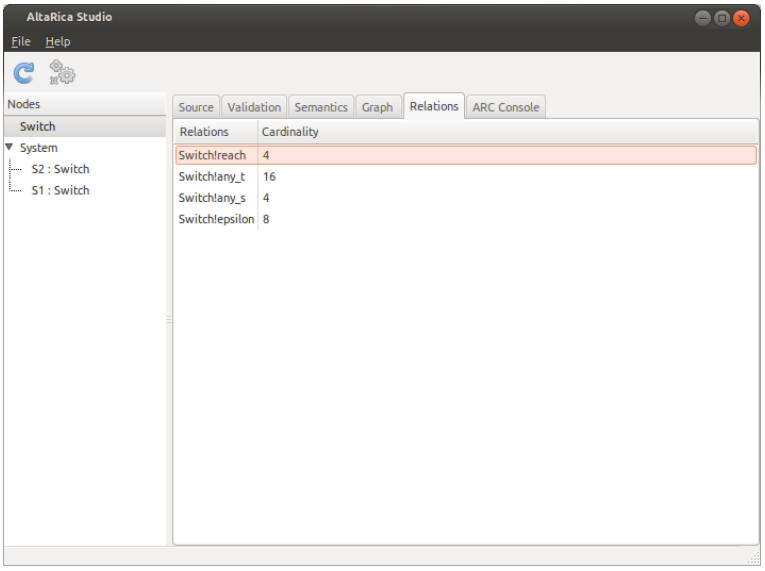


FIGURE 6.26 – ALTARICA-STUDIO : Relations calculées

6.2 Correspondance des définitions de performances de diagnostic dans ALTARICA

Dans cette section nous expliquons comment mesurer l'ensemble des définitions et métriques de performances de diagnostic avec ARC.

Tout d'abord, ce qu'il faut remarquer, c'est que les définitions et métriques de diagnostic sont définies en s'appuyant sur un petit nombre de définitions de base. Ces dernières sont :

- La notion de successeurs.
- La catégorisation des événements selon leur contrôlabilité.

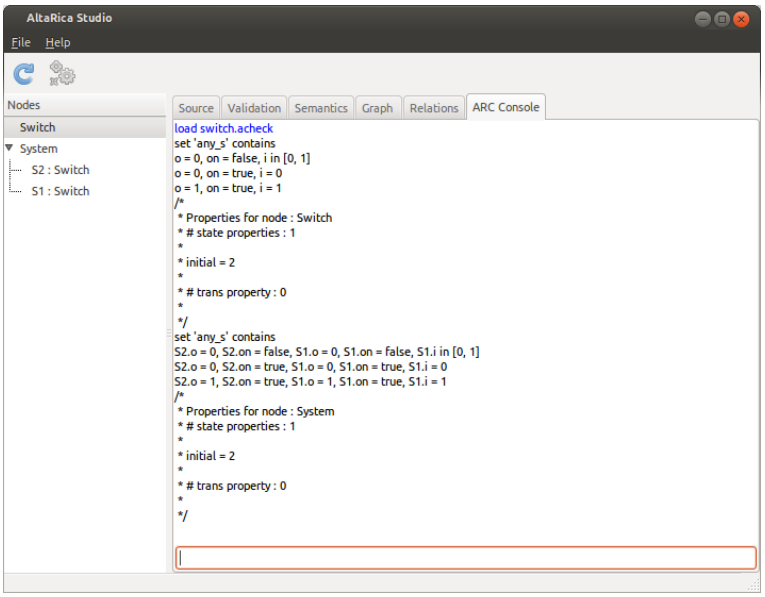


FIGURE 6.27 – ALTARICA-STUDIO : Console ARC

- La notion de configuration à la source d'une transition étiquetée par un événement, qu'utilise la notion de perceptibilité.

Toutes les autres définitions et métriques sont construites à partir de ces briques de base. Montrons donc comment les obtenir à l'aide d'ARC.

6.2.1 Successeurs

Soient les définitions données en section 3.1. Considérons le système $S = \langle Q,Q_i,E,\rightarrow\rangle$.
Nous rappelons la définition de la fonction successeurs :

$$succ(Q', E') = \{q' \in Q \mid \exists (q, e) \in Q' \times E', q \xrightarrow{e} q'\}$$

Nous avons donc en entrée un ensemble de configurations de départ Q' et un ensemble d'événements E'. Considérons que $E' = \{e1, \ldots, eN\}$.
L'idée est d'utiliser l'opérateur ACHECK `tgt` qui prend en paramètre un ensemble de transitions T. La première étape consiste donc à obtenir l'ensemble des transitions qui sont étiquetées par un événement de E'. La commande ACHECK qui permet de faire ce calcul et de l'enregistrer dans l'objet `t_Ep` est :

```
t_Ep := (label e1 | ... | label eN) & any_t;
```

Les `...` sont évidemment remplacés par les expressions `label e` pour tous les événements e de E'. L'intersection avec any_t permet de s'assurer, dans le cas d'utilisation de la méthode symbolique, que les éléments de l'ensemble de transitions résultat correspondent bien à des transitions existantes (`label` ne contraint pas les transitions obtenues par les pre- et post-conditions).
Pour nous assurer que nous partons bien de configurations dans Q', il ne faut conserver que les transitions étiquetées par un événement de E' qui ont comme configuration source une configuration de Q'. Nous réalisons ceci en utilisant la commande `rsrc` :

```
t_Ep_Qp := t_Ep & rsrc(Qp) & any_t;
```

`rsrc(Qp)`, où `Qp` représente Q', calcule l'ensemble des transitions qui sont tirables depuis Q' ; on intersecte avec any_t pour la même raison que précédemment. On applique ensuite l'opération

d'intersection avec `t_Ep` pour ne conserver que les transitions étiquetées par un événement de E' et qui sont tirables depuis une configuration de Q'.

Il nous reste désormais à trouver les configurations cibles des transitions que nous venons de calculer. Pour cela nous utilisons l'opérateur `tgt` :

```
succ_Ep_Qp := tgt(t_Ep_Qp) & any_c;
```

Nous obtenons ainsi les successeurs d'un ensemble d'événements E' à partir de Q' ; cette fois-ci, nous intersectons avec any_c pour nous assurer, en cas de calcul sur une structure symbolique, que les configurations obtenues existent dans le système.

Les fonctions $succ^n$, $reach$ et $reach^n$ sont directement calculables à partir de $succ$. Néanmoins, ARC nous fournit un opérateur dédié pour le calcul des configurations accessibles ; nous pouvons donc donner un équivalent de la fonction $reach$ pour un ensemble de configurations de départ Q' et un ensemble d'événement E' qui utilise l'opérateur `reach` :

```
reach_Ep_Qp := reach(Qp, t_Ep) & any_c;
```

6.2.2 Contrôlabilité des événements et notions dérivées

Pour pouvoir gérer la notion de contrôlabilité, nous allons utiliser les attributs en AltaRica.

Nous savons quels événements sont contrôlables et lesquels ne le sont pas. Lors de la construction du modèle, nous disons pour chaque événement s'il est contrôlable ou pas à travers un attribut. Dans le cas où $e1$ est un événement contrôlable et où $e2$ ne l'est pas, nous obtenons le code AltaRica suivant dans le champ de déclaration des événements `event` :

```
event
  e1 : cont;
  e2 : incont;
```

Pour récupérer les transitions étiquetées par les événements contrôlables, il suffit alors d'utiliser la commande Acheck `attribute` :

```
  t_c := (attribute cont) & any_t;
t_nc := (attribute incont) & any_t;
```

À partir de là, nous pouvons obtenir l'ensemble des configurations instables. Sachant que les configurations sont instables si elles sont sources d'au moins une transition étiquetée par un événement contrôlable, nous les obtenons à travers la commande Acheck suivante :

```
Q_ns := src(t_c) & any_s;
```

Nous déduisons les configurations stables par complémentarité avec l'ensemble des configurations :

```
Q_s := any_s - Q_ns;
```

Nous avons donc à notre disposition les notions de successeurs, d'événements contrôlables et de configurations stables. Nous pouvons construire à partir d'elles toutes les définitions et métriques de performances de diagnostic, sauf la perceptibilité.

Nous proposons en section 6.3.2 un exemple de système auquel nous appliquons d'une part les différentes commandes Acheck que nous venons de définir, et d'autre part des commandes Mec qui permettent de calculer ensuite les successeurs stables (l'utilisation de commandes Mec est nécessaire car la fonction $succ_s$ que nous avons définie en section 3.1 est basée sur un point fixe).

6.2.3 Perceptibilité

La perceptibilité (cf déf. 21 page 92) est une définition qui se distingue des autres. En effet, pour savoir si un événement répond à la définition, nous avons besoin de deux choses :

- Savoir dans quelles configurations il est déclenchable.
- Pouvoir comparer l'état précédant son occurrence et ceux la suivant.

C'est une information qui n'est pas forcément présente dans les effets et qui nécessite donc un calcul particulier.

Un paramètre de la définition de perceptibilité, comme pour la plupart des définitions, est un ensemble de configurations. Pour chaque configuration de cet ensemble il faut donc faire deux calculs :

- Regarder quels événements sont déclenchables dans cette configuration.
- Comparer la partie observable de la configuration aux effets des événements déclenchables.

Pour savoir quels sont les événements déclenchables dans une configuration c, il suffit de regarder l'étiquetage des transitions sortantes. On le fait en appliquant la commande ACHECK `rsrc`.

Par exemple, pour notre interrupteur, nous proposons les commandes ACHECK présentes en figure 6.28.

```
with Switch do
  // Considération d'une configuration.
  c := [on = true & i = 1 & o = 1];
  // Calcul des transitions sortantes (hors epsilon).
  t := (rsrc(c) - epsilon) & any_t;
  // Affichage des transitions sortantes.
  display(t);
done
```

FIGURE 6.28 – Perceptibilité pour le nœud `Switch`

Le résultat affiché lors de l'exécution des commandes (dans le fichier `Switch.ack`) est le suivant :

```
$ arc Switch.alt Switch.ack
...
Loading file 'Switch.alt'.
Loading file 'Switch.ack'.
set 't' contains
e = push and {
o = 1, o' = 0, on = true, on' = false, i = 1, i' in [0, 1]}
arc>
```

Nous remarquons donc que l'événement `push` est déclenchable dans $c = (on = true, i = 1, o = 1)$ car il nous donne comme résultat au moins une transition étiquetée par `push` (en l'occurrence, deux).

Ces calculs nous permettent de trouver l'ensemble des événements qui sont déclenchables depuis une configuration. Nous pouvons aussi obtenir directement depuis ARC la réponse à la question "L'événement e est-il déclenchable dans la configuration c ?". Pour cela, il suffit de croiser le résultat précédent avec les transitions de l'événement, puis de tester si l'ensemble est vide.

En reprenant l'exemple précédent, avec $e =$ `push` :

```
t_push := t & (label push);
test(t_push,0);
```

La réponse que l'on obtient est :

```
TEST(t,0) [FAILED] actual size = 2
```

Si le test est ***FAILED***, c'est que l'ensemble de transitions n'est pas vide, et donc que l'événement `push` est bien déclenchable dans c.

Finalement, nous pouvons voir la notion de déclenchables sous un troisième angle avec ARC : nous pouvons trouver les configurations pour lesquelles un événement est déclenchable dans un ensemble de configurations C'. Cet ensemble de configurations était noté $C'^{\xrightarrow{e}}$ (cf. notation précédant la définition 21 92).

Le calcul de cet ensemble de configurations à partir d'ARC et pour un événement e et un ensemble de configurations C' se fait avec les commandes suivantes :

```
Cp_e_decl := Cp & src(label e & any_t);
```

L'étape suivante consiste à obtenir la partie observable de chacune des configurations, cela est trivial et se résume à une projection sur les variables observables de la configuration.

On peut ensuite comparer la partie observable de la configuration aux effets des événements dans cette configuration pour obtenir la perceptibilité des événements.

6.3 Expérimentations

6.3.1 Expérimentations sur un modèle simple

Dans cette section nous modélisons et analysons le système sous diagnostic exemple de la partie II présenté dans la section 3.7.

6.3.1.1 Modélisation du système

Commençons par modéliser le système S_{ex} défini en 3.7.1. Nous rappelons l'automate décrivant le comportement du système en figure 6.29.

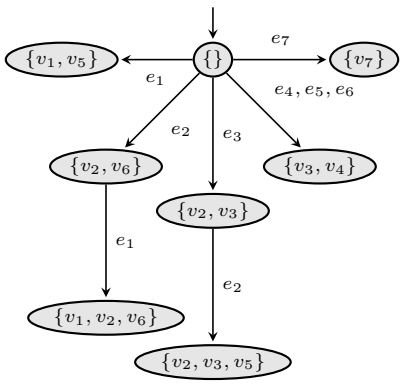


FIGURE 6.29 – S_{ex} avec valuations (rappel de la figure 3.6)

Le système peut être modélisé en ALTARICA tel que présenté en figure 6.30.

Comme nous l'avons expliqué lors de sa définition, le système que nous considérons n'a que des états stables, i.e. que des événements non contrôlables.

Lorsque nous générons le graphe de comportement de notre modèle avec ARC via la commande `dot`, nous obtenons le résultat présenté en figure 6.34.

Ce graphe de comportement est le même que celui de la figure précédente 6.29 à deux différences d'affichage près :

```
node Systeme
  state
    v1,v2,v3,v4,v5,v6,v7 : bool;
  init
    v1 := false;
    v2 := false;
    v3 := false;
    v4 := false;
    v5 := false;
    v6 := false;
    v7 := false;
  event
    e1,e2,e3,e4,e5,e6,e7 : incont;
  trans
    not (v1|v2|v3|v4|v5|v6|v7) |- e1 -> v1 := true, v5 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e2 -> v2 := true, v6 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e3 -> v2 := true, v3 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e4 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e5 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e6 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e7 -> v7 := true;
    not (v1|v3|v4|v5|v7)&v2&v6 |- e1 -> v1 := true;
    not (v1|v4|v5|v6|v7)&v2&v3 |- e2 -> v5 := true;
edon
```

FIGURE 6.30 – Exemple : Modèle simple

- Certaines variables (valuées) n'apparaissent pas, pour la même raison que pour la figure 6.19.
- Les configurations ne sont pas présentées comme simplement des ensembles de variables mais comme des ensembles de valuations *variable* = *valeur*. Nous avions simplifié la forme de nos configurations du fait que le domaine des variables est booléen.

6.3.1.2 Modélisation du système sous diagnostic

Le deuxième étape est de modéliser le système sous diagnostic. Nous devons donc modéliser l'observateur et la relation d'observation.

Nous pouvons tout d'abord rappeler l'automate décrivant le comportement du système sous diagnostic en figure 6.31.

Pour modéliser l'observateur, nous définissons un nouveau nœud qui va intégrer l'ensemble des variables d'observation. La figure 6.32 montre le code ALTARICA décrivant le nœud *Observateur*.
L'observateur est défini par l'ensemble de variables $V_{O_{ex}} = \{o_1, o_2, o_3, o_4\}$.

Nous rappelons la relation d'observation $\mathcal{R}_{O_{ex}}$ qui est la suivante :

$$\mathcal{R}_{O_{ex}} = \begin{cases} o_1 = v_1 \\ o_2 = v_2 \\ o_3 = v_3 \\ o_4 = v_4 \end{cases}$$

Pour la modéliser, nous créons un nouveau nœud `SUD` qui a comme sous-nœuds le système et l'observateur. Dans ce nœud nous lions à travers des assertions les variables d'observations aux variables d'état du système. Le modèle ALTARICA résultant est présenté en figure 6.33.

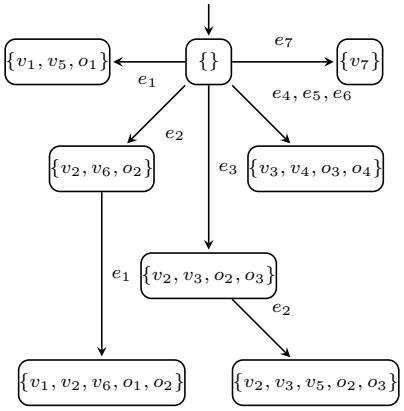


FIGURE 6.31 – Système sous diagnostic A_{ex} (rappel de la figure 3.7)

```
node Observateur
  flow
    o1,o2,o3,o4 : bool;
edon
```

FIGURE 6.32 – Exemple : Modèle simple

```
node SUD
  sub
    S : Systeme;
    O : Observateur;
  assert
    // Relation d'observation
    O.o1 = S.v1
    O.o2 = S.v2
    O.o3 = S.v3
    O.o4 = S.v4
edon
```

FIGURE 6.33 – Exemple : Modèle simple

Notons tout de même que pour que le modèle SUD soit correct, il faut que le nœud ait la visibilité suffisante sur les variables d'observation et les variables d'état du système pour leur appliquer des assertions.

Par défaut, et dans le cas des nœuds `Systeme` et `Observateur` que nous avons présentés, la visibilité des variables d'état est privée. Cela signifie qu'aucun nœud autre que le nœud lui-même ne peut accéder à ses variables d'état. Il est donc nécessaire de rendre ces variables visibles par le nœud parent `SUD`. Pour cela, il suffit de rajouter l'attribut de visibilité `parent` aux variables considérées.

Cela se concrétise avec la modification de la ligne de déclaration des variables d'état de la manière suivante :

```
node Systeme
  state
    v1,v2,v3,v4,v5,v6,v7 : bool : parent;
  ...
edon

node Observateur
  flow
    o1,o2,o3,o4 : bool;
  ...
edon
```

Nous pouvons générer le graphe de comportement du système sous diagnostic avec ARC. Le résultat est présenté en figure 6.35.

Si le graphe de comportement est le même que l'automate de comportement rappelé en figure 6.31, quelques différences d'affichage sont tout de même à noter. Les variables et leur valeur n'apparaissent pas toutes dans les nœuds.

Nous retrouvons les mêmes différences d'affichage que celles décrites pour le graphe 6.34, avec en plus une autre différence : sont uniquement affichées les variables qui ne sont pas redondantes. Dans notre cas, les variables v_1, v_2, v_3 et v_4 n'apparaissent pas parce leur valeur est exactement reprise par les variables d'observation o_1, o_2, o_3 et o_4.

6.3.1.3 Analyse du modèle

Nous avons modélisé l'ensemble de notre système sous diagnostic sous AltaRica. Nous pouvons désormais nous intéresser à l'analyse des performances de diagnostic du modèle.

Dans un soucis de simplification, l'exemple d'illustration que nous avons mis en place pour l'ensemble de la partie II ne contient que des événements incontrôlables, et donc que des configurations stables. La notion de successeurs et de successeurs stables sont donc équivalent dans ce cas.

Pour effectuer les analyses nous allons faire appel à ARC à travers les commandes que nous avons décrites notamment dans la section 6.2.

Dans la pratique, pour gérer, manipuler, mémoriser les résultats obtenus par les commandes ARC, nous utilisons une implémentation Java. Nous ne décrirons pas ici le détail des classes et opérations à effectuer avec Java. Typiquement, les relations de bases qui doivent être extraites du modèle AltaRica telles que les successeurs, les notions de contrôlables et stables, ou encore la notion de perceptible, seront obtenues par ARC. Les notions en dérivant par projection, intersection, union et tout autre opération de ce type là seront effectués avec Java, à partir des relations de bases calculées avec ARC.

Nous ne décrirons pas ces opérations effectuées sous Java car cela consiste à implémenter les définitions et métriques de performances de diagnostic décrites dans la partie II. Ces définitions correspondant principalement à des tests, des fonctions ou des calculs de ratio, il n'est pas difficile de les mettre en œuvre.

Reprenons les commandes Acheck définies précédemment (section 6.2) et appliquons les à notre système sous diagnostic exemple.

La première étape est de définir un ensemble de configurations pour lesquelles nous voulons faire les calculs. Considérons pour cet exemple l'ensemble des configurations $C' = C^1_{ex}$ qui représente l'ensemble des configurations accessibles en une occurrence d'événement non contrôlable maximum. Étant donné que tous les événements sont non contrôlables, nous pouvons le calculer simplement à travers la commande suivante :

```
Cp := initial | (tgt(rsrc(initial) & any_t) & any_s);
```

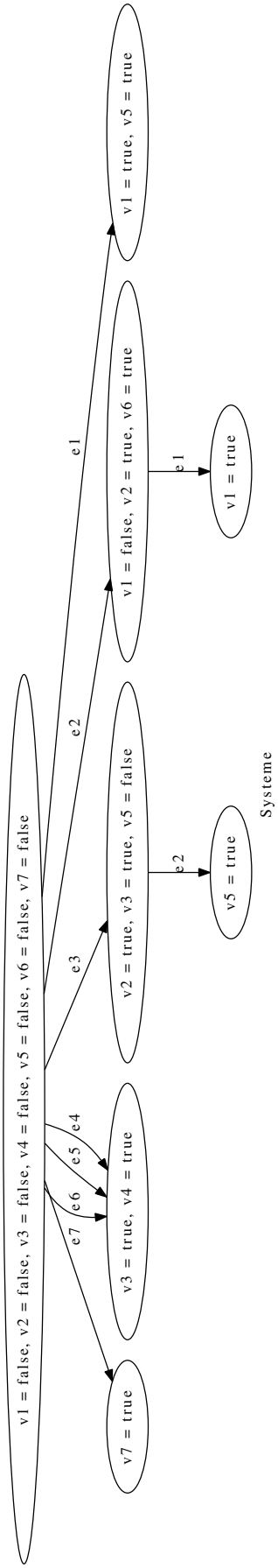


FIGURE 6.34 – Exemple simple : Graphe de comportement du nœud `Systeme` (sans les ϵ-transitions)

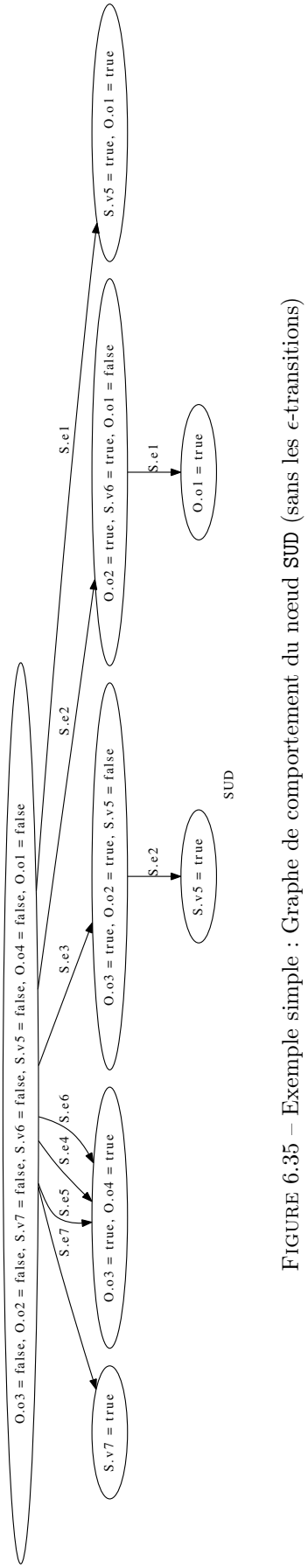


FIGURE 6.35 – Exemple simple : Graphe de comportement du nœud `SUD` (sans les ϵ-transitions)

Nous faisons ici l'union des configurations initiales avec les configurations accessibles depuis l'état initial avec une occurrence d'événement. L'ensemble `Cp` résultat calculé par ARC :

```
set 'Cp' contains
O.o3 = false, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = false,
O.o4 = false, O.o1 = false
O.o3 = false, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = true, O.o4
= false, O.o1 = true
O.o3 = false, O.o2 = false, S.v7 = true, S.v6 = false, S.v5 = false, O.o4
= false, O.o1 = false
O.o3 = false, O.o2 = true, S.v7 = false, S.v6 = true, S.v5 = false, O.o4
= false, O.o1 = false
O.o3 = true, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = false, O.o4
= true, O.o1 = false
O.o3 = true, O.o2 = true, S.v7 = false, S.v6 = false, S.v5 = false, O.o4
= false, O.o1 = false
```

Une nouvelle fois, toutes les valeurs n'apparaissent pas lorsque nous utilisons le calcul basé sur la représentation symbolique. L'absence de certaines variables est dû ici uniquement au fait que les informations redondantes ne sont pas affichées (v_1, v_2, v_3, v_4 valent respectivement toujours exactement o_1, o_2, o_3, o_4). L'ensemble des configurations résultat `Cp` correspond à l'ensemble $C^1_{ex} = \{c_1, c_2, c_3, c_4, c_5, c_6\}$.

L'idée étant derrière de calculer les effets de chacun des événements pour pouvoir appliquer les définitions et métriques, nous allons effectuer les calculs de successeurs pour chacun des événements individuellement. Dans le programme JAVA qui pilote les commandes, un calcul est fait pour chaque événement. Nous proposons ici l'illustration pour un événement. Considérons l'événement e_1. Nous calculons l'ensemble des transitions étiquetées par e_1 avec la commande ACHECK `t_Ep := (label S.e1) & any_t;` définie dans la partie précédente et obtenons comme résultat :

```
set 't_Ep' contains
e = S.e1 and {
O.o3 = false, O.o3' = false, O.o2 = false, O.o2' = false, S.v7 = false,
S.v7' = false, S.v6 = false, S.v6' = false, S.v5 = false, S.v5' = true,
O.o4 = false, O.o4' = false, O.o1 = false, O.o1' = true
O.o3 = false, O.o3' = false, O.o2 = true, O.o2' = true, S.v7 = false,
S.v7' = false, S.v6 = true, S.v6' = true, S.v5 = false, S.v5' = false,
O.o4 = false, O.o4' = false, O.o1 = false, O.o1' = true}
```

Nous obtenons ainsi dans l'ordre les deux transitions (c_1, e_1, c_2) et (c_3, e_1, c_7) de T_{ex}.

Pour ne conserver que les transitions dont la configuration source appartient à l'ensemble C' qui nous intéresse, nous appliquons la commande `t_Ep_Cp := t_Ep & rsrc(Cp) & any_t;` et obtenons :

```
set 't_Ep_Cp' contains
e = S.e1 and {
O.o3 = false, O.o3' = false, O.o2 = false, O.o2' = false, S.v7 = false,
S.v7' = false, S.v6 = false, S.v6' = false, S.v5 = false, S.v5' = true,
O.o4 = false, O.o4' = false, O.o1 = false, O.o1' = true
O.o3 = false, O.o3' = false, O.o2 = true, O.o2' = true, S.v7 = false,
S.v7' = false, S.v6 = true, S.v6' = true, S.v5 = false, S.v5' = false,
O.o4 = false, O.o4' = false, O.o1 = false, O.o1' = true}
```

L'ensemble est le même dans notre cas.

L'opération suivante donne l'ensemble des successeurs de `Ep` dans `Cp` à travers la commande `succ_Ep_Cp := tgt(t_Ep_Cp) & any_c;`. Le résultat que nous obtenons est :

```
set 'succ_Ep_Cp' contains
O.o3 = false, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = true, O.o4
= false, O.o1 = true
O.o3 = false, O.o2 = true, S.v7 = false, S.v6 = true, S.v5 = false, O.o4
= false, O.o1 = true
```

Les deux configurations résultats correspondent dans l'ordre aux configurations c_2 et c_7.

Comme nous sommes dans le cas d'un système qui ne contient que des événements non contrôlables et donc que des configurations stables, les successeurs obtenus correspondent aux successeurs stables. Nous pouvons très bien appliquer les calculs génériques relatifs à la stabilité des configurations, mais cela n'est pas nécessaire dans notre cas, donc nous ne l'expliciterons pas ici.

Nous pouvons désormais obtenir les effets de l'événement e_1, qui correspond à la partie observable des configurations que nous venons de calculer. La projection sur l'ensemble des variables observables est typiquement le type d'opération que nous effectuons en dehors d'ARC.

Ici, si nous ne conservons que les variables d'observation dans les configurations, nous obtenons pour l'événement e_1 les configurations projetées :

- `O.o3 = false, O.o2 = false, O.o4 = false, O.o1 = true`
- `O.o3 = false, O.o2 = true, O.o4 = false, O.o1 = true`

Cela correspond aux effets de e_1 dans C_{ex} que nous avons représentés dans l'exemple de la définition 15 : $\{\{o_1\},\{o_1,o_2\}\}$. Notons que les effets d'un événement e dans C sont exactement les mêmes que les effets de e dans C^{prof-1} où $prof$ représente la profondeur du système, car dans notre cas, avoir sa source dans C^{prof-1} revient à obtenir son successeur dans $C^{prof} = C$.

C'est pourquoi les effets de e_1 dans C^1_{ex} sont toujours les mêmes que ceux dans C_{ex}.

À partir de là nous pouvons obtenir l'ensemble des définitions et métriques de performances de diagnostic en implémentant les définitions formelles données en partie II, hors mis la notion de perceptibilité.

Pour décider de la perceptibilité d'un événement dans un ensemble de configurations, nous avons vu dans la partie précédente qu'il fallait effectuer un calcul supplémentaire.

Calculons la perceptibilité de l'événement e_1 dans $C' = C^1_{ex}$. Pour cela, commençons par calculer l'ensemble des configurations de C' dans lesquels e_1 est déclenchable avec la commande ACHECK `Cp_e1_decl := Cp & src(label S.e1 & any_t);`. L'ensemble des configurations obtenu est :

```
set 'Cp_e1_decl' contains
O.o3 = false, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = false,
O.o4 = false, O.o1 = false
O.o3 = false, O.o2 = true, S.v7 = false, S.v6 = true, S.v5 = false, O.o4
= false, O.o1 = false
```

Ces configurations résultats correspondent aux configurations c_1 et c_3.

Pour chacune de ces configurations où e_1 est déclenchable, il faut maintenant pouvoir comparer la partie observable de la configuration aux effets de e_1 dans cette configuration. On peut calculer les effets de e_1 dans c_1 et c_3 avec ARC avec la même méthode que celle utilisée précédemment pour calculer les effets dans un ensemble de configurations. Le code ACHECK pour cela est le suivant :

```
// Encodage des configurations
c1 := [O.o3 = false & O.o2 = false & S.v7 = false & S.v6 = false & S.v5 =
false & O.o4 = false & O.o1 = false];
c3 := [O.o3 = false & O.o2 = true & S.v7 = false & S.v6 = true & S.v5 =
false & O.o4= false & O.o1 = false];
// Calcul des effets
t_e1_c1 := t_e1 & rsrc(c1) & any_t;
t_e1_c3 := t_e1 & rsrc(c3) & any_t;
succ_e1_c1 := tgt(t_e1_c1) & any_c;
succ_e1_c3 := tgt(t_e1_c3) & any_c;
```

Les effets obtenus pour e_1 dans c_1 et c_3 sont les suivants :

```
set 'succ_e1_c1' contains
O.o3 = false, O.o2 = false, S.v7 = false, S.v6 = false, S.v5 = true, O.o4
= false, O.o1 = true
set 'succ_e1_c3' contains
O.o3 = false, O.o2 = true, S.v7 = false, S.v6 = true, S.v5 = false, O.o4
= false, O.o1 = true
```

Nous pouvons déduire, étant donné que la configuration c_1, resp. c_3, réduite à sa partie observable n'appartient pas aux effets de e_1 dans c_1, resp. c_3, alors e_1 est fortement perceptible dans C'.

Nous savons obtenir les effets des événements à partir du modèle AltaRica à l'aide d'ARC. Les autres définitions et métriques sont calculées à partir de cela via des opérations, fonctions, définitions et ratios implémentés directement en Java.

Néanmoins, il est tout à fait possible d'obtenir des résultats concernant la définition des successeurs stables basée sur un calcul de point fixe en utilisant ARC et Mec 5. Les expérimentations de la section suivante (section 6.3.2) illustrent ce point.

6.3.2 Expérimentations sur un système avec contrôlabilité

Nous présentons l'exemple d'un système modélisé en AltaRica sur lequel nous calculons l'ensemble des notions de successeurs, d'accessibles et de configurations stables dont les commandes ont été définies dans la section 6.2.

Dans cette section, nous considérons un modèle qui intègre des événements contrôlables. En outre, nous montrons dans cet exemple comment il est possible de calculer les successeurs stables à partir d'ARC, ce qui n'a pas été fait jusque là. Certaines notions d'ARC et Mec 5 utilisées dans cette section n'ont pas été introduites auparavant. Nous nous référerons aux documents références parmi lesquels [Poi00, Vin03a] pour en avoir une description précise.

Considérons le système représenté par le modèle de la figure 6.36.

```
Domain Comp_Etats = {ok,stop,ko};

node Moteur
       state
              etat : Comp_Etats : parent;
       init
              etat := stop;
       event
              defKO : incont, bornant;
              demarre : cont;
              repare : incont;
       trans
              (etat = stop) |- demarre -> etat := ok;
              (etat = ok) |- defKO -> etat := ko;
              (etat = ko) |- repare -> etat := stop;
edon
```

FIGURE 6.36 – Modèle AltaRica d'un moteur

Le modèle est composé d'un moteur qui a trois états : `ok`, `stop` et `ko`. Le moteur est initialement arrêté (état `stop`). Il peut démarrer quand il est à l'arrêt (et passer dans l'état `ok`), défaillir lorsqu'il est en fonctionnement (et passer dans l'état `ko`), ou être réparé lorsqu'il est cassé (et ainsi repasser dans l'état `stop`). Nous retrouvons dans cet exemple un événement de type contrôlable (`demarre`) qui correspond à une action interne au système et deux autres de type incontrôlable,

qui correspondent aux actions de défaillance et de réparation, qui sont externes au système. Nous obtenons ainsi des configurations stables et d'autres non stables.

Le graphe de comportement du modèle obtenu par la commande `dot` est donné en figure 6.37.

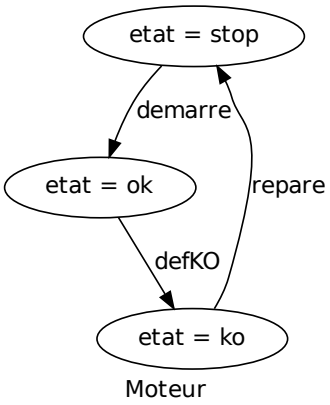


FIGURE 6.37 – Graphe de comportement `Moteur`

Considérons maintenant un système comportant deux moteurs mettant en place un mécanisme de redondance froide. Cela signifie qu'un seul moteur est activé à la fois, et que le deuxième prend le relais lorsque le premier tombe en panne.

Nous proposons le modèle ALTARICA de ce système en figure 6.38.

```
node Systeme
       sub
              M : Moteur[2]; //Moteur 0 commence
       event
              demarreM0, demarreM1, repareM0, repareM1;
              // Priorites
              {demarreM0, demarreM1} > {repareM0, repareM1},
              demarreM0 > demarreM1,
              repareM0 > repareM1;
       trans
              (M[0].etat != ok) |- demarreM1 -> ;
              (M[1].etat != ok) |- demarreM0 -> ;
              true |- repareM0 -> ;
              true |- repareM1 -> ;
       sync
              <demarreM0, M[0].demarre>;
              <demarreM1, M[1].demarre>;
              <repareM0, M[0].repare>;
              <repareM1, M[1].repare>;
edon
```

FIGURE 6.38 – Modèle ALTARICA d'un système de deux moteurs

Le nœud système comprend deux moteurs. Nous avons défini deux événements de démarrage et deux événements de réparation qui correspondent aux événements de démarrage et de réparation de chacun des moteurs. Pour caractériser cette correspondance, chacun des événements du nœud parent est synchronisé avec l'événement correspondant du moteur (champ `sync`). Dans le champ `event`, nous intégrons des priorités qui permettent notamment de décrire la redondance froide. Ces priorités sont au nombre de trois et décrivent respectivement que :

- les démarrages sont prioritaires sur les réparations.
- le premier moteur est prioritaire pour le démarrage.
- le premier moteur est prioritaire pour la réparation.

Enfin, dans le champ `trans` nous nous assurons qu'un moteur ne démarre si l'autre l'est déjà.

Le graphe de comportement du modèle obtenu par la commande `dot` pour le Système est donné en figure 6.39.

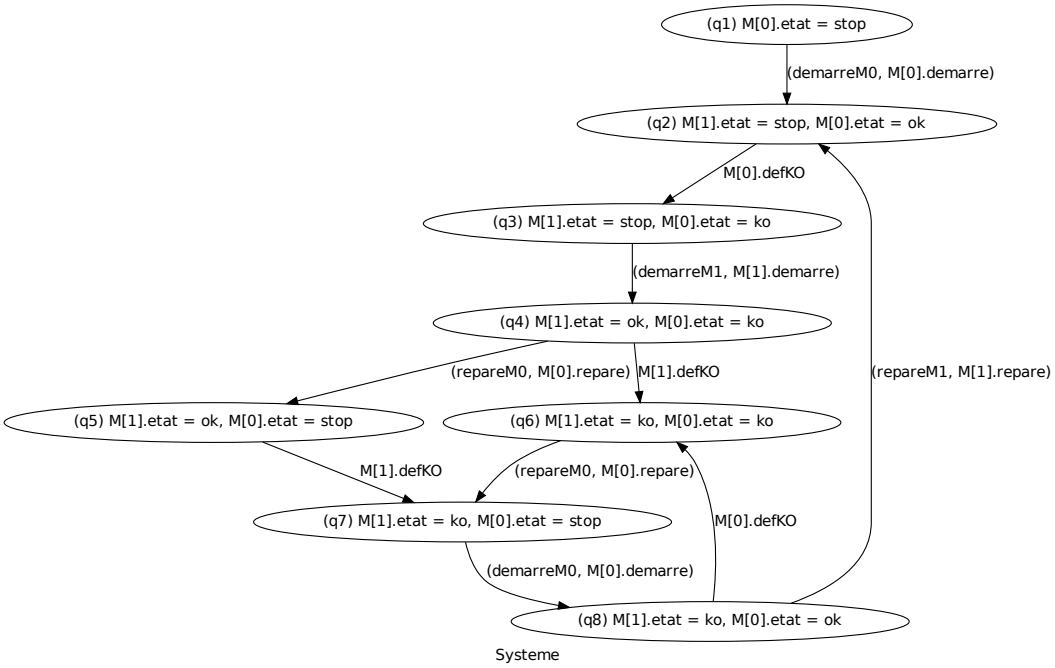


FIGURE 6.39 – Graphe de comportement `Systeme`

Maintenant que le modèle de redondance froide est introduit, nous proposons en figure 6.40 le fichier de spécifications combinant des commandes ACHECK et MEC pour le calcul des différents ensembles de configurations ou de transitions qui nous intéressent.

La première partie encadrée par `with` et `done` contient des commandes ACHECK. La suite des commandes sont des commandes MEC.

Nous donnons les spécifications ACHECK et leurs résultats en nous référant aux commandes présentées en section 6.2. Comme ensembles de départ nous considérons $Q' = \{q_2\}$ et $E' = \{$`M[0].defKO`$\}$. `Qp` contient donc la configuration q_2 et `t_Ep` contient l'ensemble des transitions étiquetées par l'événement `M[0].defKO`.

Nous déroulons ensuite les calculs des ensembles `t_Ep_Qp`, `succ_Ep_Qp`, `reach_Ep_Qp`, `t_c`, `t_nc`, `Q_ns`, et `Q_s` tels que nous les avons définis en 6.2.

L'ensemble des transitions prenant leur source dans Q' et étiquetées par un événement de E' est représenté par `t_Ep_Qp` et donne comme résultat :

```
set 't_Ep_Qp' contains
e = M[0].defKO and {
M[1].etat = stop, M[1].etat' = stop, M[0].etat = ok, M[0].etat' = ko}
```

La seule transition résultat est celle étiquetée par l'événement `defKO` qui part de q_2.

L'ensemble des successeurs de Q' par E', représenté par `succ_Ep_Qp`, donne comme résultat :

```
with Systeme do
  Qp := [M[0].etat = ok & M[1].etat = stop];
  t_Ep := (label M[0].defKO) & any_t;
  t_Ep_Qp := t_Ep & rsrc(Qp) & any_t;
  succ_Ep_Qp := tgt(t_Ep_Qp) & any_c;
  reach_Ep_Qp := reach(Qp, t_Ep) & any_c;

  t_c := (attribute cont) & any_t;
  t_nc := (attribute incont) & any_t;
  t_b := (attribute bornant) & any_t;

  Q_ns := src(t_c) & any_s;
  Q_s := any_s - Q_ns;
done

//q' est un succ de q par e
succ(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) := Systeme!t(q,e,q') & not
     Systeme!epsilon(e) & Systeme!reach(q);

//q' est un succ stable de q par e
succ_s(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) += succ(q,q',e) & Systeme!Q_s(q') | <q'' :
     Systeme!c>(succ(q,q'',e) & Systeme!Q_ns(q'') & <e' : Systeme!ev>(succ_s(q'',q',e')));

succ'_Ep_Qp(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) := Systeme!t_Ep_Qp(q,e,q') & not
     Systeme!epsilon(e);

succ_s_Ep_Qp(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) += succ'_Ep_Qp(q,q',e) &
     Systeme!Q_s(q') | <q'' : Systeme!c>(succ'_Ep_Qp(q,q'',e) & Systeme!Q_ns(q'') & <e' :
     Systeme!ev>(succ_s(q'',q',e')));
```

FIGURE 6.40 – Spécifications pour l'exemple `Systeme`

```
set 'succ_Ep_Qp' contains
M[1].etat = stop, M[0].etat = ko
```

La seule configuration qui répond à cette propriété est q_3.

L'ensemble des accessibles à partir de Q' et en utilisant des transitions étiquetées par des événements de E' est représenté par `reach_Ep_Qp` et donne comme résultat :

```
set 'reach_Ep_Qp' contains
M[1].etat = stop, M[0].etat = ok
M[1].etat = stop, M[0].etat = ko
```

Les configurations qui répondent à cette propriété sont $\{q_2, q_3\}$.

L'ensemble des transitions étiquetées par des événements contrôlables sont représentées par `t_c` qui vaut :

```
set 't_c' contains
e = (demarreM1, M[1].demarre) and {
M[1].etat = stop, M[1].etat' = ok, M[0].etat = ko, M[0].etat' = ko}
e = (demarreM0, M[0].demarre) and {
M[1].etat = stop, M[1].etat' = stop, M[0].etat = stop, M[0].etat' = ok
M[1].etat = ko, M[1].etat' = ko, M[0].etat = stop, M[0].etat' = ok}
```

Les événements `(demarreM0, M[0].demarre)` et `(demarreM1, M[1].demarre)` sont contrôlables et les transitions résultats sont celles qui sont étiquetées par ces événements. Elles sont au nombre de trois. Les transitions non contrôlables sont donc les autres :

```
set 't_nc' contains
e = M[0].defKO and {
M[1].etat = stop, M[1].etat' = stop, M[0].etat = ok, M[0].etat' = ko
M[1].etat = ko, M[1].etat' = ko, M[0].etat = ok, M[0].etat' = ko}
e = M[1].defKO and {
M[1].etat = ok, M[1].etat' = ko, M[0].etat = stop, M[0].etat' = stop
M[1].etat = ok, M[1].etat' = ko, M[0].etat = ko, M[0].etat' = ko}
e = (repareM1, M[1].repare) and {
M[1].etat = ko, M[1].etat' = stop, M[0].etat = ok, M[0].etat' = ok}
```

```
e = (repareM0, M[0].repare) and {
M[1].etat = ok, M[1].etat' = ok, M[0].etat = ko, M[0].etat' = stop
M[1].etat = ko, M[1].etat' = ko, M[0].etat = ko, M[0].etat' = stop}
```

Les configurations sources de transitions étiquetées par des événements contrôlables sont non stables :

```
set 'Q_ns' contains
M[1].etat = stop, M[0].etat in {stop, ko}
M[1].etat = ko, M[0].etat = stop
```

`Q_ns` contient les configurations q_1, q_3 et q_7. `Q_s` contient donc les autres configurations :

```
set 'Q_s' contains
M[1].etat = ok, M[0].etat in {stop, ko}
M[1].etat = stop, M[0].etat = ok
M[1].etat = ko, M[0].etat = ok
M[1].etat = ko, M[0].etat = ko
```

En plus de ces spécifications, nous avons ajouté celle qui permet de calculer l'ensemble des transitions étiquetées par des événements bornants :

```
t_b := (attribute bornant) & any_t;
```

Les résultats pour cette spécification, qui correspondent aux transitions étiquetées par les événements `M[0].defKO` et `M[1].defKO`, sont les suivants :

```
set 't_b' contains
e = M[0].defKO and {
M[1].etat = stop, M[1].etat' = stop, M[0].etat = ok, M[0].etat' = ko
M[1].etat = ko, M[1].etat' = ko, M[0].etat = ok, M[0].etat' = ko}
e = M[1].defKO and {
M[1].etat = ok, M[1].etat' = ko, M[0].etat = stop, M[0].etat' = stop
M[1].etat = ok, M[1].etat' = ko, M[0].etat = ko, M[0].etat' = ko}
```

Les deux premières commandes Mec qui sont à la suite des commandes Acheck (en dehors du `with ... do ... done`), sont utilisées pour calculer l'ensemble des successeurs stables du système. Les deux suivantes permettent de paramétrer ce calcul par les ensembles Q' et E'. Ces commandes sont définies comme des relations.

Pour calculer la relation de successeurs stables, on transcrit la définition donnée en section 3.1 en Mec, et cela car il est possible d'y effectuer des calculs de points fixes alors que ce n'est pas le cas avec des commandes Acheck.

On commence par définir la relation de successeurs dont on a besoin dans la définition de $succ_s$:

```
succ(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) := Systeme!t(q,e,q') & not
                    Systeme!epsilon(e) & Systeme!reach(q);
```

La relation de successeur telle qu'elle est exprimée est définie entre deux configurations `q` et `q'` et avec un événement `e`. Nous exprimons dans la suite que `q'` est un successeur de `q` par `e` s'il existe une transition (q, e, q') dans le système. Nous nous assurons finalement que `q` est accessible.

Pour que le résultat des calculs pour cette relation soit présentable, nous avons raccourci la partie concernant l'événement. Au lieu d'avoir une valeur pour l'événement du systeme, et une pour l'événement de chacun des moteurs, nous donnons un seul événement qui suffit à identifier la transition :

```
succ contains :
q.M[0].etat = ko, q.M[1].etat = ok, q'.M[0].etat = stop, q'.M[1].etat = ok, e = repareM0
q.M[0].etat = stop, q.M[1].etat = ok, q'.M[0].etat = stop, q'.M[1].etat = ko, e = M[1].defKO
q.M[0].etat = ko, q.M[1].etat = ok, q'.M[0].etat = ko, q'.M[1].etat = ko, e = M[1].defKO
q.M[0].etat = ok, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = stop, e = M[0].defKO
q.M[0].etat = stop, q.M[1].etat = stop, q'.M[0].etat = ok, q'.M[1].etat = stop, e = demarreM0
q.M[0].etat = ko, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = ok, e = demarreM1
q.M[0].etat = ok, q.M[1].etat = ko, q'.M[0].etat = ko, q'.M[1].etat = ko, e = M[0].defKO
q.M[0].etat = stop, q.M[1].etat = ko, q'.M[0].etat = ok, q'.M[1].etat = ko, e = demarreM0
q.M[0].etat = ko, q.M[1].etat = ko, q'.M[0].etat = stop, q'.M[1].etat = ko, e = repareM0
q.M[0].etat = ok, q.M[1].etat = ko, q'.M[0].etat = ok, q'.M[1].etat = stop, e = repareM1
```

Les successeurs résultats sont, dans l'ordre, (q_4, q_5,(repareM0, M[0].repare)), (q_5, q_7, M[1].defKO), (q_4, q_6,M[1].defKO), (q_2, q_3,M[0].defKO), (q_1, q_2,(demarreM0, M[0].demarre)), (q_3 , q_4,(demarreM1, M[1].demarre)), (q_8, q_6,M[0].defKO), (q_7, q_8,(demarreM0, M[0].demarre)), (q_6, q_7,(repareM0, M[0].repare)), (q_8, q_2,(repareM1, M[1].repare)).

À partir de la relation successeurs, à l'aide de la définition de $succ_s$ et en utilisant un calcul de point fixe, nous pouvons calculer la relation de successeur stable :

```
succ_s(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) += succ(q,q',e) &
Systeme!Q_s(q') | <q'' : Systeme!c>(succ(q,q'',e) & Systeme!Q_ns(q'') & <e' :
                    Systeme!ev>(succ_s(q'',q',e')));
```

La relation de successeur stable associe deux configurations `q`, `q'` et un événement `e`. Ce qui est exprimé dans cette relation, c'est que `q'` est un successeur stable de `q` par `e` si `q'` est stable et successeur de `q` par `e`, ou s'il existe une configuration non stable `q''` successeur de `q` par `e` telle que `q'` est un successeur stable de `q''`.

Le résultat que donne Mec pour cette relation (avec réécriture) est :

```
succ_s contains :
q.M[0].etat = ko, q.M[1].etat = ok, q'.M[0].etat = stop, q'.M[1].etat = ok, e = repareM0
q.M[0].etat = stop, q.M[1].etat = ok, q'.M[0].etat = ok, q'.M[1].etat = ko, e = M[1].defKO
q.M[0].etat = ko, q.M[1].etat = ok, q'.M[0].etat = ko, q'.M[1].etat = ko, e = M[1].defKO
q.M[0].etat = ok, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = ok, e = M[0].defKO
q.M[0].etat = stop, q.M[1].etat = stop, q'.M[0].etat = ok, q'.M[1].etat = stop, e = demarreM0
q.M[0].etat = ko, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = ok, e = demarreM1
q.M[0].etat = ok, q.M[1].etat = ko, q'.M[0].etat = ko, q'.M[1].etat = ko, e = M[0].defKO
q.M[0].etat = stop, q.M[1].etat = ko, q'.M[0].etat = ok, q'.M[1].etat = ko, e = demarreM0
q.M[0].etat = ko, q.M[1].etat = ko, q'.M[0].etat = ok, q'.M[1].etat = ko, e = repareM0
q.M[0].etat = ok, q.M[1].etat = ko, q'.M[0].etat = ok, q'.M[1].etat = stop, e = repareM1
```

Les successeurs stables, i.e. les couples (q, q', e) où q' est successeur stable de q par e, donnés en résultats sont, dans l'ordre, (q_4, q_5,(repareM0, M[0].repare)), (q_5, q_8,M[1].defKO), (q_4, q_6, M[1].defKO), (q_2, q_4,M[0].defKO), (q_1, q_2,(demarreM0, M[0].demarre)), (q_3, q_4,(demarreM1, M[1].demarre)), (q_8, q_6,M[0].defKO), (q_7, q_8,(demarreM0, M[0].demarre)), (q_6, q_8,(repareM0, M[0].repare)), (q_8, q_2,(repareM1, M[1].repare)).

Nous venons de définir des relations qui comprennent les successeurs et les successeurs stables d'un point de vue global. Nous pouvons calculer l'ensemble des successeurs stables des configurations de Q' pour un ensemble d'événements E' en définissant deux nouvelles relations `succ'_Ep_Qp` et `succ_s_Ep_Qp` :

```
succ'_Ep_Qp(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) :=
      Systeme!t_Ep_Qp(q,e,q') & not Systeme!epsilon(e);

succ_s_Ep_Qp(q : Systeme!c, q' : Systeme!c, e : Systeme!ev) +=
succ'_Ep_Qp(q,q',e) & Systeme!Q_s(q') | <q'' : Systeme!c>(succ'_Ep_Qp(q,q'',e) &
          Systeme!Q_ns(q'') & <e' : Systeme!ev>(succ_s(q'',q',e')));
```

À travers ces relations, nous nous assurons juste que `q` appartient à `Qp` et que la transition sortante de `q` appartient à `t_Ep`. Les résultats (avec réécriture) de ces relations pour notre système sont les suivants :

```
succ'_Ep_Qp contains :
q.M[0].etat = ok, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = stop, e = M[0].defKO
succ_s_Ep_Qp contains :
q.M[0].etat = ok, q.M[1].etat = stop, q'.M[0].etat = ko, q'.M[1].etat = ok, e = M[0].defKO
```

Le seul successeur stable de Q' pour E' est q_4 (solution = (q_2, q_4,M[0].defKO)). Nous pouvons noter que le résultat de la relation Mec `succ'_Ep_Qp` est le même que celui pour la relation correspondante Acheck `succ_Ep_Qp` définie plus haut dans cette section.

Chapitre 7

Intégration dans le contexte industriel

La thèse dont il est question dans ce rapport est une thèse CIFRE [1]. Cela implique que la thèse se déroule dans le cadre de l'entreprise. Il y a donc une forte dimension industrielle dans ce travail de thèse.

Ce chapitre est dédié à la partie industrielle des travaux. Nous allons présenter dans un premier temps le contexte industriel du travail de thèse. Nous expliquerons ensuite comment nous avons développé notre travail tout en prenant en compte les contraintes industrielles fortes, et présenterons plus en détails les méthodes utilisées pour appliquer notre travail à des systèmes avioniques. Finalement, nous nous intéresserons à la suite qui peut être donnée aux travaux dans le cadre de l'entreprise.

7.1 Contexte industriel des travaux

Les travaux de thèse se sont déroulés dans le cadre de travaux de recherche industrielle dans Thales, et plus particulièrement Thales Avionics.

Cette section expose le contexte précis des travaux en commençant par présenter l'équipe que j'ai rejointe puis en donnant uni idée de l'état d'avancement des travaux et des choix d'orientation de ces travaux à mon arrivée.

7.1.1 Présentation du groupe d'études

L'équipe que j'ai rejointe pour effectuer mon travail de thèse est un groupe d'études sur la maintenance et le diagnostic. Elle fait partie de la société Thales Avionicsdu groupe Thales, et appartient plus particulièrement à l'unité *Missions et Fonctions*.

Les travaux ont vocation à être appliqués à des systèmes dits *avioniques*. Les systèmes avioniques comprennent un ensemble d'équipements qui ont pour but d'assister le pilotage des aéronefs en fournissant un ensemble d'outils de repérage, de communication, de prédiction et de surveillance de l'aéronef et de son environnement. À Thales Avionics Toulouse en particulier, Thales développe notamment les outils de gestion du vol (FM [2]), de pilote automatique, de surveillance (TAWS [3], etc.) d'avertissement au pilote (FWS [4]) et de maintenance des systèmes

1. **C**onventions **I**ndustrielles de **F**ormation par la **RE**cherche.
2. ***F**light **M**anagement.*
3. ***T**errain **A**wareness and **W**arning **S**ystem.*
4. ***F**light **W**arning **S**ystem.*

avioniques (BITE [5], etc.).

Les travaux de l'équipe font partie des travaux concernant la maintenance des systèmes avioniques. Plus particulièrement le diagnostic des systèmes avioniques pour la maintenance.

Le travail de l'équipe étude consiste notamment à réfléchir à de nouvelles technologies et méthodes de diagnostic à appliquer aux systèmes avioniques dans le cadre de la maintenance de ces systèmes.

7.1.2 État d'avancement avant la thèse

Le contexte de travail du groupe d'études est celui que nous avons décrit dans la section 2.1.5. Le point de départ du travail est donc celui actuellement déployé dans les produits, c'est-à-dire, pour rappel, l'utilisation de tables de règles décrivant des relations de causes à effets entre les composants et les observations définies par l'ensemble des messages de monitoring reçus.

Parmi les premières préoccupations d'un travail d'étude dans le contexte industriel, on trouve la notion de faisabilité. Celle-ci se décline sur plusieurs axes. Tout d'abord, il faut s'intéresser à l'applicabilité de la méthode que l'on souhaite mettre en place. Il faut aussi s'assurer de la viabilité industrielle de la proposition, i.e. s'assurer que l'investissement n'est pas trop important comparé aux résultats. Finalement, il faut aussi savoir intégrer les nouveaux processus dans ceux qui sont actuellement en place.

C'est avec cela en tête que l'équipe s'est donc intéressée à l'amélioration des méthodes de diagnostic déployées dans les produits. Très rapidement, il est apparu que les méthodes à base de modèles sont particulièrement adaptées à la gestion des problèmes complexes. Ce dernier point étant la principale difficulté à traiter dans cette étude, l'orientation vers les modèles s'est faite naturellement.

En parallèle, l'équipe s'est intéressée aux travaux des autres équipes de Thales. Il était important d'étudier la possibilité de mettre en commun, avec ces équipes dont les domaines sont connexes à la maintenance, les efforts à déployer sur la modification des processus, l'acquisition de nouvelles compétence de modélisation .... Et en effet, les analyses de sûreté de fonctionnement s'en rapprochent. La chose la plus importante est que les deux domaines s'intéressent aux impacts des défaillances sur le système. L'équipe des études pour la sûreté de fonctionnement travaillant déjà sur des méthodes à base de modèles, et plus particulièrement sur des modèles basés sur le langage AltaRica, l'orientation vers les méthodes à base de modèles s'est vue confirmée.

Avant le départ de la thèse, l'utilisation de raisonnement à base de modèles pour faire du diagnostic était fortement pressentie (l'état de l'art effectué en début de thèse confirmant ensuite cette voie).

Les premières expérimentations débutent sur l'utilisation des modèles pour le diagnostic. Un premier système est modélisé en AltaRica, à l'aide de l'outil SIMFIA, puis directement en AltaRica version LaBRI pour comparer les deux méthodes. L'équipe modélise le comportement dysfonctionnel des composants d'un système, et souhaite ensuite utiliser les méthodes d'exploration et de vérification formelles de modèles pour manipuler les relations de causes à effets du système sous diagnostic.

À cette étape, l'exploitation des caractéristiques des méthodes à base de modélisation formelle est clairement engagée. La volonté de gérer les problèmes complexes et d'obtenir des résultats précis et exhaustifs est affichée. Pour obtenir de tels résultats, la priorité est d'utiliser des techniques de modélisation et d'analyses efficaces. Ainsi, l'utilisation d'outils avec des interfaces graphiques évoluées comme *SIMFIA* (cf. 2.3.4.3.3 page 56) ou *OCAS* (cf. 2.3.4.3.4 page 58) n'est pas un critère important pour le choix d'un langage de modélisation. L'expressivité du langage et les possibilités de model-checking sont eux importants et ont fait pencher la balance du côté du langage AltaRica LaBRI.

5. ***B**ult-**I**n **T**est **E**quipment.*

Finalement, l'idée d'obtenir des équations encodant les relations de causes à effets du système afin d'en effectuer le diagnostic apparaît juste avant la thèse. Cela découle de l'observation de la similarité entre les relations de causes à effets pour le diagnostic et les arbres de fautes utilisés en sûreté de fonctionnement. Ces derniers, générés automatiquement à partir de modèles AltaRica par des méthodes de calcul dits de coupes minimales, ont été à l'origine de la volonté d'avoir un processus de génération de règles de diagnostic automatisé.

Ces expérimentations, ces choix, ces volontés d'orientations des travaux sont le point de départ du travail de thèse décrit dans ce document.

7.2 Avancée des travaux industriels avec la thèse

Maintenant que nous avons une idée précise du contexte dans lequel nous avons débuté notre travail, nous pouvons présenter l'évolution des travaux dans l'entreprise durant la thèse. Les travaux du groupe d'études et le travail de thèse ont évolué ensemble, chacun profitant des avancées de l'autre.

Dans cette section nous présentons la solution mise en place dans le groupe d'études qui comprend la partie industrielle des travaux de thèse. Cette méthode s'appuie sur le travail d'état de l'art effectué sur les différentes approches pour faire du diagnostic (cf. chapitre 2) et sur les conclusions tirées à partir de celui-ci et présentées en section 2.4. L'article [KGS+11] publié lors de conférence DX présente en partie cette solution.

7.2.1 Solution de diagnostic

Étant donné le contexte dans lequel s'inscrivent nos travaux, prendre en compte les contraintes industrielles était un point très important. Afin de proposer une solution viable, nous nous sommes appliqués à ce que les solutions que nous proposons prennent en compte au maximum ces contraintes, et ce évidemment dans les travaux appliqués du groupe d'études mais aussi dans ceux développés dans le cadre de la thèse.

La solution de diagnostic que nous proposons a été bâtie pour prendre en compte les difficultés de l'application d'une approche à base de modèle dans le milieu industriel, ainsi que les contraintes dues au contexte, à savoir principalement :

- La difficulté de construction du modèle.
- La nécessité de proposer une solution qui ne soit pas déconnecté des procédures industrielles actuelles.
- L'assurance de la cohérence entre système et modèle.
- La facilité à mettre à jour le modèle et à y intégrer des modifications.
- L'importance de la réutilisabilité de la méthode et des modèles pour plusieurs systèmes.
- L'importance de la polyvalence des modèles et de l'ouverture de la méthode à des domaines autres que le diagnostic pour la maintenance.
- La taille conséquente des modèles.
- Les objectifs de performance.
- Les contraintes d'embarcabilité.

Le premier choix, celui du langage de modélisation, s'est porté, comme le préconisait l'état de l'art des approches pour le diagnostic, et comme nous l'avons précisé dans la section qui précède, sur le langage AltaRica.

Ensuite, pour adresser la difficulté de modélisation et la réutilisabilité de la méthode pour différents systèmes, nous avons travaillé sur la génération automatique de code AltaRica. L'idée est de générer le modèle à partir de documents de description de systèmes.

Finalement, nous avons décidé de générer à partir du modèle les règles de diagnostic. Cela nous permet alors d'aboutir à une solution proche de la méthode utilisée dans la pratique et de rejoindre les procédures industrielles actuelles. De plus, il apparaîtra rapidement qu'effectuer des analyses directement sur certains modèles, du fait de leur taille, est impossible en l'état. Passer par les coupes minimales nous amène alors une solution pour effectuer certaines de ces analyses.

Notre solution de diagnostic s'articule autour de trois phases illustrées en figure 7.1 :

1. La génération automatique du modèle AltaRica.
2. La génération des règles de diagnostic.
3. L'utilisation des règles pour la mesure de performances ou pour le diagnostic.

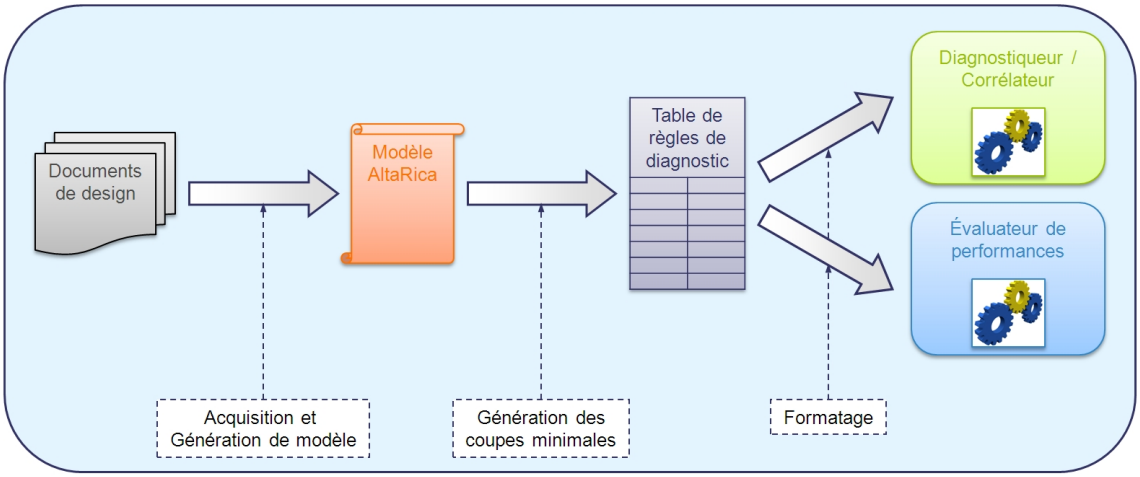


FIGURE 7.1 – Chaîne de diagnostic

De cette solution développée dans le groupe d'études durant la thèse, nous développerons les deux premières phases en expliquant le processus de modélisation puis en décrivant la méthode utilisée pour l'obtention des règles de diagnostic. Finalement nous expliquerons comment les métriques de diagnostic définies dans ce document ont été intégrées dans le processus actuel.

7.2.2 Modélisation de systèmes avioniques sous diagnostic

La conviction de l'utilisation de modèles formels pour représenter les systèmes sous diagnostic était donc déjà présente avant la thèse. Cette volonté s'est transformée en choix à la suite du travail d'état de l'art sur les différentes approches pour faire du diagnostic effectué en début de thèse.

7.2.2.1 Modélisation par couche

Les systèmes avioniques sont composés typiquement de plusieurs types de composants. Ils peuvent être répartis dans deux catégories de composants :

- Les composants physiques.
- Les composants fonctionnels.

Les composants fonctionnels doivent remplir un rôle précis dans le système et les composants physiques sont présents pour supporter les fonctionnels afin que ces derniers puissent accomplir leur travail.

Nous retrouvons parmi les composants physiques par exemple :

- Les unités de calcul (CPM [6]) utilisées pour les tâches nécessitant de la capacité de calcul.
- Les composants d'entrée-sortie (IOM [7]) qui sont utilisés par les fonctions pour émettre ou recevoir des messages ou des données.
- Les composants réseaux (*switches*, câbles, ports ...) qui transportent les messages et les données.
- Les capteurs qui donnent des mesures de données comme par exemple la pression ou la température.

Parmi les composants fonctionnels, nous trouvons :

- Les fonctions de calcul.
- Les fonctions d'entrée-sortie qui implémentent un protocole et sont en charge de l'envoi et de la réception de messages et de données.
- Les fonctions de comparaison qui servent à comparer plusieurs valeurs.
- Les fonctions de monitoring dont le rôle est de détecter, par exemple, lorsque la valeur relevée par un capteur est anormale.

Ces fonctions de monitoring sont très importantes pour le diagnostic car ce sont elles qui fournissent les observations utilisées par le diagnostiqueur pour trouver les causes d'une situation problématique. Les valeurs d'observation correspondent au résultat de l'envoi d'un message par une fonction de monitoring.

Pour modéliser un système avionique, nous allons nous appuyer sur les deux catégorisations des composants. Nous modélisons donc un système par couches : une couche physique et une couche fonctionnelle. Cette structuration en couche est très importante pour apporter de la souplesse dans les analyses en permettant d'effectuer des modifications de manière simple et précise, sans remettre en cause l'ensemble du modèle.

Du point de vue d'AltaRica, notre nœud principal sera composé de deux sous-nœuds, l'un décrivant la couche physique, et l'autre décrivant la couche fonctionnelle. Dans le nœud principal, nous décrirons une couche virtuelle, que nous appelons *couche logique*, qui aura pour rôle d'associer les composants fonctionnels aux composants physiques.

7.2.2.2 Génération automatique de modèle

Notre travail se déroulant dans le domaine industriel et dans le but d'une utilisation dans l'entreprise, il nous est apparu comme très important de prendre en compte les contraintes induites par ce contexte.

Nous avons vu dans le chapitre 2 que la principale difficulté à l'utilisation d'une approche à base de modèle est la phase de modélisation du système. Cette opération requiert du temps et de l'application qu'il ne faut pas négliger. Mais une fois celle-ci effectuée, il est possible de profiter de tous les avantages du raisonnement à base de modèle.

Ce que nous avons aussi vu dans le chapitre 2, c'est qu'une approche à base de modèle est totalement appropriée pour la description hiérarchique et modulaire de modèle, et pour la réutilisation des modules de celui-ci.

Avec cela en tête, nous avons décidé de mettre en place un processus de génération automatique de modèle. Pour cela, nous devons utiliser les informations que nous avons sur le système pour

6. ***C**ore **P**rocessing **M**odule.*
7. ***I**n/**O**ut **M**odule.*

définir des règles de génération. De plus, nous voulons pouvoir générer du code décrivant différents systèmes.

Pour être aussi proche que possible du design courant des systèmes, nous prenons comme entrée pour la génération de modèle un ensemble de documents qui sont fournis pour chacun des systèmes et qui correspondent aux documents d'ingénierie produits par les concepteurs système. Ces documents sont principalement de deux types :

- Ceux qui décrivent comment chaque composant du système peut défaillir. Nous trouvons par exemple dans cette catégorie des documents comme les AMDE [8] (ou FMEA [9] en anglais), qui sont requis lors de la procédure standard de certification avionique, comme décrit dans l'ARP4754A [SAE10].
- Ceux qui décrivent comment les composants sont connectés entre eux. Dans cette catégorie nous trouvons plusieurs types de documents décrivant le câblage, les flux de données fonctionnelles, le routage réseau, etc. dont le nom et le format dépendent de la procédure mise en place dans l'entreprise.

De ces documents, nous ne conservons que les informations qui sont pertinentes du point de vue de la maintenance et du diagnostic, comme les modes de défaillances des composants, la façon dont les composants sont interconnectés, les différentes fonctions du système qui sont en charge du monitoring, les différents messages de maintenance envoyés par les fonctions, ...

Dès lors que nous avons les informations sur le comportement dysfonctionnel des composants et sur la façon dont ils sont reliés, nous pouvons définir des règles de génération pour encoder ces données dans un modèle AltaRica. Parmi ces règles, on retrouve par exemple les règles suivantes :

- Un composant physique ou une fonction sera représenté par un nœud dans le modèle AltaRica.
- L'état d'un composant sera encodé dans une variable d'état du nœud correspondant.
- Les changements de mode de panne sont modélisés en utilisant des transitions étiquetées par des événements de défaillances dans le nœud correspondant.
- Les messages et données seront des variables de flux dans le modèle AltaRica.

Avec ces règles de génération nous sommes capables de créer un modèle AltaRica représentant un système en accord avec la façon dont il a été conçu par les ingénieurs système.

S'il a été plutôt facile en pratique de trouver et de traiter automatiquement les documents décrivant la façon dont les composants sont connectés les uns aux autres, il n'est en revanche pas évident d'obtenir automatiquement les informations concernant le comportement dysfonctionnel des composants. En effet, le format des fichiers décrivant ce comportement n'est pas forcément exploitable automatiquement, de par le fait que les informations ne sont pas tout le temps organisées de la même façon dans ces fichiers, ou simplement de par leur format informatique (pdf, texte brut, etc.).

Dans le cas où générer automatiquement le comportement des composants n'est pas possible, il faut le renseigner manuellement. Mais en utilisant les propriétés de l'approche à base de modèle et du langage AltaRica, on peut s'appuyer notamment sur la notion de bibliothèque de composants pour rendre le travail moins difficile, plus efficace, générique et réutilisable. La génération du modèle est dans ce cas semi-automatique.

La génération automatique d'un modèle correspondant au système que l'on veut analyser a plusieurs avantages. Tout d'abord, le modèle étant formel, nous pouvons profiter des avantages du raisonnement à base de modèle, à savoir la possibilité d'effectuer des analyses formelles et du model-checking. En outre, de par l'aspect automatique de la méthode, nous prenons en compte et gérons la plus grande difficulté de l'approche à base de modèle qui est la création du modèle. Il y a aussi d'autres avantages du fait du contexte industriel particulier des travaux, à Thales. En

8. **A**nalyse des **M**odes de **D**éfaillances et de leurs **E**ffets.
9. ***F**ailure **M**ode **E**ffects **A**nalysis.*

effet, la création d'un modèle dysfonctionnel représentant les défaillances du système est un travail que font les ingénieurs en sûreté de fonctionnement. La génération de ce modèle est donc aussi très intéressante et proche de ce qu'ils décrivent dans leur contexte. Une synergie et un travail commun peuvent alors être envisagés.

7.2.3 Analyse et exploitation de modèles avioniques pour le diagnostic

7.2.3.1 Taille des modèles

Les systèmes que nous souhaitons pouvoir traiter sont de types divers. Cela peut aller d'un sous-système qui correspond à la partie du système dont nous sommes responsables d'un point de vue diagnostic, à une suite avionique complète.

Les comportements des systèmes avioniques du point de vue du diagnostic pour la maintenance sont relativement simples. Dans ce cadre, nous nous intéressons uniquement aux défaillances des équipements, à leurs moyens de détection et aux messages de monitoring produits vers le diagnostiqueur.

À côté de cela, les périmètres des systèmes pouvant être larges, le nombre de composants, de messages, d'événements et d'états à considérer peut être très important. La complexité faible des systèmes du type de ceux que nous considérons ne suffit pas à assurer qu'ils seront faciles à traiter. Dans notre contexte, nous nous retrouvons avec des systèmes dont le nombre de composants peut varier de quelques uns dans le cadre de petits sous-systèmes à plusieurs dizaines pour des sous-systèmes représentatifs d'un système avionique complet (représentatif au sens où le sous-système rassemble l'ensemble des types de monitoring, de procédés de communication et de défaillances que l'on retrouve dans le système global), jusqu'à plusieurs centaines et plus d'un millier de composants pour les systèmes avioniques complets.

Malgré ces comportements peu complexes, au vu du nombre de composants qui peuvent composer un système sous diagnostic à traiter, on peut aisément imaginer que le comportement global du système puisse être difficile à calculer.

En effet, une fois un système de ce type modélisé en AltaRica, calculer l'automate de comportement le caractérisant peut être impossible. Le nombre de variables peut être tel que le calcul explose rapidement en mémoire. Analyser un modèle AltaRica en utilisant l'ensemble des configurations et des transitions comme nous l'avons décrit en 6.3.1.3 se révèle impossible dans ce type de situation. Il nous faut donc réfléchir à une méthode différente pour obtenir les relations de causes à effets dans le cadre de systèmes avioniques à périmètres larges.

S'il est impossible de raisonner sur l'automate de comportement global pour caractériser les performances de diagnostic d'un modèle, une idée peut être d'obtenir les informations de relations de causes à effets du modèle à travers des coupes ou séquences.

Un premier moyen d'éviter ce problème serait d'obtenir ces relations par un calcul de type *Pre*. Mais cette solution ne fonctionne pas non plus dans le cas de larges systèmes avioniques. En effet ces derniers contiennent un grand nombre d'événements de défaillance, et étant donné que les défaillances sont non contrôlables et souvent indépendantes, alors le degré entrant des configurations se trouve être trop important pour permettre un calcul de type *Pre*.

L'analyse des performances de diagnostic des modèles de systèmes avioniques de ce type n'est pas possible en l'état. Une idée est donc de chercher à réduire la taille du modèle que nous analysons.

7.2.3.2 Réduction de modèle

Arrivant devant la conclusion que nous ne pouvions pas analyser en l'état certains larges modèles avioniques Thales, nous avons engagé un travail de collaboration entre le LaBRI et Thales Avionics Toulouse. La technique de réduction de modèle AltaRica est un fruit de ce travail.

L'ensemble de ce qui est décrit dans cette partie est extrait du rapport de recherche [GPKV11]. On trouve d'ailleurs dans ce dernier plus de détails sur la réduction de modèles AltaRica et sur les objets manipulés lors de cette réduction.

7.2.3.2.1 Principe de la réduction

L'algorithme de réduction est appliqué sur l'automate à contraintes (cf. définition 46 page 156) qui représente le modèle. L'automate que l'on considère et qui correspond à un nœud AltaRica est obtenu à travers une opération de *mise à plat* du modèle (suppression de la hiérarchie et création d'un nœud unique). On obtient donc la *sémantique à plat* du modèle [Poi00].

Étant donné l'automate à contraintes $\mathcal{A}$ qui représente le système sous diagnostic, nous voulons évaluer une formule booléenne ϕ construite avec des variables de $\mathcal{A}$. ϕ est appelée *formule cible*. Notre but est de réduire $\mathcal{A}$ aux seules variables, assertions, transitions, qui sont nécessaire pour évaluer ϕ sur tout l'automate.

Plus précisément, l'automate réduit $\mathcal{A}^\phi$ satisfait les propriétés suivantes :

- Pour toute configuration σ de $\mathcal{A}$, on peut obtenir sa projection σ' dans $\mathcal{A}^\phi$ telle que σ satisfait ϕ si et seulement si σ' satisfait ϕ.
- Pour toute séquence des macro-transitions de $\mathcal{A}$, sa restriction aux macro-transitions de $\mathcal{A}^\phi$ est une séquence exécutable dans $\mathcal{A}^\phi$.
- Toute séquence des macro-transitions de $\mathcal{A}^\phi$ est une séquence exécutable dans $\mathcal{A}$.

L'idée principale de l'algorithme de réduction est de calculer ce qui peut influencer la valeur de vérité de ϕ. À partir des variables qui apparaissent dans ϕ, on calcule les éléments de $\mathcal{A}$ qui sont liés à ses variables : les variables de flux sont influencées par les assertions, les assertions par des variables de flux et d'état, les variables d'état par les transitions, et ainsi de suite. Ces relations entre éléments de $\mathcal{A}$ permettent de restreindre $\mathcal{A}$ à un sous-ensemble de ses variables, les parties ne traitant pas de ses variables étant simplement ignorées.

Par la suite, nous utilisons quelques notations supplémentaires à celles introduites en même temps que l'automate à contraintes en section 6.1.2.3, ainsi qu'à celles introduites en fin de section 3.1. Si e est un terme ou une formule booléenne, alors $Var(e)$ représente l'ensemble des variables qui apparaissent dans e. Si $f : A \to B$ est une fonction de A vers B, on note $Dom(f) \subseteq A$ son domaine. De manière similaire, pour une variable v, $Dom(v)$ désignera l'ensemble des valeurs que peut prendre v.

7.2.3.2.2 Dépendances entre variables

La sémantique du langage AltaRica définit la façon dont les variables sont assignées à une valeur. Les variables de flux sont calculées selon les assertions, et les variables d'état sont modifiées lorsqu'une transition est tirée. On peut considérer qu'à cause des assertions, la valeur d'une variable v dépend de toutes les autres variables du modèle. C'est vrai, mais certaines variables peuvent être redondantes. Ainsi, trouver un ensemble de variables plus précis qui détermine la valeur de v peut servir à réduire significativement la complexité du modèle. On peut remarquer que cet ensemble de variables n'est pas unique : par exemple, considérons l'assertion $(a = (b \wedge c)) \wedge (c \neq d)$ construite avec quatre variables booléennes. a dépend de $\{b, c, d\}$; cet ensemble peut être réduit à $\{b, c\}$ mais aussi à $\{b, d\}$ car c et d sont fonctionnellement dépendantes. La suite décrit une méthode pour calculer un tel ensemble réduit de variables.

Étant donné un automate à contraintes $\mathcal{A} = \langle V_S, V_F, E, T, A, I\rangle$ et une formule $\phi \in BF(V_S \cup V_F)$, on note $Req_\mathcal{A}(\phi)$ l'ensemble des variables qui influencent la valeur de vérité de ϕ. Cet ensemble est construit de la manière suivante :

- $Var(\phi) \subseteq Req_\mathcal{A}(\phi)$
- $\forall a \in A \cup I,\ Var(a) \cap Req_\mathcal{A}(\phi) \neq \emptyset \implies Var(a) \subseteq Req_\mathcal{A}(\phi)$
- $\forall t = (g, e, \alpha) \in T,\ Dom(\alpha) \cap Req_\mathcal{A}(\phi) \neq \emptyset \implies Var(t) \subseteq Req_\mathcal{A}(\phi)$ où $Var(t)$ représente l'ensemble des variables apparaissant dans g et α.

Remarquons que cette construction dépend de la façon d'écrire l'assertion A. Néanmoins, l'algorithme de réduction est correct quelle que soit la façon dont est écrite l'assertion. Il est tout de même possible d'obtenir de meilleurs résultats si l'on écrit l'assertion d'une certaine manière ; nous expliquerons comment réécrire A dans ce but dans la prochaine section.

La réduction de $\mathcal{A}$ à ϕ, notée $\mathcal{A}^\phi$, est la restriction de $\mathcal{A}$ aux variables de $Req_{\mathcal{A}}(\phi)$. L'automate à contraintes $\mathcal{A}^\phi = \langle V_S^\phi, V_F^\phi, E^\phi, T^\phi, A^\phi, I^\phi \rangle$ est défini comme suit :

- $V_S^\phi = V_S \cap Req_{\mathcal{A}}(\phi)$
- $V_F^\phi = V_F \cap Req_{\mathcal{A}}(\phi)$
- $T^\phi = \{\langle g, e, \alpha \rangle \in T | Dom(\alpha) \cap Req_{\mathcal{A}}(\phi) \neq \emptyset\} \cup \{t_\epsilon\}$
- $E^\phi = \{e \in E | \exists (g, e, \alpha) \in T^\phi\}$
- $A^\phi = \{a \in A | Var(a) \cap Req_{\mathcal{A}}(\phi) \neq \emptyset\}$
- $I^\phi = \{i \in I | Var(i) \cap Req_{\mathcal{A}}(\phi) \neq \emptyset\}$

t_ϵ est la macro-transition $\langle \texttt{true}, \epsilon, \emptyset \rangle$ correspondant à une ϵ-transition.
$\mathcal{A}^\phi$ est suffisant pour calculer les séquences de transitions qui produisent une configuration satisfaisant ϕ. La preuve est donnée dans [GPKV11] en section 5.3.

7.2.3.2.3 Réduction en utilisant les dépendances fonctionnelles

Dans la plupart des cas, toutes les variables d'un automate à contraintes sont dépendantes les unes des autres et aucun gain n'est obtenue avec la réduction car $Req_{\mathcal{A}}(\phi) = V_S \cup V_F$. Cependant, il est possible d'obtenir un meilleur résultat si le système est partitionnable en sous-systèmes déconnectés les uns des autres.

La dépendance mutuelle des variables est due à la sémantique d'AltaRica qui lie les variables des composants en utilisant des assertions. D'un point de vue formel, les assertions lient les variables sans aucune notion d'orientation ou de causalité entre ces variables. Néanmoins, lorsque nous modélisons un système, nous pensons souvent les assertions comme des affectations, ou a minima, comme un flux de données d'un point à un autre. De ce fait, beaucoup de modèles possèdent une orientation implicite due à la façon dont on décrit les systèmes industriels, et souvent, de nombreuses variables de flux dépendent fonctionnellement d'autres variables. Ces dépendances fonctionnelles peuvent être utilisées pour supprimer des assertions et ainsi réduire les connexions entre les composants de l'automate.

La section suivante donne un algorithme qui calcule deux ensembles C et FD tels que :

- $C \subseteq BF(V_S \cup V_F)$, $FD \subseteq V_F \times T(V_S \cup V_F)$
- $C \cup \{(v = t) \in BF(V_S \cup V_F) \mid (v, t) \in FD\}$ et l'ensemble des assertions original A ont la même sémantique.
- $\forall (v, t), (v', t') \in FD, v = v' \implies t = t'$ i.e. chaque variable de flux apparaît au plus une fois dans la partie gauche de FD.

Chacun des couples $(v, f) \in FD$ associe à une variable de flux v un terme f qui définit la valeur de v. En utilisant ces dépendances fonctionnelles, il est possible de réduire davantage l'automate à contraintes avant l'étape de réduction donnée dans la section précédente. En réalité, chacune des variables de flux apparaissant en premier élément d'un couple de FD peut être remplacée par la fonction qui lui est associée. Cette variable peut ensuite être supprimée du modèle. Ce processus a pour avantages de réduire le nombre de variables de flux et de supprimer des dépendances du modèle. Les mêmes substitutions doivent bien sûr être appliquées sur les formules en entrée.

7.2.3.2.4 Calcul des dépendances fonctionnelles

Dans cette section nous donnons l'algorithme utilisé pour calculer les dépendances fonctionnelles entre les variables de l'automate à contraintes. Pour $\mathcal{A}$ l'ensemble des assertions de l'automate, l'algorithme crée deux nouveaux ensembles :

1. Le premier est un ensemble de dépendances fonctionnelles $FD \subseteq V_F \times T(V_S \cup V_F)$ qui associe à une variable de flux v un terme t construit sur $(V_S \cup V_F) \setminus \{v\}$. L'équation $v = t$ est un invariant du modèle qui est sémantiquement équivalent à une assertion appartenant à $\mathcal{A}$.
2. Le deuxième, $C \subseteq BF(V_S \cup V_F)$, contient les assertions pour lesquelles aucune dépendance fonctionnelle n'a été trouvée.

L'algorithme principal (`calculer-dependances-fonctionnelles` qui suit) est assez simple. Pour chaque assertion c, il cherche une dépendance fonctionnelle qui est sémantiquement équivalente (ligne 4), i.e. une variable $v \in Var(c)$ et un terme $f \in T(Var(c) \setminus \{v\})$ tels que les formules c et $v = f$ ont les mêmes solutions. Si aucun couple de ce type n'est obtenu, l'assertion est gardée telle quelle et est stockée dans C (lignes 5-6). Si une dépendance est trouvée, elle est ajoutée à FD. Pour éviter la création de cycles, nous tenons à jour une relation de dépendance DEP entre variables (lignes 8-11) ; si $(v, v') \in DEP$ alors v dépend de v'.

Algorithme 1: `calculer-dependances-fonctionnelles(`A`)`

```
1  FD ← ∅, C ← ∅
2  DEP ← ∅
3  pour chaque c ∈ A faire
4  |  (v, f) ← rechercher-fd(c, DEP)
5  |  si (v, f) = (⊥, ⊥) alors
6  |  |  C ← C ∪ {c}
7  |  sinon
8  |  |  FD ← FD ∪ {(v, f)}
9  |  |  pour chaque v' ∈ Var(f) faire
10 |  |  |  DEP ← DEP ∪ {(v, v')}
11 |  |  fin
12 |  fin
13 fin
14 retourner ⟨FD, C⟩
```

L'algorithme `rechercher-fd` page 197 vérifie s'il existe une dépendance fonctionnelle dans la sémantique d'une assertion c. L'algorithme commence par vérifier si l'assertion c est une égalité entre une variable de flux x et un autre terme t (lignes 2-3). Si c'est le cas et si les variables qui apparaissent dans t ne dépendent pas de x (ligne 4), alors le couple (x, t) est retourné.

Si aucune dépendance fonctionnelle n'est trouvée syntaxiquement, la recherche est réalisée sur la sémantique de c (lignes 15-21). Pour chaque variable de flux x apparaissant dans c qui ne dépend pas (w.r.t. DEP) d'autres variables de c, on vérifie si sa valeur est une fonction d'autres variables. Ce que l'on regarde ici plus précisément, c'est qu'il est impossible d'obtenir deux valuations $\mathcal{V}_1$ et $\mathcal{V}_2$ de variables de $Var(c)$ vérifiant l'assertion c (i.e. dans $[\![c]\!]$) qui ne diffèrent que par la valeur de x, car sinon cela signifierait que x ne dépend pas des autres variables de c. Ce test peut être réalisé par un solveur. Pour rappel, $[\![c]\!]$ représente l'ensemble des valuations des variables de c qui vérifient l'assertion c. $c[x/d_1]$ représente l'assertion c où la variable x est substituée par sa valeur d_1.

Finalement, à la ligne 18, `contr-fonc(`x`,`c`)` construit le terme $t \in T(Var(c) \setminus \{x\})$ tel que $x = t$ et c sont sémantiquement équivalents.

7.2.3.2.5 Exemple

Dans cette partie nous donnons un exemple d'utilisation de la réduction sur l'exemple simple qui nous a servi à illustrer l'ensemble de la partie II et que nous avons modélisé en section 6.3.1. Nous rappelons en figure 7.2 le code AltaRica modélisant ce système sous diagnostic.

Considérons le nœud `SUD` représentant l'ensemble du système sous diagnostic. L'automate à contraintes, obtenu à travers la mise à plat du modèle où le nœud principal est `SUD`, peut être affiché avec la commande ARC `ca`. Le modèle à plat résultant de la commande `arc> ca SUD` est représenté en figure 7.3 (page 198).

Algorithme 2: `rechercher-fd`$(c,\ DEP)$

```
1  // Test syntaxique
2  si c est de la forme T1 = T2 alors
3  |  pour chaque x ∈ {T1, T2} ∩ VF faire
4  |  |  si ∀v ∈ Var(c) \ {x}, (v, x) ∉ DEP* alors
5  |  |  |  si x = T1 alors
6  |  |  |  |  retourner (T1, T2)
7  |  |  |  sinon
8  |  |  |  |  retourner (T2, T1)
9  |  |  |  fin
10 |  |  fin
11 |  fin
12 fin
13
14 // Test sémantique
15 pour chaque x ∈ Var(c) ∩ VF faire
16 |  si ∀v ∈ Var(c) \ {x}, (v, x) ∉ DEP* alors
17 |  |  si ∀(d1, d2) ∈ Dom(x)^2, d1 ≠ d2 ⟹ [[c[x/d1]]] ∩ [[c[x/d2]]] = ∅ alors
18 |  |  |  retourner (x, constr-fonc(x, c))
19 |  |  fin
20 |  fin
21 fin
22 retourner (⊥, ⊥)
```

```
node Systeme
  state
    v1,v2,v3,v4,v5,v6,v7 : bool : parent;
  init
    v1 := false;
    v2 := false;
    v3 := false;
    v4 := false;
    v5 := false;
    v6 := false;
    v7 := false;
  event
    e1,e2,e3,e4,e5,e6,e7 : incont;
  trans
    not (v1|v2|v3|v4|v5|v6|v7) |- e1 -> v1 := true, v5 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e2 -> v2 := true, v6 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e3 -> v2 := true, v3 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e4 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e5 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e6 -> v3 := true, v4 := true;
    not (v1|v2|v3|v4|v5|v6|v7) |- e7 -> v7 := true;
    not (v1|v3|v4|v5|v7)&v2&v6 |- e1 -> v1 := true;
    not (v1|v4|v5|v6|v7)&v2&v3 |- e2 -> v5 := true;
edon
```

```
node Observateur
  flow
    o1,o2,o3,o4 : bool : public;
edon

node SUD
  sub
    S : Systeme;
    O : Observateur;
  assert
    // Relation d'observation
    O.o1 = S.v1
    O.o2 = S.v2
    O.o3 = S.v3
    O.o4 = S.v4
edon
```

FIGURE 7.2 – Modèle ALTARICA de l'Exemple simple

Il n'y a plus qu'un seul nœud, qui correspond au nœud supérieur fourni : le nœud `SUD`. Les variables sont renommées selon la hiérarchie d'origine pour assurer leur unicité. Les transitions ont aussi été réécrites et simplifiées si possible (cela n'a pas été le cas ici).

Des informations quantitatives à propos du nœud supérieur sont aussi données. On y voit par exemple que le modèle en l'état comporte quatre dépendances fonctionnelles. Il est en effet dans ce cas évident de voir que les quatre assertions sont de la forme $x = t$ où x est une variable de flux, t un terme construit sur $(V_F \cup V_S) \setminus \{x\}$, et où les variables des différentes assertions ne sont pas dépendantes.

```
// Constraint automaton of node 'SUD'
// statistics:
// number of variables : 11
//   flow variables : 4
//   state variables : 7
// max cardinality : 2
// number of events : 8
// number of functional dependencies : 4
// number of constraints : 0
// number of transitions : 10
node SUD
  flow // 4 flow variables
    'O.o1' : bool : public;
    'O.o2' : bool : public;
    'O.o3' : bool : public;
    'O.o4' : bool : public;
  state // 7 state variables
    'S.v1' : bool : parent;
    'S.v2' : bool : parent;
    'S.v3' : bool : parent;
    'S.v4' : bool : parent;
    'S.v5' : bool : parent;
    'S.v6' : bool : parent;
    'S.v7' : bool : parent;
  trans
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e1' -> 'S.v1' := true, 'S.v5' := true;
    (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4') and ('S.v2' and
('S.v6' and not ('S.v5'))))))) |- 'S.e1' -> 'S.v1' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e2' -> 'S.v2' := true, 'S.v6' := true;
    (not ('S.v7') and (not ('S.v1') and (not ('S.v4') and ('S.v2' and ('S.v3' and (not
('S.v5') and not ('S.v6'))))))) |- 'S.e2' -> 'S.v5' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e3' -> 'S.v2' := true, 'S.v3' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e4' -> 'S.v3' := true, 'S.v4' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e5' -> 'S.v3' := true, 'S.v4' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e7' -> 'S.v7' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6'))))))) |- 'S.e6' -> 'S.v3' := true, 'S.v4' := true;
  edon
```

```
  init
    'S.v1' := false,
    'S.v2' := false,
    'S.v3' := false,
    'S.v4' := false,
    'S.v5' := false,
    'S.v6' := false,
    'S.v7' := false;
  event // 8 events
    'S.e6' : incont;
    'S.e7' : incont;
    'S.e5' : incont;
    'S.e4' : incont;
    'S.e3' : incont;
    'S.e2' : incont;
    'S.e1' : incont;
  assert // 4 functional dependencies
    'O.o3' = 'S.v3';
    'O.o1' = 'S.v1';
    'O.o2' = 'S.v2';
    'O.o4' = 'S.v4';
```

FIGURE 7.3 – Automate à contrainte du modèle simple

Considérons maintenant que nous nous intéressons à la formule $\phi_{ex} = (o_1)$. Le calcul de l'automate réduit $A^{\phi_{ex}}$ peut être obtenu avec ARC à travers la commande `arc>target-reduction SUD "O.o1"`. Le modèle réduit est donné en figure 7.4.

On remarque que les variables de flux ont toutes été supprimées, ainsi que les assertions. En outre, la formule cible a été réécrite en $\phi_{ex} = (S.v1)$.

En revanche le système après réduction conserve les mêmes dépendances entre variables d'états, événements et transitions. Les graphes de dépendances sont d'ailleurs proposés en figures 7.5 et 7.6 respectivement avant et après réduction.

Si la réduction a permis de supprimer toutes les variables de flux et toutes les assertions, cela n'a pas simplifié le comportement du système pour la formule ϕ_{ex}. L'exemple simple est bâti de telle façon que l'on ne peut pas casser des dépendances.

Nous voyons avec cet exemple que la réduction ne fonctionne pas sur tous les systèmes. Il est nécessaire d'avoir des systèmes qui intègrent des comportements parallèles et indépendants afin que la réduction puisse avoir un intérêt. Pour illustrer cela, prenons un exemple plus proche des systèmes avioniques que nous avons à analyser, car basé sur des flux d'entrée et de sortie, et des variables d'observations regardant certaines de ses entrées et sorties.

```
// statistics:
// number of variables : 7
//   flow variables : 0
//   state variables : 7
// max cardinality : 2
// number of events : 8
// number of functional dependencies : 0
// number of constraints : 0
// number of transitions : 10
node SUD
  state // 7 state variables
    'S.v1' : bool : parent;
    'S.v2' : bool : parent;
    'S.v3' : bool : parent;
    'S.v4' : bool : parent;
    'S.v5' : bool : parent;
    'S.v6' : bool : parent;
    'S.v7' : bool : parent;
  trans
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e6' -> 'S.v3' := true, 'S.v4' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e1' -> 'S.v1' := true, 'S.v5' := true;
    ('S.v2' and ('S.v6' and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not
('S.v4') and not ('S.v5')))))) |- 'S.e1' -> 'S.v1' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e2' -> 'S.v2' := true, 'S.v6' := true;
    ('S.v2' and ('S.v3' and (not ('S.v7') and (not ('S.v1') and (not ('S.v4') and (not
('S.v5') and not ('S.v6')))))) |- 'S.e2' -> 'S.v5' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e3' -> 'S.v2' := true, 'S.v3' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e4' -> 'S.v3' := true, 'S.v4' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e5' -> 'S.v3' := true, 'S.v4' := true;
    (not ('S.v2') and (not ('S.v7') and (not ('S.v3') and (not ('S.v1') and (not ('S.v4')
and (not ('S.v5') and not ('S.v6')))))) |- 'S.e7' -> 'S.v7' := true;
edon
```

```
init
  'S.v1' := false,
  'S.v2' := false,
  'S.v3' := false,
  'S.v4' := false,
  'S.v5' := false,
  'S.v6' := false,
  'S.v7' := false;
event // 8 events
  'S.e7' : incont;
  'S.e5' : incont;
  'S.e4' : incont;
  'S.e3' : incont;
  'S.e2' : incont;
  'S.e1' : incont;
  'S.e6' : incont;
```

FIGURE 7.4 – Automate à contrainte du modèle simple

Considérons un système composé de quatre nœuds identiques. Les nœuds transmettent simplement leur entrée et leur sortie à moins qu'une défaillance arrive, auquel cas la sortie vaut `false`. Les nœuds sont organisés en deux colonnes. Sur la colonne de droite, les nœuds $C01$ et $C11$ reçoivent en entrée la conjonction des sorties des nœuds $C00$ et $C10$ de la colonne de gauche. En figure 7.7, nous représentons le système, le code ALTARICA le modélisant ainsi que des données sur le nœud supérieur (`Systeme`).

Considérons maintenant que nous nous intéressons à la formule $\phi_{ex} = (o_1)$ qui vérifie que la variable de sortie o_1 est vraie. Le système réduit pour cette formule est présenté en figure 7.8.

On remarque que pour ce type système, la réduction est efficace. En effet, on voit que toutes les caractéristiques du modèle propres au composant $C11$ ont simplement été supprimées dans le modèle réduit. Il est clair ici que la valeur du flux de sortie o_1 n'est pas dépendant de l'autre variable de sortie o_2 et de l'ensemble des entrées-sorties du composant. Cela est visible sur les graphes de dépendances du système avant et après réduction présentés respectivement en figures 7.10 et 7.11 en pages 203-204.

En outre, on voit que d'autres variables de flux ont été supprimées puisque correspondant à des dépendances fonctionnelles. L'ensemble des dépendances fonctionnelles du système avant réduction est présenté en figure 7.9.

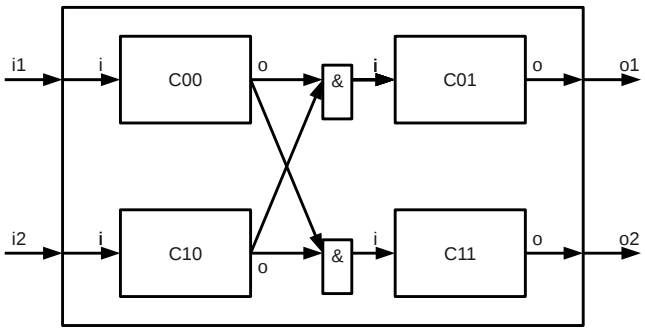
Le système réduit va donc ne conserver que les variables qui sont des feuilles descendantes des variables présentes dans la formule considérée. Ici, pour la formule $\phi_{ex} = (o_1)$, on ne conserve donc que les variables $C00.i$, $C00.s$, $C10.i$, $C10.s$ et $C01.s$. La variable o_1 disparaît elle aussi ; la formule est donc réécrite par ARC, ce qui donne :

FIGURE 7.5 – Exemple simple : Graphe de dépendance du nœud SUD avant réduction

FIGURE 7.6 – Exemple simple : Graphe de dépendance du nœud SUD après réduction

```
node C
  flow i, o : bool
  state s : { ok, ko };
  init s := ok;
  event fail;
  trans
   s = ok |- fail -> s := ko;
  assert
    if (s = ok)
      then (o = i)
      else (o = false);
edon

node System
  flow i1, i2, o1, o2 : bool;
  sub C00, C01, C10, C11 : C;
  assert
    i1 = C00.i;
    i2 = C10.i;
    C01.i = (C00.o & C10.o);
    C11.i = (C00.o & C10.o);
    C01.o = o1;
    C11.o = o2;
edon
```




```
=== System ===
statistics:
number of variables : 16
flow variables : 12
state variables : 4
max cardinality : 2
number of events : 5
number of functional dependencies : 10
number of constraints : 0
number of transitions : 5
```

FIGURE 7.7 – Exemple d'un système avec des flux d'entrée et de sorties

```
// statistics:
//  number of variables : 5
//    flow variables : 2
//    state variables : 3
//  max cardinality : 2
//  number of events : 4
//  number of functional dependencies : 0
//  number of constraints : 0
//  number of transitions : 4
node System
 flow // 2 flow variables
   'C00.i' : bool;
   'C10.i' : bool;
 state // 3 state variables
   'C00.s' : { ok, ko };
   'C01.s' : { ok, ko };
   'C10.s' : { ok, ko };
 init
   'C10.s' := ok,
   'C01.s' := ok,
   'C00.s' := ok;
 event // 4 events
   'C00.fail';
   'C01.fail';
   'C10.fail';
 trans
   ('C10.s' = ok) |- 'C10.fail' -> 'C10.s' := ko;
   ('C01.s' = ok) |- 'C01.fail' -> 'C01.s' := ko;
   ('C00.s' = ok) |- 'C00.fail' -> 'C00.s' := ko;
edon
```

FIGURE 7.8 – Automate réduit de l'exemple de la figure 7.7 pour la formule (o_1)

```
(('C01.s' = ok) and (('C10.i' and ('C10.s' = ok)) and ('C00.i' and ('C00.s' = ok))))
```

Finalement, on remarque les caractéristiques du graphe de comportement des deux modèles

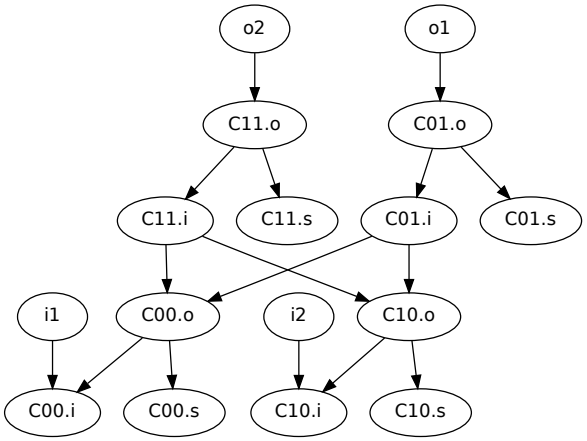


FIGURE 7.9 – Dépendances fonctionnelles

sont différentes. En effet, là où le modèle avant réduction comportait 64 configurations et 768 transitions, le modèle après réduction compte 32 configurations et 320 transitions. Cela signifie que la moitié du graphe de comportement est inutile pour décider ce que vaut la variable de sortie o_1.

7.2.3.3 Génération automatique de règles de diagnostic

La taille de certains systèmes avioniques que l'on souhaite étudier, et donc par là, celles des modèles les représentant, est trop importante pour pouvoir faire des analyses directement sur l'automate de comportement du système, ce dernier n'étant pas calculable pour ce type de modèles. L'idée de calculer des séquences ou des coupes sans calculer l'automate global à travers par exemple un algorithme de type *Pre* ne s'est pas avéré envisageable non plus du fait de la largeur des modèles de ce type.

Néanmoins, nous venons de voir qu'il est possible de réduire un modèle en fonction de sa nature. En effet, la réduction fonctionne particulièrement bien sur des modèles présentant des comportements parallèles et indépendants. C'est le cas des systèmes avioniques, dont les conditions d'envoi et d'arrivée des messages de monitoring ne portent pas sur tout le système mais uniquement sur une partie de celui-ci.

Il est dès lors envisageable de calculer les coupes et les séquences à partir d'une formule. En effet, chercher les coupes ou séquences vérifiant une formule dans le système global revient à vérifier les coupes et les séquences vérifiant la formule dans le modèle réduit pour cette formule (nous sommes sûrs de conserver dans le modèle réduit toutes les caractéristiques du modèle global qui sont susceptibles de modifier la valeur des variables de la formule).

7.2.3.3.1 Calcul des coupes de défaillances

Dans cette partie nous décrivons le calcul de coupes qui est implémenté dans ARC à la suite de la collaboration entre THALES AVIONICS et le LABRI, à laquelle j'ai participé, notamment pour la prise en compte des contraintes industrielles. Ce travail a été effectué par l'équipe ALTARICA du LABRI dans laquelle je travaillais durant ma thèse. Ce que nous allons expliquer ici est détaillé dans [GPKV11].

Nous rappelons que les coupes ont été définies en section 3.6. Nous y avons décrit aussi les notions d'événements visibles, de chemin, etc. que nous réutiliserons dans ce qui suit.

Dans [Rau02], A. Rauzy propose un algorithme qui calcule les coupes à partir d'un graphe

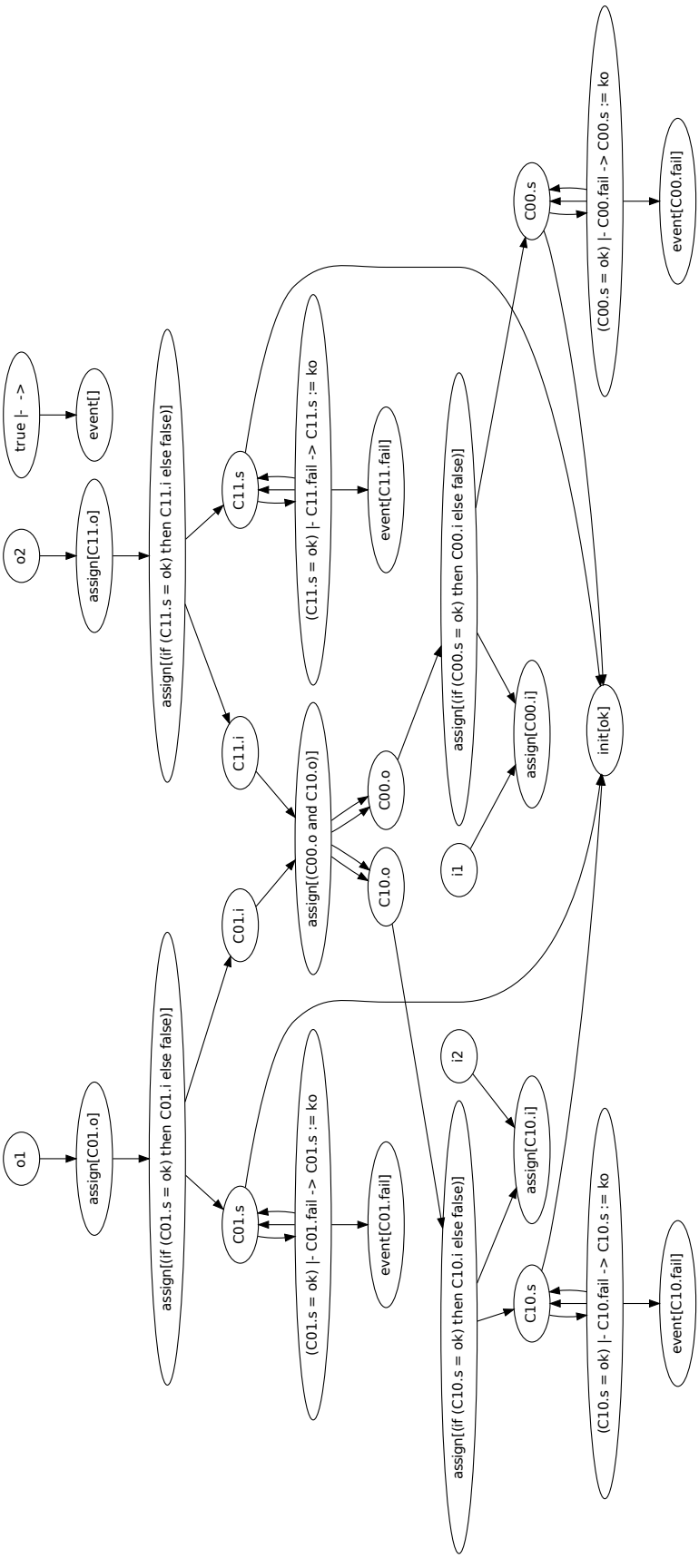


FIGURE 7.10 – Exemple simple : Graphe de dépendance du nœud `System` avant réduction

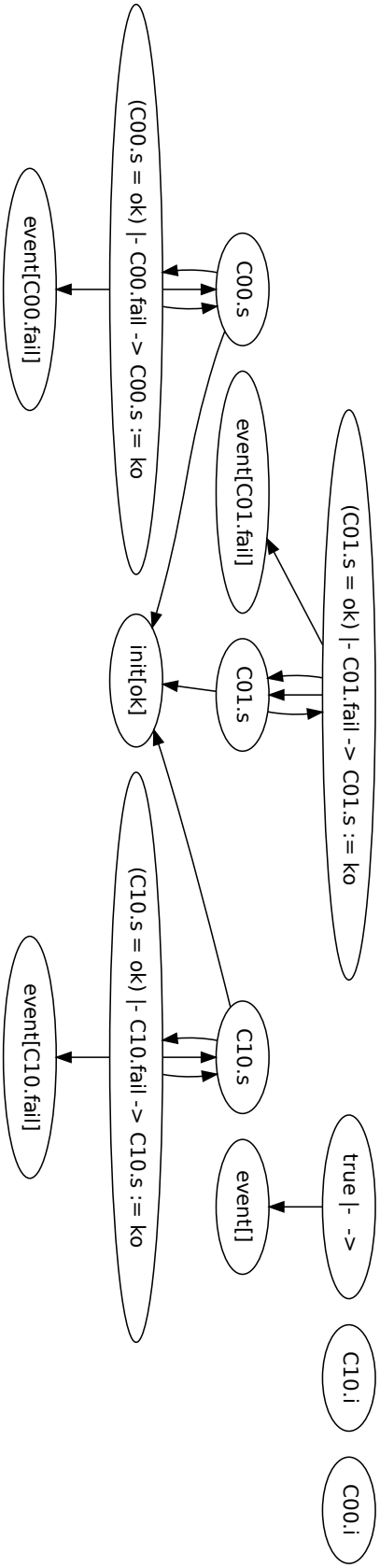


FIGURE 7.11 – Exemple simple : Graphe de dépendance du nœud `System` après réduction

d'état représentant la sémantique d'un modèle AltaRica. Le principe de cet algorithme est de calculer des couples $(c, \vec{e})$ où c est une configuration et $\vec{(e)}$ est un vecteur de bits (un bit $\vec{e}[e_i]$ par événement visible e_i) tels que :

1. Il existe un chemin p allant d'une configuration initiale à c ;
2. p est étiqueté par des événements e_i tels que $\vec{e}[e_i] = 1$.

Chaque vecteur $\vec{e}$ est ensuite traduit en une formule booléenne $F_{\vec{e}}$:

$$F_{\vec{e}} \stackrel{\text{def}}{=} \bigwedge_{\vec{e}[e_i]=1} e_i \wedge \bigwedge_{\vec{e}[e_i]=0} \neg e_i$$

La formule F_ϕ, qui représente les coupes pour une spécification d'états redoutés ϕ, est la disjonction des formules $F_{\vec{e}}$ pour lesquelles il existe une configuration qui satisfait ϕ :

$$F_\phi \stackrel{\text{def}}{=} \bigvee_{\{(c,\vec{e})|c\models\phi\}} F_{\vec{e}}$$

L'algorithme 3 est celui proposé par A. Rauzy dans [Rau02] (l'auteur n'en dit pas plus sur la structure de données utilisée par l'algorithme) ; Nous passons sur les détails liés aux *automates de mode*.

Algorithme 3: Calcul des vecteurs encodant les coupes

```
1 C ← {⟨c, 0⃗⟩|c ∈ Initial}, D ← ∅
2 tant que C ≠ ∅ faire
3 |   Soit ⟨c, e⃗⟩ ∈ C
4 |   C ← C \ {⟨c, e⃗⟩}, D ← D ∪ {⟨c, e⃗⟩}
5 |   pour chaque transition t = ⟨c, e_i, c'⟩ faire
6 |   |   C ← C ∪ {⟨c', e⃗[e_i] ← 1⟩}
7 |   fin
8 fin
```

L'algorithme de Rauzy correspond essentiellement au calcul d'accessibilité des configurations augmentées avec les vecteurs $\vec{e}$. Une façon d'implémenter cet algorithme est d'embarquer dans le modèle les vecteurs à travers des variables booléennes. Dans l'implémentation que nous proposons, nous augmentons le modèle en lui ajoutant un observateur qui enregistre les occurrences des événements visibles avec des variables booléennes. Cette augmentation se découpe en plusieurs étapes :

1. Ajouter à l'automate à contraintes une nouvelle variable booléenne v_e pour chaque événement visible e. Chaque variable v_e est initialisée à faux.
2. Pour chaque macro-transition étiquetée par un événement visible e, ajouter l'affectation $v_e \leftarrow$ `true`.

Cette opération d'augmentation est illustrée en figure 7.12.

Sur l'exemple considéré, nous avons ajouté une nouvelle variable booléenne `fails_event` initialisée à `false` qui passe à `true` lorsque l'événement visible `fails` est déclenché.

Les variables ajoutées ne modifient pas le comportement du système car leur valeur n'est jamais utilisée dans le modèle. Elles ne sont là que pour prendre une valeur qui sera utilisée ensuite en post-traitement pour calculer les coupes. L'automate de comportement du modèle augmenté vis-à-vis de celui du modèle de départ se comporte exactement de la même façon qu'un SUD par rapport au système qu'il diagnostique. Les sémantiques sont bisimilaires et les séquences d'événements sont conservées (ainsi que les coupes, par définition).

À partir du modèle augmenté, le calcul des coupes est simplement basé sur un calcul d'accessibilité :

```
node Generator
   flow
     out : bool;
   state
     mode : { ok, ko };
     on : bool;
   init
     mode := ok, on := false;
   event
     push;
     fails : visible;
   assert
     if mode = ko then out = false else
          out = true;
   trans
      mode = ok |- push -> on := not on;
      mode = ok |- fails -> mode := ko;
edon
```

```
node ObservedGenerator
   flow
     out : bool;
   state
     mode : { ok, ko };
     on : bool;
   init
     mode := ok, on := false;
   state
     fails_event : bool;
   init
     fails_event := false;
   event
     push, fails;
   assert
     if mode = ko then out = false else
          out = true;
   trans
      mode = ok |- push -> on := not on;
      mode = ok |- fails -> mode := ko,
           fails_event := true;
edon
```

FIGURE 7.12 – Exemple d'augmentation d'un modèle ALTARICA pour le calcul de coupes

1. Calculer symboliquement l'ensemble des configurations accessibles qui satisfont la formule ϕ pour l'automate augmenté (encoder par exemple le résultat avec un diagramme de décision (DD)).
2. Projeter les résultats obtenus sur les variables booléennes v_e (on obtient alors un diagramme de décision binaire (BDD) avec uniquement les variables d'événements v_e booléennes).
3. Traduire les résultats projetés en une formule booléenne (cela correspond à la satisfiabilité du BDD).

Avec cette méthode nous obtenons les coupes d'événements amenant dans une configuration vérifiant une formule ϕ.

Nous venons de montrer comment on peut obtenir les coupes d'un modèle ALTARICA. L'utilisation de la réduction du modèle permet de ne garder que le sous-modèle nécessaire aux calculs des coupes d'événements répondant à une formule cible. Ainsi le calcul de coupes d'événements s'avère être possible dans le cas de larges systèmes si ces derniers ont la caractéristique d'avoir un comportement présentant peu de dépendances.

À partir de ces coupes, nous pouvons obtenir les coupes minimales en appliquant un algorithme de minimisation ou bien exploiter la structure symbolique encodant les coupes pour minimiser directement lors du calcul de ces coupes. Si l'on est intéressé de savoir comment on peut calculer les coupes minimales, ou encore les séquences et les séquences minimales, on pourra notamment se reporter à [GPKV11] qui décrit les algorithmes implémentés dans ALTARICA pour ces calculs (dont l'algorithme de calcul de coupes que l'on vient de voir).

7.2.3.3.2 Coupes pour le diagnostic

Comme nous l'avons décrit en section 2.1.5, les méthodes historiquement utilisées pour faire du diagnostic dans le domaine de l'avionique et à THALES en particulier sont basées sur le raisonnement à base de règles.

Nous avons vu que les règles actuellement développées sont de type "messages ↔ coupables" (cf. 2.1.2.2.1) et définissent quels LRU ont pu amener le diagnostiqueur, local ou global, à recevoir un message.

Les coupes que nous avons calculées expliquent quels événements incontrôlables du système ont pu provoquer une observation (exprimée par une formule). Les coupes minimales expriment elles quels sont les événements qui sont les premiers implicants d'une observation, i.e. quels sont les événements qui ont pu faire *apparaître* l'ensemble des messages décrits par l'observation. Les coupes qui ne sont pas minimales représentent alors les coupes d'événements qui ont pu conserver l'observation une fois qu'elle est apparue.

Les messages de monitoring étant représentés par des variables d'observation dans notre système sous diagnostic, un ensemble de messages de monitoring qui sont apparus est encodé par une observation dans notre SUD. A minima, nous pouvons donc aussi encoder l'apparition d'un message de monitoring comme une observation.

À côté de cela, à partir des éléments des coupes que sont les événements incontrôlables, il est facile d'obtenir les LRU auxquels sont associés lesdits événements. Nous comprenons donc qu'à travers un calcul de coupes, on peut obtenir une table de règles du type de celles qui sont décrites aujourd'hui par les ingénieurs. Plus précisément, on peut obtenir cette table à partir des coupes minimales, car les LRU présents dans les tables de règles sont réellement ceux qui peuvent avoir déclenché le monitoring, et qui correspondent donc à la notion d'implicants premiers. Il apparaît donc qu'il est possible de créer la table de règles de diagnostic de manière automatique et exhaustive à travers un calcul de coupes minimales.

Plus intéressant, nous pouvons créer une table de règles plus précise de type "observations $\leftrightarrow$ événements" (cf. 2.1.2.2.2) qui exploite un peu plus les résultats des coupes minimales. Il s'agit ici, plutôt que d'associer seulement des LRU à un message de monitoring, d'associer des événements à des observations, i.e. un diagnostic à un ensemble de messages de monitoring. On obtient ainsi automatiquement une table de règles précise et exhaustive.

La difficulté que l'on peut identifier ici est que dans le cas d'un système sous diagnostic large, si l'ensemble de messages de monitoring, et donc l'observation pour laquelle on veut calculer les coupes minimales concerne trop de messages de monitoring, il est possible que la réduction au sous-modèle de dépendance ne soit pas suffisante pour permettre de faire le calcul des coupes minimales.

Finalement, une solution hybride consiste à calculer pour chaque message de monitoring l'ensemble des coupes minimales. En se réduisant à une seule variable d'observation, cela nous permet d'assurer que la réduction pourra se faire en toute efficacité dès lors que le système le permet.

Obtenir un tel résultat est une avancée du point de vue industriel car la table obtenue est plus précise que celle construite habituellement par les ingénieurs puisqu'elle porte non pas sur les LRU mais directement sur les défaillances. Une telle précision permettra au diagnostiqueur utilisant la table de règles de raisonner sur les défaillances directement et ainsi d'être plus précis dans son diagnostic final.

L'autre point évident qui montre l'intérêt de cette méthode, c'est le fait que ces règles soient générées automatiquement. Nous obtenons ainsi des résultats exhaustifs et précis pour des systèmes de plus en plus complexes. Il est de plus en plus difficile, et même réellement impossible, d'en assurer autant lorsque les tables sont écrites manuellement et que les relations de causes à effets du système sont déduites par l'esprit humain. Tout cela est bien entendu relatif à la représentativité et la correction du modèle. Mais l'avantage de demander à l'humain de créer un modèle plutôt que d'écrire des règles, au-delà du fait qu'il soit plus "naturel" pour un esprit humain de décrire le comportement de composants plutôt que de déduire ce qui a pu amener à une observation, c'est qu'un modèle formel de type AltaRica par exemple peut être validé et vérifié à l'aide de techniques et d'outils avancés, en l'occurrence à l'aide du model-checking et d'ARC, alors qu'il n'est pas possible d'en faire autant pour vérifier une table de règles.

Avec la méthode de génération automatique de la table de règles, nous rejoignons les procédures actuelles de Thales de diagnostic pour la maintenance. Nous apportons ainsi une solution proche de celle utilisée actuellement, mais mieux adaptée au contexte qui se complexifie, et plus efficace.

7.2.3.3.3 La commande ARC `cuts`

ARC propose une commande `cuts` qui permet de calculer les coupes avec l'algorithme que nous avons présenté un peu plus tôt dans cette partie (7.2.3.3.1), ainsi que leur version minimale.

La commande `cuts` répond à la syntaxe suivante :

`cuts [`*`options`*`] `*`nodeid acceptance-condition`*

où les options sont :

`--visible-tags=`vt_1`,...,`vt_n

`--disabled-tags=`dt_1`,...,`dt_m

`--min`

`--enum`

La commande `cuts` permet donc de calculer donc l'ensemble des coupes et leur version minimales menant à des configurations vérifiant une *formule cible*. La formule *acceptance-condition* est une formule booléenne sur les variables du nœud *nodeid*.

La réponse à la commande est une formule booléenne qui encode les coupes, ou leur version minimale selon l'option choisie, qui peut être de deux formes différentes selon que l'on utilise ou pas l'option `enum` :

- Sans l'option `enum`, la formule booléenne résultat est donnée en utilisant la syntaxe de l'outil ARALIA [RD97].
- Avec l'option `enum`, ARC énumère l'ensemble des éléments de la solution.

Les options `visible-tags` et `disabled-tags` permettent respectivement de définir des événements comme visibles (on donne alors les attributs des événements à considérer en paramètre de l'option) et d'interdire à des événements d'apparaître dans les coupes.

Finalement l'option `min` permet d'effectuer un calcul de coupes minimales.

ARC implémente aussi une commandes `sequences` tout à fait similaire, la seule différence étant la nécessité de renseigner le paramètre `--order=k` qui définit une borne maximale pour les séquences.

Nous pouvons illustrer la commande `cuts` sur notre exemple simple (cf. 6.3.1). Considérons que l'observation que nous souhaitons atteindre est $o = \{o_2, o_3\}$. La formule booléenne décrivant cette observation est $\phi = (o_2 \& o_3 \& not\ o_1 \& not\ o_4)$. Plus exactement, dans le cadre de notre modèle ALTARICA tel que décrit en figure 7.2, la formule est :

```
O.o2 & O.o3 & not O.o1 & not O.o4
```

On peut donc obtenir l'ensemble des coupes pour cette formule en utilisant ARC :

```
arc>cuts --visible-tags=incont SUD "O.o2 & O.o3 & not O.o1 & not O.o4"
N1 := 1;
N6 := ('S.e1'? -N1 : N1);
N5 := ('S.e6'? -N1 : N6);
N4 := ('S.e7'? -N1 : N5);
N3 := ('S.e5'? -N1 : N4);
N2 := ('S.e4'? -N1 : N3);
N0 := ('S.e3'? -N2 : N1);
root := -N0;
```

Les coupes peuvent aussi être énumérées :

```
arc>cuts --visible-tags=incont --enum SUD "O.o2 & O.o3 & not O.o1 & not O.o4"
(-S.e1, -S.e6, -S.e7, -S.e5, -S.e4, S.e3)
```

L'événement e_2 n'apparaissant pas, cela signifie qu'il peut être ou ne pas être arrivé. Ces résultats correspondent donc à l'ensemble de coupes $\{\{e_3\}, \{e_2, e_3\}\}$.

Si on ajoute l'option `min` à la dernière commande, nous obtenons les coupes minimales. Le résultat donné par ARC est *`(S.e3)`*, ce qui correspond à l'ensemble composé de l'unique coupe minimale $\{e_3\}$.

Ces résultats correspondent bien à ceux donnés dans la définition de l'exemple d'illustration, en table 3.4.

7.2.3.4 Intégration de métriques

Un des derniers travaux d'intégration industrielle effectué a concerné les métriques de performances de diagnostic. Nous avons intégré ces définitions au travail du groupe d'études. Le contexte industriel dans lequel s'applique les métriques a un périmètre plus petit que celui dans lequel elles sont été définies. Nous donnons ici quelques unes des ces caractéristiques.

La première chose est que les systèmes avioniques que nous considérons sont déterministes. Leur représentation l'est aussi. Dans nos modèles, il n'existe qu'une seule transition sortant d'une même configuration et étant étiquetée par un même événement.

Une deuxième chose est qu'aujourd'hui, les analyses sont bornées aux simples pannes. Cela signifie que nous ne considérons en pratique les effets d'un événement que par son occurrence depuis l'état initial. Comme nous l'avons déjà exprimé dans ce document, de telles analyses sont considérées comme étant pertinentes du fait la probabilité moindre des pannes multiples.

Une troisième caractéristique est que nos modèles des systèmes que nous considérons ne contiennent que des événements correspondant à des défaillances. Ainsi, il n'y a pas eu besoin pour intégrer ces métriques de différencier des états stables et des états non stables.

Dans ce contexte, nous comprenons que chaque défaillance a un seul et unique effet puisque le modèle est déterministe et puisque nous ne considérons que les pannes simples. Cet effet représente donc la signature de l'événement.

Les notions de détectabilité et de diagnosticabilité ont facilement été adaptées à ce contexte. Plus exactement, les définitions et métriques ont été créées pour être générales mais avec le soucis qu'elles soient compatibles avec le milieu industriel dans lequel s'inscrivait la thèse.

Très simplement donc, les métriques de détectabilité et de diagnosticabilité ont été appliquées au contexte industriel.

7.2.4 Dimensionnement d'un exemple de système avionique

Nous avons appliqué la solution décrite tout au long de cette section 7.2 à deux modèles avioniques que nous appellerons *`Modele-light`* et *`Modele-full`*. `Modele-full` est la modélisation d'une suite avionique d'un avion régional dans la catégorie 75 à 95 sièges. `Modele-light` est la modélisation d'un sous-système représentatif de la suite avionique, au sens où il contient l'ensemble des types de monitoring, de procédés de communication et de défaillances du système complet.

Nous présentons dans cette partie quelques chiffres dimensionnant ces expérimentations.

7.2.4.1 Système

Pour ces expérimentations, nous ne considérons donc que le périmètre utile à la maintenance de la suite avionique. Nous donnons une idée de la taille des systèmes considérés à travers la table 7.1.

7.2.4.2 Génération automatique de modèle

Nous pouvons nous intéresser à la table 7.2 qui dimensionne la première étape de génération : l'acquisition des données à partir de la description d'entrée de l'ingénierie système et la génération de modèle AltaRica. Nous rappelons que la description en entrée peut regrouper différents types

Système	# composants physiques	# liens physiques	# flux d'échange
Système `light`	22	114	404
Système `full`	295	2881	6940

TABLE 7.1 – Taille des systèmes avioniques

de documents, écrits par différentes personnes et pour différentes utilisations. Le nombre caractérisant la description du système en entrée représente le nombre total de lignes lues des différents documents d'entrée.

Système	Description d'entrée	Code AltaRica généré
Système `light`	4859	1736
Système `full`	76898	35812

TABLE 7.2 – Statistiques pour les fichiers de génération (en nombre de lignes)

Nous pouvons ensuite donner une idée du temps que prend le processus de modélisation pour ces systèmes à travers la table 7.3. Nous pouvons remarquer que le temps du processus de génération correspond en fait au temps d'acquisition des données d'entrée.

Système	Acquisition des données d'entrée (1)	Génération de code (2)	Temps total de génération de modèle (1+2)
Système `light`	32 sec.	1 sec.	33 sec.
Système `full`	11 min.	2 sec.	11 min 2 sec.

TABLE 7.3 – Statistiques du processus de génération de modèle

Nous pouvons finalement nous intéresser aux résultats de la génération de modèle avec la table 7.4.

Modèle	# variables d'état	# variables de flux	# événements	# variables d'observation	# dépendances fonctionnelles
`Modele-light`	109	282	146	101	282
`Modele-full`	1348	7713	2268	2388	7693

TABLE 7.4 – Taille des modèles générés

Avec de telles caractéristiques, nous réalisons rapidement qu'analyser le modèle en entier est impossible. En effet, étant donné que les variables du modèle sont au moins booléennes, le graphe d'accessibilité pour `Modele-light` peut compter au moins $2^{109+282}$ configurations. Pour `Modele-full` ce serait plutôt $2^{1348+7713}$ configurations.

Nous avons considéré un sous-modèle de `Modele-full`, obtenu par réduction sur une variable d'observation. Il compte 42 variables, toutes d'états, et 73 événements. Le nombre de configurations pour ce sous-modèle est d'environ $6.67 * 10^{17}$ et le nombre de transitions d'environ $1.79 * 10^{19}$. On comprend donc aisément que le calcul du graphe de comportement du modèle global explose en mémoire.

7.2.4.3 Génération automatique de règles de diagnostic

Pour la dernière phase de génération du processus, il est intéressant de regarder les caractéristiques à propos de la taille des modèles calculés durant la phase de réduction. Les figures 7.13 et 7.14 montrent respectivement la proportion d'événements et de variables dans les modèles réduits pour `Modele-full`.

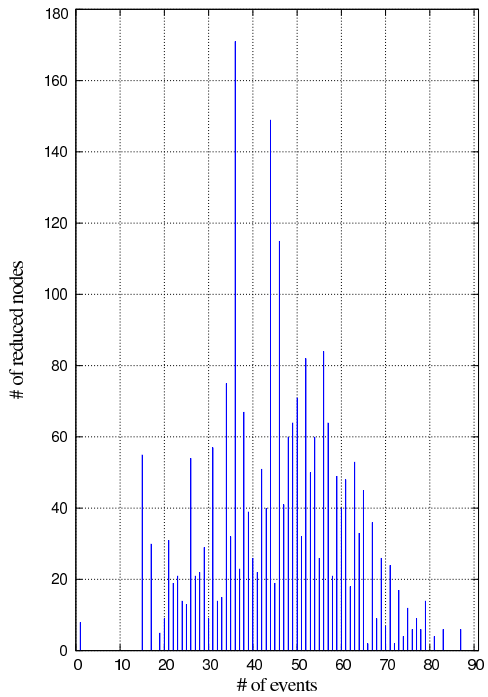


FIGURE 7.13 – Événements dans les modèles réduits

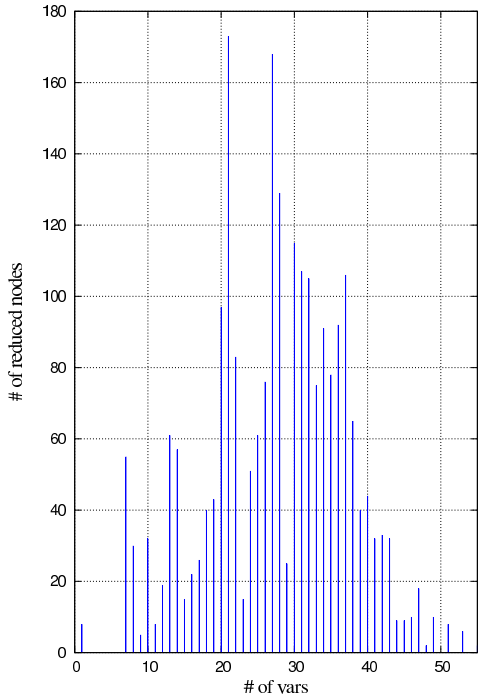


FIGURE 7.14 – Variables dans les modèles réduits

Nous pouvons voir par exemple que le plus gros modèle réduit pour `Modele-full` n'excède pas 90 événements et 55 variables alors que le modèle global comporte 2268 événements et 9061 variables.

7.3 Bilan et exploitations à venir

Dans ce chapitre nous avons présenté le groupe d'études à THALES AVIONICS dans lequel se sont déroulés les travaux de thèse. Nous avons montré comment les travaux de thèse ont interagi et contribué aux travaux du groupe d'études sur le diagnostic pour la maintenance.

La dimension industrielle était primordiale dans cette thèse en entreprise et tous les choix ont été faits avec en tête leur application dans le contexte industriel des travaux. Plus encore, nous avons travaillé à une solution qui ne soit pas divergente des techniques actuellement utilisées dans les produits.

Avec la solution que nous avons présentée dans la section 7.2, nous avons eu la volonté forte de permettre une intégration à court terme dans les produits, avec un gain fort. Avec la génération automatique de modèle puis de règles de diagnostic, nous prenons en compte et traitons les problèmes amenés par la complexification des systèmes.

Tous les résultats de thèse n'ont pas encore été intégrés dans les travaux du groupe d'études.

Le but est, à la suite de cette thèse, de continuer à les intégrer. Ainsi nous pouvons envisager des travaux qui seront mis en place dans le futur, parmi lesquels :

- Finir d'intégrer les résultats à travers l'exploitation des règles de diagnostic pour les métriques.
- Intégrer les métriques spécifiques au contexte industriel qui utilisent les notions de taux de défaillances ainsi que la structure particulière des systèmes avioniques.
- Comparer différents systèmes industriels avec les techniques présentés dans ce rapport.
- Calculer l'objet "dernier événement" (cf. 4.4.2.5) qui est suffisant pour l'application des métriques telles que nous les avons définies.
- Travailler sur une profondeur de systèmes supérieure à 1 dans les métriques de systèmes avioniques.
- Trouver des méthodes exploitant les particularités des systèmes avioniques, telles que nous l'avons fait avec la méthode de réduction, pour augmenter la taille des systèmes sur lesquels il est possible de calculer les coupes minimales à partir d'une observation complète, et non seulement d'une seule variable d'observation.

Comme nous le voyons, non seulement les travaux de thèse ont pu être intégrés dans les travaux industriels, cela contribuant à les faire avancer et à bâtir une solution de chaîne de diagnostic, mais de plus les perspectives d'intégration du reste des travaux sont nombreuses et intéressantes.

Conclusion

Travaux sur la diagnosticabilité

Durant les quinze dernières années, les activités de recherche sur le diagnostic et le pronostic, et parmi elle l'étude de la diagnosticabilité, se sont beaucoup développées. Les motivations industrielles que nous avons exposées dans ce document ne sont pas étrangères à cela. Dans sa thèse [Puc08], Xavier Pucel met d'ailleurs en avant les principales interprétations de la diagnosticabilité.

La notion de diagnosticabilité s'est vue précisée au fur et à mesure des années et des travaux menés. Les premiers travaux ont commencé par définir la diagnosticabilité comme la capacité du système à isoler toutes les défaillances possibles. Dans cette définition, la notion de diagnosticabilité s'applique au système global qui est dit diagnosticable lorsque tous les événements génèrent des observations différentes, et non diagnosticable sinon. Cela correspond à une définition d'isolabilité d'un système à laquelle nous avons fait référence en début de section 4.2.1. On trouve par exemple ce type de diagnosticabilité dans [SSL+95b, CPR+00b, CPR+00a, JHCK01, YL02].

Une deuxième version de diagnosticabilité s'intéresse cette fois-ci non plus seulement au système mais aux défaillances, fautes ou événements selon le cas. Dans cette version, la diagnosticabilité est plus précise et désigne la capacité à isoler un événement. Ainsi, un événement est dit diagnosticable si les observations qu'ils génèrent sont uniques. Une nouvelle fois, cette notion se rapproche de celle d'isolabilité d'événements, ou encore de diagnosticabilité forte telle que nous l'avons décrite en définition 24. On trouve par exemple ce type de description de la diagnosticabilité dans [Pen04, Sch07, SH08, Rib09, HNLT10, Bat11].

Enfin, plus récemment, on trouve une notion de diagnosticabilité plus précise. En effet, on trouve des travaux qui abordent des notions se rapprochant de la notion d'ambiguité d'événements que nous avons introduite en définition 28. On ne catégorise ainsi plus seulement les événements comme diagnosticables ou non diagnosticables, mais on s'intéresse à regarder à quel point ils sont diagnosticables les uns par rapport aux autres. Des travaux de ce type sont exposés dans [TMEO06, Puc08, Yas08]. Dans ces travaux, la diagnosticabilité est basée sur la notion de *degré de diagnosticabilité* du système elle-même basée sur le nombre de *D-classes* dans le système. Les D-classes regroupent les défaillances non discriminables (deux défaillances non discriminables sont dans le même groupe) et le degré de diagnosticabilité du système est alors le quotient du nombre de D-classes sur le nombre total de défaillances. Le choix que nous avons fait de catégoriser les événements par le fait qu'ils ne sont pas fortement diagnosticables (voir la définition du groupe d'ambiguité, déf. 28) nous permet de donner un sens particulier à la notion d'ambiguité qui représente alors le nombre maximal de tentatives nécessaire pour diagnostiquer un événement (cf. définitions 29, 30 et 31 pages 100–101). Le degré de diagnosticabilité moyen (métrique 10 page 102) qui caractérise la diagnosticabilité du système représente alors en moyenne le nombre de tentatives maximum nécessaires pour isoler un événement.

Bilan

Dans ce document, nous avons présenté les travaux de thèse effectués sur le monitoring et les performances de diagnostic des systèmes avioniques.

Nous avons commencé par présenter le contexte de ce travail, à savoir le domaine de l'avionique,

et plus particulièrement, le diagnostic pour la maintenance avionique. Nous avons ensuite donné une vue d'ensemble des trois principales approches pour faire du diagnostic : le raisonnement à base de règles, le raisonnement à base de cas et le raisonnement à base de modèles. Cela nous a permis de remarquer que l'approche la plus satisfaisante pour notre problème et dans notre contexte est celle à base de modèles.

Dans la deuxième partie, nous avons commencé par formaliser la notion de système à événements discrets représentant notre modèle de système. Les systèmes considérés sont déterministes ou non déterministes et peuvent contenir tout type d'événement. Nous avons défini la notion de système sous diagnostic qui caractérise un système associé à un observateur et à une stratégie d'observation. Cette définition du système sous diagnostic nous a permis de démontrer les relations fortes qui existent entre un système et sa version sous diagnostic. Nous nous sommes intéressés aux performances de diagnostic de ces systèmes à travers les notions de détectabilité (les événements sont-ils visibles lorsqu'ils sont déclenchés ?) et de diagnosticabilité (les événements sont-ils localisables lorsqu'ils arrivent ?). Nous avons ensuite continué le chapitre concernant les performances de diagnostic en particularisant les définitions et métriques introduites au contexte avionique, et nous l'avons terminé en proposant une analyse des performances de diagnostic à partir d'objets (séquences, coupes et leurs versions minimales, etc.) obtenus depuis le système. En nous appuyant sur la définition des performances de diagnostic, nous avons ensuite pu définir des relations de comparaison de systèmes sous diagnostic selon leurs performances, et à travers cela, nous avons pu comparer différents monitorings pour un même système, cela permettant d'obtenir par exemple un ensemble de capteurs minimal conservant les performances de diagnostic.

Dans la dernière partie nous avons commencé par montrer comment se traduit l'utilisation des performances de diagnostic en pratique. Nous avons présenté le langage AltaRica et ses outils puis montré et illustré leur utilisation pour à la fois modéliser les systèmes sous diagnostic (directement avec le langage AltaRica) et mesurer les performances de diagnostic (avec l'outil ARC). Dans un deuxième chapitre, nous avons expliqué l'intégration des travaux dans le contexte industriel et avons ainsi décrit comment les travaux du groupe d'études dans lequel s'est déroulée la thèse ont évolué en interaction avec l'ensemble des travaux présentés dans ce rapport durant ces dernières années. Pour conclure cette partie nous avons donné une idée des suites envisagées de l'intégration des travaux de thèse dans le contexte industriel.

Perspectives

Les travaux que nous avons présentés dans ce document amènent d'autres idées de travaux. En plus des perspectives de travaux et d'intégration industriels que nous avons données dans la section 7.3, nous présentons ici quelques unes des perspectives de recherche possibles pour l'approfondissement de ce qui a été abordé dans cette thèse.

Nous pouvons commencer par une chose que nous avons mise en avant dans la section 4.4 traitant des relations de causes à effets que l'on peut obtenir à partir d'un système sous diagnostic. Nous avons vu que la relation définie par les séquences de défaillances est plus précise que la notion d'effets que nous avons considéré pour la définition des performances de diagnostic dans la partie II. Il serait intéressant d'un point de vue théorique ou en pratique dans le cas de systèmes relativement petits, d'adapter les définitions et métriques de performances de diagnostic à l'utilisation des séquences ; cela permettrait ainsi d'obtenir des résultats plus précis.

Ensuite, nous entrevoyons deux perspectives qui concernent la méthode de réduction de modèles AltaRica que nous avons présentée en section 7.2.3.2. La première idée est de généraliser la réduction de modèles et de ne plus l'appliquer seulement lors du calcul de coupes ou séquences, mais aussi lors de la vérification de n'importe quelle formule. La deuxième idée ne découle pas directement de la réduction présentée mais en est inspirée. La réduction implémentée dans ARC permet de ne conserver que les informations nécessaires pour évaluer une formule cible. On peut alors imaginer réfléchir à une réduction qui ne conserverait que les informations nécessaires au calcul des effets d'un événement. Là où la première réduction regarde ce qui impacte une formule, la deuxième regarde ce qui est impacté par l'occurrence d'un événement (ou d'un ensemble d'événements). Avoir une telle réduction pourrait permettre d'élargir le périmètre de calculabilité de la

relation d'effets, tout comme la première élargit celui du calcul des coupes et séquences.

Finalement, il y a aussi une perspective qui est dans le prolongement direct de cette thèse qui concerne l'optimisation de monitoring. Définir une relation d'ordre sur les systèmes sous diagnostic comme nous l'avons fait permet d'ouvrir un champ de perspectives. L'optimisation du monitoring que nous avons abordée concerne la minimisation de l'ensemble des observations (et donc de l'ensemble des capteurs) tout en conservant un même niveau de performances de diagnostic. Il est possible, sur les mêmes bases, de s'intéresser par exemple au placement de capteurs. Dès lors que nous avons les informations décrivant l'ensemble des positions et rôles possibles de ces capteurs, il est possible de comparer différents monitoring et d'obtenir le meilleur (avec les relations d'ordre que nous avons définies). La solution de monitoring proposée peut être ainsi non seulement optimisée vis-à-vis l'ensemble des capteurs à considérer, mais aussi vis-à-vis de leur placement. Et plus intéressant encore, en changeant le placement des capteurs, il est possible d'améliorer les performances de diagnostic.

Bibliographie

[Aam89] A. Aamodt. Towards robust expert systems that learn from experience – an architectural framework, 1989. EKAW–89 ; Third European Knowledge Acquisition for Knowledge–Based Systems Workshop , Paris, July 1989. pp 311–326.

[Aam91] A. Aamodt. A knowledge–intensive approach to problem solving and sustained learning, 1991. Ph.D. dissertation, University of Trondheim, Norwegian Institute of Technology, May 1991. (University Microfilms PUB 92–08460).

[AB01] UpTime Solutions AB. Rodon - tolerance based simulation & automated fmea generation in aircraft elevators, 2001. 4th Annual Systems Engineering Conference, Dallas, October 22-25, 2001.

[AB09] UpTime Solutions AB. Rodon - general overview, 2009. RODON Documentation.

[Ade11] R. Adeline. *Méthodes pour la validation de modèles formels pour la Sûreté de Fonctionnement et extension aux problèmes multi-physique.* Thèse de doctorat, Université de Toulouse, March 2011.

[AGPR00] A. Arnold, A. Griffault, G. Point, and A. Rauzy. The altarica formalism for describing concurrent systems. *Fundamenta Informaticae*, 40 :109–124, 2000.

[Agr96] G. Agre. Diagnostic bayesian networks, 1996. Computers and Artificial Intelligence, Vol. 16, No. 1, 1.

[Alt89] K.D. Althoff. Knowledge acquisition in the domain of cnc machine centers ; the moltke approach, 1989. Third European Workshop on Knowledge–Based Systems, Paris, July 1989. pp 180–195.

[AS07] D. Andersson and P. Sköld. Evaluation of a diagnostic tool for use during system development and operations, 2007. Master Thesis - Department of Electrical Engineering, Linköping University.

[Ash91] K. Ashley. Modeling legal arguments : Reasoning with cases and hypotheticals, 1991. MIT Press, Bradford Books, Cambridg.

[AW91] K.D. Althoff and S. Wess. Case-based knowledge acquisition, learning and problem solving in diagnostic real world tasks, 1991. Proc. EKAW-91, Glasgow & Crieff.

[Bar89] R. Bareiss. Exemplar–based knowledge acquisition : A unified approach to concept representation, classification and learning, 1989. Boston, Academic Press.

[Bat11] M. Batteux. *Diagnosticabilité et diagnostic de systèmes technologiques pilotés : développement d'une chaîne de conception outillée d'un système de diagnostic appliquée aux systèmes technologiques pilotés.* PhD thesis, Université Paris Sud-Paris XI, 2011.

[BCD+07] P. Bieber, Ch. Castel, G. Durrieu, Ch. Seguin, C. Pagetti, and L. Sagaspe. Dependability modelling and assessment of avionics systems with altarica., 2007. 3èmes journées Altarica, Bordeaux, novembre 2008.

[BCK+02] P. Bieber, C. Castel, C. Kehren, C. Seguin, C. Bougnol, and J.P. Heckman. Analyse d'un système hydraulique avion avec altarica, 2002. First Altarica Workshop, Toulouse, octobre 2002.

[BCL+04] G. Biswas, M.-O. Cordier, J. Lunze, L. Trave-Massuyes, and M. Staroswiecki. Diagnosis of complex systems : Bridging the methodologies of the fdi and dx communities, 2004. IEEE Transactions on systems, Man and Cybernetics, Vol. 34, No. 5, October 2004.

[BDRS06] M. Boiteau, Y. Dutuit, A. Rauzy, and J. P. Signoret. The AltaRica data-flow language in use : modeling of production availability of a multi-state system. *Reliability Engineering & System Safety*, 91(7), 2006.

[BdSB84] H.W. Block and W. de Souza Borges. Comparing coherent systems. *Lecture Notes-Monograph Series*, pages 187–192, 1984.

[Bea71] R. V. Beard. Failure accommodation in linear systems through self-reorganization, 1971. Dept. MVT-71-1, Man Vehicle Laboratory, Cambridge, MA.

[Ber09] R. Bernard. *Analyse de Sécurité multi-systèmes.* Thèse de doctorat, Université de Bordeaux, November 2009.

[BFHJ02] A. Benveniste, E. Fabre, S. Haar, and C. Jard. Diagnosis of asynchronous discrete event systems, a net unfolding approach, 2002. Publication Interne IRISA, No 1456.

[BIFM09] P. Bunus, O. Isaksson, B. Frey, and B. Münker. Model-based diagnostics techniques for avionics applications with rodon, 2009. AST 2009, March 26-27, Hamburg, Germany.

[BKJS06] M. Blanke, M. Kinnaert, J., and M. Staroswiecki. Structural analysis, 2006. Diagnosis and Fault-Tolerant Control, chapter 5, pp. 109-188.

[BKLS07] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki. Structural analysis, 2007. Diagnosis and Fault-Tolerant Control, chapter 4, pp. 69-108.

[BR94] S. Brlek and A. Rauzy. Synchronization of constrained transition systems. In *ed. H. Hong, Proc. of the First International Symposium on Parallel Symbolic Computation – PASCO'94*, World Scientific Publishing, pages 54–62, 1994.

[Bra91] K. Branting. Exploiting the complementarity of rules and precedents with reciprocity and fairness, 1991. Proceedings from the Case–Based Reasoning Workshop 1991, Washington DC. Sponsored by DARPA. Morgan Kaufmann, pp 39–50.

[Bro76] A. L. Brown. Quantitative knowledge, causal reasoning and the localization of failures, 1976. Technical Report 362, MIT Artificial Intelligence Laboratory.

[Bro82] A. L. Brown. Pedagogical, natural language and knowledge engineering techniques in sophie i, ii and iii, 1982. Intelligent Tuoring Systems, Academic Press NY, pp. 227-282.

[BS84] B. G. Buchanan and E. H. Shortliffe. Rule-based expert systems : The mycin experiments of the stanford heuristic programming project, 1984. Addison-Wesley, Reading, MA.

[CB05] S. Bumbaru C.D. Bocaniala, J.S. da Costa. On the applicability of state-of-the-art fault diagnosis methodologies to simple and complex systems, 2005. The annals of "dunarea de jos" university of galati.

[CCC93] S. Cheon, S. Chang, and H. Chung. Development strategies of an expert system for multiple alarm processing and diagnosis in nuclear-power-plants, 1993. IEEE Transactions on Nuclear Science, 40(1) :21-30.

[CCYB95] C. Fernandez Chamizo, P. A. Gonzalez Calero, L. Hernandez Yariez, and A. Urech Bagué. Case-based retrieval of software components, 1995. Expert Systems with Applications, Vol. 9, No. 3, pp. 397-421.

[CDL+00a] M.-O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. Ai and automatic control approaches of model-based diagnosis : Links and underlying hypotheses, 2000. Proc. 4th IFAC Symp. Fault Detection, Supervision Safety Technical Processes SAFEPROCESS, Budapest, Hungary, pp. 274–279.

[CDL+00b] M.-O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. A comparative analysis of ai and control theory approaches to model-based diagnosis, 2000. Proc. 14th European Conf. Artificial Intelligence ECAI, Berlin, Germany, pp. 136–140.

[CDL+04] M.-O. Cordier, P. Dague, F. Lévy, J. Montmain, M. Staroswiecki, and L. Travé-Massuyès. Conflicts versus analytical redundancy relations : A comparative analysis of the model based diagnosis approach from the rtificial intelligence and automatic control perspectives, 2004. IEEE transactions on systems, man and cybernetics—part b : cybernetics, vol. 34, no. 5, october 2004.

[CePT89a] L. Console and D. Theseider Dupré et P. Torasso. Abductive reasoning through direct deduction from completed domain models, 1989. Methodologies for Intelligent System 4, . pp 175-182. North Holland.

[CePT89b] L. Console and D. Theseider Dupré et P. Torasso. A theory of diagnosis for incomplete causal models, 1989. Proc. 11th IJCAI, Detroit, pp.1311-1317.

[CFW75] R. N. Clark, D. C. Fosth, and V. M. Walton. Detection instrument malfunctions in control systems, 1975. EEE Trans. Aerospace Electron. Syst., AES-II, 465-473.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In E. Allen Emerson and A. Prasad Sistla, editors, *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pages 154–169. Springer, 2000.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. MIT Press (Cambridge, MA), 1999.

[Che08] L. Cheng. Latest progress of research on fault diagnosis based on information fusion, 2008. Information Technology Journal.

[Chu12] F. Chucri. *Exploiter la Structure des Modèles pour la Vérification par la Méthode CEGAR*. PhD thesis, Université de Bordeaux, Decembre 2012.

[CKPR72] A. Colmerauer, H. Kanoui, R. Pasero, and P. Roussel. Un systeme de communication en français. *Rapport préliminaire de fin de contrat IRIA, Groupe Intelligence Artificielle, Faculté des Sciences de Luminy, Université d'Aix-Marseille II*, 1972.

[CM93] O. Coudert and J. C. Madre. Fault tree analysis : 10^{20} prime implicants and beyond. In *Proceedings of Reliability and Maintainability Symposium*, pages 240–245, 1993.

[Con94] L. Console. Introduction, 1994. Annals of Mathematics and Artificial Intelligence 11, pp. 1-10.

[CP86] P. T. Cox and T. Piertzykowski. Causes for events : their computation and application, 1986. Proc. 8th CADE.

[CPR+00a] L. Console, C. Picardi, M. Ribaudo, et al. Diagnosis and diagnosability analysis using pepa. In *ECAI*, pages 131–135, 2000.

[CPR+00b] L. Console, C. Picardi, M. Ribaudo, et al. Diagnosis and diagnosability analysis using process algebras. In *Proceedings of the Eleventh International Workshop on Principles of Diagnosis (DX-00), MX, Morelia, Mexico*, pages 25–32, 2000.

[CPVG05] R. Ceballos, S. Pozo, C. Del Valle, and R. M. Gasca. An integration of fdi and dx techniques for determining the minimal diagnosis in an automatic way, 2005. Lecture notes in computer science.

[CR97] M.-M. Corsini and A. Rauzy. Toupie : The μ-calculus over finite domains as a constraint language. *J. Autom. Reasoning*, 19(2) :143–171, 1997.

[CT91] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis, 1991. Readings in Model-Based Diagnosis, Morgan Kaufmann, pp. 78-88.

[Dar09] A. Darwiche. *Modeling and reasoning with Bayesian networks*, volume 1. Cambridge University Press, 2009.

[Dav84] R. Davis. Diagnostic reasoning based on structure and behavior, 1984. Artificial Intelligence 24(1-3) :347-410.

[DDDW77] J. C. Deckert, M. N. Desai, J. J. Deyst, and A. S. Willsky. F-8 dfbw sensor failure identification using analytic redundancy, 1977. IEEE Transactions on Automatic Control AC-22, 795-803.

[DH88] R. Davis and W. Hamscher. Model-based resoning : Troubleshooting, 1988. Readings in Model-Based Diagnosis, Morgan Kaufmann, pp. 3-24.

[Dic86] A Dicky. An algebraic and algorithmic method for analysing transition systems. *Theor. Comput. Sci.*, 46(2-3) :285–303, October 1986.

[dK76] J. de Kleer. Local methods for localizing faults in electroniuc circuits, 1976. Out of print - MIT Artificial Intelligence Laboratory.

[dKW87] J. de Kleer and B. C. Williams. Diagnosing multiple faults, 1987. Artificial Intelligence 32, 97-130.

[dKW89] J. de Kleer and B. C. Williams. Diagnosing with behavioral modes, 1989. Proc. 11th IJCAI, Detroit, pp. 1324-1330.

[DL94] A.-M. Dery and W. Lejouad. From modular to distributed architectures : Centaur and smeci experiments, 1994. Rapport de recherche INRIA Numéro 2398.

[DP02] P. Darfeuil and G. Pelloquin. Expérimentation d'ocas chez turboméca, 2002. First Altarica Workshop, Toulouse, octobre 2002.

[DSH+82] R. Davis, H. Shrobe, W. Hamscher, M. Shirley, and S. Polit. Diagnosis based on structure and function, 1982. Proc. Of AAAI-82, Pittsburgh, PA, pp. 137-142, August 1982.

[FHMM82] M. B. First, B. J. Heimer, S. McLinden, and R. A. Miller. Computer-assisted localization of peripheral nervous system lesions, 1982. Computer and Biomedical Research 15(6) :525-543, December 1982.

[Fir93] 1993. First European Workshop on Case–based Reasoning, Posters and Presentations, 1–5 November 1993. Vol. I–II. University of Kaiserslautern.

[For96] K. D. Forbus. Qualitative reasoning, 1996. DRAFT - Chapter for the CRC Handbook of Computer Science.

[Fra90] P. M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy - a survey and some new results, 1990. Automatica, Vol. 26, No. 3, pp. 459-474.

[Gen83] D. Gentner. Structure mapping – a theoretical framework for analogy, 1983. Cognitive Science, Vol.7. s.155–170.

[Gen84] M. R. Genesereth. The use of design descriptions in automated diagnosis, 1984. Artificial Intelligence 24(1-3) :411-436.

[GPKV11] A. Griffault, G. Point, F. Kuntz, and A. Vincent. Symbolic computation of minimal cuts for AltaRica models. Technical Report RR-1456-11, INRIA, September 2011. 65 pages.

[GPV07] A. Griffault, G. Point, and A. Vincent. Altarica au labri : Travaux récents et projets, 2007. 3èmes journées Altarica, Bordeaux, novembre 2007.

[GR02] A. Griffault and A. Rauzy. Altarica : Passé, présent, futur, 2002. First Altarica Workshop, Toulouse, octobre 2002.

[GV04] A. Griffault and A. Vincent. The mec 5 model-checker. In *Proceedings of the 16th International Conference on Computed Aided Verification (CAV 2004), Boston, MA, USA*, volume 3114 of *Lectures Notes in Computer Science*, pages 488–491. Springer Verlag, July 2004.

[Ham86] K. J. Hammond. Chef : A model of case-based planning, 1986. roc. American Association for Artificial Intelligence, AAAI-86, August 1986. Philadelphia, PA, US.

[Ham91] W. Hamscher. Modeling digital circuits for troubleshooting, 1991. Artificial Intelligence 51(1-3) :223-271.

[HBZ92] G. Holzer, W. Bertsch, and Q. Zhang. Design criteria of a gaschromatography mass-spectrometry based expert system for arson analysis, 1992. Analytica Chimica Acta, 259(2) :225-35.

[HCdK92] W. Hamscher, L. Console, and J. de Kleer. Readings in model-based diagnosis, 1992. Book - Morgan Kaufmann Publisher, San Mateo, CA.

[HGG06] L. Hélouët, T. Gazagnaire, and B. Genest. Diagnosis from scenarios, 2006. Proc. of the 8th Int. Workshop on Discrete Events Systems, WODES'06 - pages 307-312.

[HHNT86] J. Holland, K. Holyoak, R. Nisbett, and P. Thagard. Induction : Processes of inference, learning and discovery, 1986. MIT Press, Cambridge, MA.

[Hin92] T. R. Hinrichs. Problem solving in open worlds, 1992. Lawrence Erlbaum Associates.

[HNLT10] K.M. Hangos, E. Németh, R. Lakner, and A. Toth. Detectability and diagnosability of faults in lumped process systems. In *20th European Symposium on Computer Aided Process Engineering-ESCAPE20*, pages 1447–1452, 2010.

[Hum08] S. Humbert. *Déclinaison d'exigences de sécurité, du niveau système vers le niveau logiciel, assistée par des modèles formels.* Thèse de doctorat, Université de Bordeaux I, April 2008.

[IEE92] 1992. IEEE Expert 7(5), Special issue on case–based reasoning. October 1992.

[Isa09] O. Isaksson. Model-based diagnosis of a satellite electrical power system with rodon, 2009. Thesis - Linköpings University.

[Jen96] F.V. Jensen. *An introduction to Bayesian networks*, volume 74. UCL press London, 1996.

[JHCK01] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *Automatic Control, IEEE Transactions on*, 46(8) :1318–1321, 2001.

[Jon73] H. L. Jones. Failure detection in linear systems, 1973. Thesis - MIT, Cambridge, MA.

[Jos98] C. Jost. Comparaison qualitative et quantitative de modèles proie-prédateur à des données chronologiques en écologie, 1998. Thèse - Institut national agronomique Paris-Grignon.

[Jr.93] C. M. Higgins Jr. Classification and approximation with rule-based networks, 1993. Thesis - Department of Electrical Engineering, California Institute of Technology, Pasadena, California.

[Keh05] C. Kehren. *Motifs formels d'architectures de systèmes pour la sûreté de fonctionnement.* Thèse de doctorat, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace (SUPAERO), 2005.

[KGS+11] Fabien Kuntz, Stéphanie Gaudan, Christian Sannino, Éric Laurent, Alain Griffault, and Gérald Point. Model-based diagnosis for avionics systems using minimal cuts. In M. Sachenbacher, O. Dressler, and M. Hofbaur, editors, *22nd International Workshop on Principles of Diagnosis*, pages 138–145, Murnau, Allemagne, 2011.

[Khu08] M.T. Khuu. *Contribution à l'accélération de la simulation stochastique sur des modèles AltaRica Data Flow.* Thèse de doctorat, Université de la Méditerranée (Aix-Marseille II), 2008.

[Kit80] M. Kitamura. Detection of sensor failures in nuclear plant using analytic redundancy, 1980. Trans. Am. Nucl. Soc., 34, 581-583.

[Kit93] H. Kitano. Challenges for massive parallelism, 1993. Proc. 13th. Int. Conference on Artificial Intelligence, IJCAI-93 : pp813-34.

[KMS99] S. Kochar, H. Mukerjee, and F.J. Samaniego. The "signature" of a coherent system and its application to comparisons among systems. *Naval Research Logistics (NRL)*, 46(5) :507–523, 1999.

[Kol83] J. Kolodner. Maintaining organization in a dynamic long–term memory, 1983. Cognitive Science, Vol.7, s.243–280.

[Kot89] P. Koton. Using experience in learning and problem solving, 1989. Massachusetts Institute of Technology, Laboratory of Computer Science (Ph.D. diss, October 1988). MIT/LCS/TR–441.

[KPPH00] T. Jie Ying Kirubarajan, K. R. Pattipati, and A. Patterson-Hine. A hidden markov model-based algorithm for fault diagnosis withpartial and imperfect tests, 2000. IEEE Transactions on Systems, Man and Cybernetics, Part C : Applications and Reviews.

[LA97] S. Leonhardt and M. Ayoubi. Methods of fault diagnosis, 1997. Control Eng. Practice, Vol. 5, No. 5, pp. 683-692.

[LBFL80] R. Lindsay, B. Buchanan, E. Feigenbaum, and J. Lederberg. Applications of artificial intelligence for organic chemistry : The dendral project, 1980. McGraw-Hill, New York, NY.

[LK89] S. T. Liu and G. E. Kelly. Rule-based diagnostic method for hvac fault detection, 1989. Proceedings of IBPSA Building Simulation 89, 1989, pp. 319–324.

[Lun00] K. Lunde. Object-oriented modeling in model-based diagnosis, 2000. Modelica Workshop 2000 Proceedings.

[LW07] J. Lundkvist and S. Wahnström. Dynamic model based diagnosis for combustion engines in rodon, 2007. Master Thesis performed at Vehicular Systems at Linköpings University and at Sörman Information & Media AB.

[Mat05] J. Mattioli. Smmart - state of the art report, 2005. Project SMMART : System for Mobile Maintenance Accessible in Real Time.

[MC74] R. C. Montgomery and A. K. Caglayan. A self-reorganizing digital flight control system for aircraft, 1974. AIAA 12th Aerospace Sciences Meeting, Washington, D.C., 3(I Jan.-I Feb. 1974.

[McD80] J. McDermott. R1 : An expert in the computer systems domain, 1980. Proceedings of AAAI 1, pages 269-271.

[MMZ08] K. Medjaher, A. Mechraoui, and N. Zerhouni. Diagnostic et pronostic de défaillances par réseaux bayésiens, 2008. 4ème Journées Francophones sur les Réseaux Bayésiens, Lyon, 29 - 30 mai 2008.

[MNS+94] H. Matsumoto, M. Nomura, M. Shimoda, T. Saito, H. Yokoyama, K. Baba, J. Kawakami, Y. Katayama, A. Kaji, and S. Nigawara. Neural network state diagnostic system for equipment, 1994. US Patent number 5333240 - Assignee : Hitachi, Ltd., Tokyo, Japan.

[Moh97] F. Mohamed. Approche à base de logique floue pour le test et le diagnostic des circuits analogiques, 1997. Thèse - Institut National Polytechnique de Grenoble.

[Moz92] I. Mozetic. Model-based diagnosis : An overview, 1992. Advanced Topics in Artificial Intelligence ; pp. 419-430.

[MP71] R. K. Mehra and I. Peshon. An innovations approach to fault detection and diagnosis in dynamic systems, 1971. Automatica, 7, 637-640.

[MS91] A. McNeile and N. Simons. Methods of behaviour modelling : A commentary on behaviour modelling techniques for mda, 1991.

[MWM91] M. Mulholland, N. Walker, and F. Maris. Expert system for repeatability testing of high-performance liquid- chromatographic methods, 1991. Journal of Chromatography, 550(1-2) :257-66.

[New68] A. Newell. On the analysis of human problem solving protocols, 1968. Carnegie Institute of Technology.

[NS65] A. Newell and H. Simon. An example of human chess play in the light of chess playing programs, 1965. Progress in Biocybernetics. Elsevier Press, Amsterdam.

[Oeh92] R. Oehlmann. Learning causal models by self–questioning and experimentation, 1992. AAAI–92 Workshop on Communicating Sientific and Technical Knowledge. American Association of Artificial Intelligence.

[Oxm93] R. E. Oxman. Case-based design support : Supporting architectural composition through precedent libraries, 1993. Journal of Architectural Planning Research.

[Pag04] C. Pagetti. *Extension temps réel du langage AltaRica.* Thèse de doctorat, École Centrale de Nantes et de l'Université de Nantes, 2004.

[PB86] B. Porter and R. Bareiss. Protos : An experiment in knowledge acquisition for heuristic classification tasks, 1986. Proceedings of the First International Meeting on Advances in Learning (IMAL), Les Arcs, France, pp. 159–174.

[PdM90] E. Plaza and R. López de Mántaras. A case–based apprentice that learns from fuzzy examples, 1990. Methodologies for Intelligent System 5, . pp 420–427. North Holland.

[Pen04] Y. Pencolé. Diagnosability analysis of distributed discrete event systems. In *ECAI*, volume 16, page 43, 2004.

[PG06] G. Point and A. Griffault. On the partial translation of Lustre programs into the AltaRica language and vice versa. RR-1415-06 RR-1415-06, November 2006.

[PGA87] D. Poole, R. Goebel, and R. Aleliunas. Theorist : A logical reasoning system for defaults and diagnosis, 1987. The Knowledge Frontier.

[Pit96] J. Pitrat. Implementation of a reflective system, 1996. Future Generation Computer Systems, 12, p.235-242.

[Poi00] G. Point. *AltaRica : Contribution à l'unification des méthodes formelles et de la sûreté de fonctionnement*. Thèse de doctorat, LaBRI – Université Bordeaux I, Janvier 2000.

[Poo89] D. Poole. Normality and faults in logic-based diagnosis, 1989. Proc. 11th IJCAI, Detroit, pp.1304-1310.

[Pos43] E. L. Post. Formal reductions of the general combinatorial decision problem, 1943. American Journal of Mathematics, 65 :197–268.

[PR99] G. Point and A. Rauzy. AltaRica : Constraint automata as a description language. *Journal Européen des Systèmes Automatisés*, 33(8–9) :1033–1052, 1999.

[Pro91] 1991. Proceedings from the Case–Based Reasoning Workshop, Washington D.C., May 8–10, 1991.

[PSS81] R. Patil, P. Szolovits, and W. Schwartz. Causal understanding of patient illness in medical diagnosis, 1981. Proc. of IJCAI-81, Vancouver, BC, pp. 893-899, August 1981.

[PTL+98] G. Point, P. Thomas, S. Lajeunesse, A. Griffault, A. Rauzy, and J.-P. Signoret. Le langage AltaRica. In *Actes du 11ième colloque national de Fiabilité et Maintenabilité, Lambda-Mu'11*, pages 119–125. SEE, 1998.

[Puc08] X. Pucel. *A unified point of view on diagnosability*. Thèse de doctorat, INSA de Toulouse, 2008.

[Rau01] A. Rauzy. Mathematical Foundation of Minimal Cutsets. *IEEE Transactions on Reliability*, 50(4) :389–396, december 2001.

[Rau02] A. Rauzy. Modes automata and their compilation into fault trees. *Reliability Engineering and System Safety*, 78 :1–12, 2002.

[RD97] A. Rauzy and Y. Dutuit. Exact and truncated computations of prime implicants of coherent and non-coherent fault trees within Aralia. *Reliability Engineering & System Safety*, 58(2) :127–144, November 1997.

[Rei87] R. Reiter. A theory of diagnosis form first principles, 1987. Artificial Intelligence 32(1) :57-96.

[Rib09] P. Ribot. *Vers l'intégration diagnostic/pronostic pour la maintenance des systèmes complexes*. PhD thesis, Université Paul Sabatier-Toulouse III, 2009.

[Ris83] E. Rissland. Examples in legal reasoning : Legal hypotheticals, 1983. Proceedings of the Eighth International Joint Conference on Artificial Intelligence, IJCAI, Karlsruhe.

[RL86] L. R. Rabiner and B. H. Luang. An introduction to hidden markov models, 1986. IEEE ASSP magazine january 1986.

[Rob03] S. Robin. Diagnostic à base de modèles - cours 3 : Modèles de panne et diagnostic abductif, 2003. Cours DEA 2003-2004 / Option DIAG - IRISA.

[RW87] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim.*, 25(1) :206–230, January 1987.

[RW91] A. M. Richter and S. Weiss. Similarity, uncertainty and case–based reasoning in patdex, 1991. Automated reasoning, essays in honour of Woody Bledsoe. Kluwer, pp. 249–265.

[SAB+73] E. Shortliffe, S. Axline, B. Buchanan, T. Merigan, and S. Cohen. An artificial intelligence program to advise physicians regarding antimicrobial therapy, 1973. Computers and Biomedical Research, 6 :544-560.

[Sab06] Z. Sabeh. Diagnostic à base de modèle : application à un moteur diesel suralimenté à injection directe, 2006. Thèse - Institut National Polytechnique de Lorraine.

[SAE10] SAE. Guidelines for Development of Civil Aircraft and Systems. Technical report, ARP 4754A, 2010.

[Sag08] L. Sagaspe. *Allocation sûre dans les systèmes aéronautiques : modèlisation, vérification et génération.* Thèse de doctorat, Université de Bordeaux, December 2008.

[Sch82] R. Schank. Dynamic memory ; a theory of reminding and learning in computers and people, 1982. Cambridge University Press.

[Sch02] R. Schoenig. Modélisation d'un système de contrôle-commande embarqué dans le cadre d'analyses fonctionnelle et dysfonctionnelle, 2002. First Altarica Workshop, Toulouse, octobre 2002.

[Sch04] K. Schlechta. *Coherent systems*, volume 2. Elsevier Science Limited, 2004.

[Sch07] A. Schumann. *Towards efficiently diagnosing large scale discrete-event systems.* PhD thesis, Citeseer, 2007.

[SD89] P. Struss and O. Dressler. Physical negation - integrating fault models into the general diagnostic engine, 1989. Proc. 11th IJCAI, Detroit, pp. 1318-1323.

[SH08] A. Schumann and J. Huang. A scalable jointree algorithm for diagnosability. In *23rd American National Conference on Artificial Intelligence (AAAI-08)*, 2008.

[Sim85] Robert L. Simpson. A computer model of case–based reasoning in problem solving : An investigation in the domain of dispute mediation, 1985. Technical Report GIT–ICS–85/18, Georgia Institute of Technology.

[SK05] J. W. Sheppard and M. A. Kaufman. Bayesian diagnosis and prognosis using instrument uncertainty, 2005. IEEE 2005 AUTOTESTCON Conference Record, Orlando, Florida.

[SM81] E. Smith and D. Medin. Categories and concepts, 1981. Harvard University Press.

[Sme94] P. Smets. What is dempster-shafer's model ?, 1994. Advances in the Dempster-Shafer theory of evidence, pp. 5-34, Wiley.

[Smy88] P. Smyth. The application of information theory to problems in decision tree design and rule-based expert systems, 1988. Thesis - California Institute of Technology, Pasadena, California.

[Sou09] H. Soueidan. *Discrete event modeling and analysis for systems biology models.* Thèse de doctorat, Université de Bordeaux, 2009.

[SR92] C. B. Skalak and E. Rissland. Arguments and cases : An inevitable twining, 1992. Artificial Intelligence and Law, An International Journal, 1(1), pp.3–48.

[SS88] S. Sharma and D. Sleeman. Refiner ; a case–based differential diagnosis aide for knowledge acquisition and knowledge refinement, 1988. EWSL 88 ; Proceedings of the Third European Working Session on Learning, Pitman, 1988. pp 201–210.

[SSB02] M. Schwabacher, J. Samuels, and L. Brownston. The nasa integrated vehicle health management technology experiment for x-37, 2002. SPIE AeroSense 2002.

[SSL+95a] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *Automatic Control, IEEE Transactions on*, 40(9) :1555–1575, 1995.

[SSL+95b] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *Automatic Control, IEEE Transactions on*, 40(9) :1555–1575, 1995.

[Sta00] M. Staroswiecki. Quantitative and qualitative models for fault detection and isolation, 2000. Mechanical Systems and Signal Processing (2000) 14(3), 301-325.

[Str97] P. Struss. Model-based and qualitative reasoning : An introduction, 1997. Annals of Mathematics and Artificial Intelligence 19, pp. 355–381.

[Syc87] E. P. Sycara. Resolving adversial conflicts : An approach to integrating case-based and analytic methods, 1987. Technical Report GIT-ICS-87/26, Georgia Institute of Technology, School of Information and Computer Science, Atlanta GA.

[Sä07] O. Särnholm. Structural algorithms in rodon - with a prototype implementation in java, 2007. Master's thesis performed at the Vehicular Systems division, department of Electrical Engineering at Linköping University.

[TMEO06] L. Travé-Massuyes, T. Escobet, and X. Olive. Diagnosability analysis based on component-supported analytical redundancy relations. *Systems, Man and Cybernetics, Part A : Systems and Humans, IEEE Transactions on*, 36(6) :1146–1160, 2006.

[TTE91] D. Thompson, T. Tomski, and S. Ellacott. An expert system for the preliminary design of timber roofs, 1991. Computers and Structures, 40(1) :115-25.

[Tul77] E. Tulving. Episodic and semantic memory, 1977. Organization of memory, Academic Press, pp. 381–403.

[vC86] M. van Caneghem. L'anatomie de prolog, 1986. Thèse, Inter'Editions, Paris.

[VER07] S. VERRON. Diagnostic et surveillance des processus complexes par reseaux bayesiens, 2007. Thèse - Ecole Doctorale d'Angers.

[Via85] M. Vialatte. Description et applications du moteur d'inférence snark, 1985. PhD thesis, Université Paris VI.

[Vin03a] A. Vincent. *Conception et réalisation d'un vérificateur de modèles AltaRica.* Thèse de doctorat, Université de Bordeaux, December 2003.

[Vin03b] A. Vincent. Vérification de modèles en altarica, 2003. 2ème journées Altarica, Marseille, octobre 2003.

[VK94] R. Valette and L. A. Künzle. Réseaux de petri pour la détection et le diagnostic, 1994. GDR Automatique - Journées sûreté, surveillance, supervision : détection et localisation des défaillances, novembre 1994.

[VKR93] S. Venkatamaran, R. Krishnan, and K. K. Rao. A rule-case based system for image analysis, 1993. Proc. 1st. European Workshop on Case-Based Reasoning, Posters & Presentations, 2 : pp.410-15.

[WA94] I. D. Watson and S. Abdullah. Developing case-based reasoning systems : A case study in diagnosing building defects, 1994. Proc. IEE Colloquium on Case-Based Reasoning : Prospects for Applications, Digest No : 1994/057, pp.1/1-1/3.

[Wil76] A. S. Willsky. A survey of design methods for failure detection in dynamic systems, 1976. Automatica, Volume 12, Issue 6, November 1976, Pages 601-611.

[Wit53] L. Wittgenstein. Philosophical investigations, 1953. Blackwell, pp. 31–34.

[WM94] I. Watson and F. Marir. Case-based reasoning : A review, 1994. The Knowledge Engineering Review, Vol. 9 No. 4 : pp. 355-381.

[Yas08] A.A. Yassine. *Génération des tests et placement de capteurs pour le diagnostic des systèmes physiques s' appuyant sur une modélisation structurelle.* PhD thesis, Université Joseph-Fourier-Grenoble I, 2008.

[YL91] C.-C. Yu and C. Lee. Fault diagnosis based on qualitative/quantitative process knowledge, 1991. Aiche journal.

[YL02] T.S. Yoo and S. Lafortune. Polynomial-time verification of diagnosability of partially observed discrete-event systems. *Automatic Control, IEEE Transactions on*, 47(9) :1491–1495, 2002.

Annexes

Annexe A

Exemples pour la superfluité

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.E^{Z_{ex_j}}$	E_{ex}	E_{ex}	E_{ex}	E_{ex}

TABLE A.1 – Ensemble des événements déclenchables pour γ_j

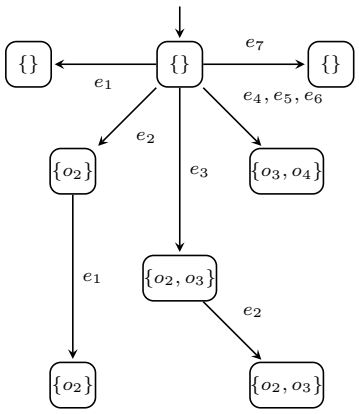


FIGURE A.1 – LTS observable γ_1

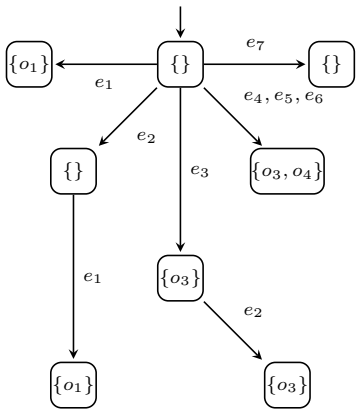


FIGURE A.2 – LTS observable γ_2

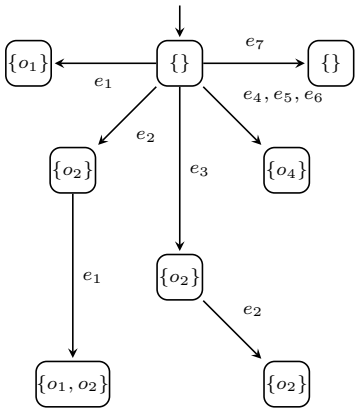


FIGURE A.3 – LTS observable γ_3

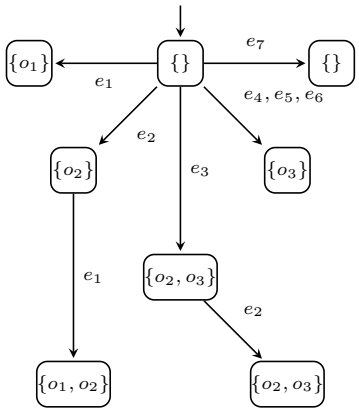


FIGURE A.4 – LTS observable γ_4

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.effets(Z_{ex_j}, e_1)$	$\{\{\}, \{o_2\}\}$	$\{\{o_1\}\}$	$\{\{o_1\}, \{o_1, o_2\}\}$	$\{\{o_1\}, \{o_1, o_2\}\}$
$\gamma_j.effets(Z_{ex_j}, e_2)$	$\{\{o_2\}, \{o_2, o_3\}\}$	$\{\{\}, \{o_3\}\}$	$\{\{o_2\}\}$	$\{\{o_2\}, \{o_2, o_3\}\}$
$\gamma_j.effets(Z_{ex_j}, e_3)$	$\{\{o_2, o_3\}\}$	$\{\{o_3\}\}$	$\{\{o_2\}\}$	$\{\{o_2, o_3\}\}$
$\gamma_j.effets(Z_{ex_j}, e_4)$	$\{\{o_3, o_4\}\}$	$\{\{o_3, o_4\}\}$	$\{\{o_4\}\}$	$\{\{o_3\}\}$
$\gamma_j.effets(Z_{ex_j}, e_5)$	$\{\{o_3, o_4\}\}$	$\{\{o_3, o_4\}\}$	$\{\{o_4\}\}$	$\{\{o_3\}\}$
$\gamma_j.effets(Z_{ex_j}, e_6)$	$\{\{o_3, o_4\}\}$	$\{\{o_3, o_4\}\}$	$\{\{o_4\}\}$	$\{\{o_3\}\}$
$\gamma_j.effets(Z_{ex_j}, e_7)$	$\{\{\}\}$	$\{\{\}\}$	$\{\{\}\}$	$\{\{\}\}$

TABLE A.2 – Effets des événements de γ_j dans Z_{ex_j}

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.Obs_{OK_{ex}}$	$\{\{\}, \{o_2\}\}$	$\{\{\}, \{o_1\}\}$	$\{\{\}, \{o_1, o_2\}\}$	$\{\{\}, \{o_1, o_2\}\}$
$\gamma_j.E_{FD}^{Z_{ex_j}}$	$\{e_3, e_4, e_5, e_6\}$	$\{e_3, e_4, e_5, e_6\}$	$\{e_2, e_3, e_4, e_5, e_6\}$	$\{e_2, e_3, e_4, e_5, e_6\}$
$\gamma_j.E_{fD}^{Z_{ex_j}}$	$\{e_2\}$	$\{e_2\}$	$\{e_1\}$	$\{e_1\}$
$\gamma_j.RDF_{Z_{ex_j}}$	$\frac{4}{7} \simeq 57\%$	$\frac{4}{7} \simeq 57\%$	$\frac{5}{7} \simeq 71\%$	$\frac{5}{7} \simeq 71\%$

TABLE A.3 – Caractéristiques de détectabilité pour γ_j dans Z_{ex_j}

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.groupeAmb(Z_{ex_j}, e_1)$	$\{e_1, e_2, e_7, e_I\}$	$\{e_1\}$	$\{e_1\}$	$\{e_1\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_2)$	$\{e_1, e_2, e_3\}$	$\{e_2, e_3, e_7, e_I\}$	$\{e_2, e_3\}$	$\{e_2, e_3\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_3)$	$\{e_2, e_3\}$	$\{e_2, e_3\}$	$\{e_2, e_3\}$	$\{e_2, e_3\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_4)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_5)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_6)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z_{ex_j}, e_7)$	$\{e_1, e_7, e_I\}$	$\{e_2, e_7, e_I\}$	$\{e_7, e_I\}$	$\{e_7, e_I\}$

TABLE A.4 – Groupes d'ambiguité des événements de γ_j dans Z_{ex_j}

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.degDiag(Z_{ex_j}, e_1)$	4	1	1	1
$\gamma_j.degDiag(Z_{ex_j}, e_2)$	3	4	2	2
$\gamma_j.degDiag(Z_{ex_j}, e_3)$	2	2	2	2
$\gamma_j.degDiag(Z_{ex_j}, e_4)$	3	3	3	3
$\gamma_j.degDiag(Z_{ex_j}, e_5)$	3	3	3	3
$\gamma_j.degDiag(Z_{ex_j}, e_6)$	3	3	3	3
$\gamma_j.degDiag(Z_{ex_j}, e_7)$	3	3	2	2
$\gamma_j.DDM_{Z_{ex_j}}$	$\frac{21}{7} = 3$	$\frac{19}{7} \simeq 2.71$	$\frac{16}{7} \simeq 2.29$	$\frac{16}{7} \simeq 2.29$

TABLE A.5 – Degrés de diagnostic des événements de γ_j dans Z_{ex_j}

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_1)$	$\{e_1, e_7, e_I\}$	$\{e_1\}$	$\{e_1\}$	$\{e_1\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_2)$	$\{e_2\}$	$\{e_2, e_7, e_I\}$	$\{e_2, e_3\}$	$\{e_2\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_3)$	$\{e_3\}$	$\{e_3\}$	$\{e_2, e_3\}$	$\{e_3\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_4)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_5)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_6)$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$	$\{e_4, e_5, e_6\}$
$\gamma_j.groupeAmb(Z^1_{ex_j}, e_7)$	$\{e_1, e_7, e_I\}$	$\{e_2, e_7, e_I\}$	$\{e_7, e_I\}$	$\{e_7, e_I\}$

TABLE A.6 – Groupes d'ambiguité des événements de γ_j dans $Z^1_{ex_j} = \{z_1\}$

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.degDiag(Z^1_{ex_j}, e_1)$	3	1	1	1
$\gamma_j.degDiag(Z^1_{ex_j}, e_2)$	1	3	2	1
$\gamma_j.degDiag(Z^1_{ex_j}, e_3)$	1	1	2	1
$\gamma_j.degDiag(Z^1_{ex_j}, e_4)$	3	3	3	3
$\gamma_j.degDiag(Z^1_{ex_j}, e_5)$	3	3	3	3
$\gamma_j.degDiag(Z^1_{ex_j}, e_6)$	3	3	3	3
$\gamma_j.degDiag(Z^1_{ex_j}, e_7)$	3	3	2	2
$\gamma_j.DDM_{Z^1_{ex_j}}$	$\frac{17}{7} \simeq 2.43$	$\frac{17}{7} \simeq 2.43$	$\frac{16}{7} \simeq 2.29$	$\frac{14}{7} = 2$

TABLE A.7 – Degrés de diagnostic des événements de γ_j dans $Z^1_{ex_j} = \{z_1\}$

γ_j	γ_1	γ_2	γ_3	γ_4
$\gamma_j.Obs_{OK_{ex}}$	$\{\{\}, \{o_2\}\}$	$\{\{\}, \{o_1\}\}$	$\{\{\}, \{o_1, o_2\}\}$	$\{\{\}, \{o_1, o_2\}\}$
$\gamma_j.E_{FD}^{Z^1_{ex_j}}$	$\{e_3, e_4, e_5, e_6\}$	$\{e_3, e_4, e_5, e_6\}$	$\{e_1, e_2, e_3, e_4, e_5, e_6\}$	$\{e_1, e_2, e_3, e_4, e_5, e_6\}$
$\gamma_j.RDF_{Z^1_{ex_j}}$	$\frac{4}{7} \simeq 57\%$	$\frac{4}{7} \simeq 57\%$	$\frac{6}{7} \simeq 86\%$	$\frac{6}{7} \simeq 86\%$

TABLE A.8 – Caractéristiques de détectabilité pour γ_j dans $Z^1_{ex_j}$