Contribution à la commande sûre des Systèmes à Événements Discrets

Jean-Marc Roussel

HABILITATION A DIRIGER DES RECHERCHES

École Normale Supérieure de Cachan

Présentée par

Jean-Marc Roussel

Maître de Conférences
Docteur de l'ENS Cachan

Contribution à la commande sûre des Systèmes à Événements Discrets

---

Soutenue le 01/07/2014 devant le jury composé de :

Président :	Pr Janan Zaytoon	Université de Reims Champagne-Ardennes, chercheur au CReSTIC
Rapporteurs :	Pr Hassane Alla	Université Joseph Fourier à Grenoble, chercheur au gipsa-lab
	Pr Jean-Louis Boimond	Université d'Angers, chercheur au LARIS
	Pr Jean-François Pétin	Université de Lorraine, chercheur au CRAN
Examinateurs :	Pr Christophe Bérenguer	Grenoble INP, chercheur au gipsa-lab
	Pr Jean-Jacques Lesage	ENS Cachan, chercheur au LURPA

Laboratoire Universitaire de Recherche en Production Automatisée
ENS Cachan / Université Paris-Sud
61, avenue du Président Wilson, 94235 CACHAN CEDEX

Remerciements

Ce mémoire d'habilitation présente les résultats de recherche conduites au LURPA dans le cadre de l'Automatique des Systèmes à Événements Discrets et de l'analyse de sûreté de fonctionnement.

Je tiens tout d'abord à remercier tous les collègues de ces deux champs disciplinaires pour les échanges que nous avons eu durant ces nombreuses années. Ces échanges ont été tout d'abord scientifiques puis, avec nombreux d'entre-vous, plus personnels. Vous être nombreux à m'avoir conseillé de rédiger ce mémoire en connaissant la teneur de mes travaux.

Je tiens à remercier les membres de ce jury pour leur participation : Janan Zaytoon pour m'avoir fait l'honneur de le présider, Hassane Alla, Jean-Louis Boimond et Jean-François Pétin pour avoir accepté de rapporter sur ce document, Christophe Bérenguer pour l'avoir examiné. Je tiens à remercier plus particulièrement Jean-Jacques Lesage pour avoir accepté le rôle de garant mais surtout pour ses conseils et ses nombreux encouragements.

Je tiens également à remercier l'ensemble des membres ou anciens membres du LURPA comme ceux du département de Génie Mécanique de L'ENS Cachan. L'ambiance présente dans ces deux entités permet de s'investir pleinement dans son travail tout en partageant de chaleureux moments.

Je tiens également à remercier tous les étudiants de thèse, de master ou de DEA avec qui j'ai eu l'occasion de travailler et plus particulièrement ceux qui m'ont fait confiance en me demandant de les encadrer.

Introduction

Ce mémoire a été rédigé en vue d'obtenir l'Habilitation à Diriger des Recherches. J'y présente mon implication dans l'enseignement et la recherche depuis mon recrutement à l'ENS Cachan en tant que maître de conférences en septembre 1995.

Mes activités de recherche rentrent dans le spectre de la section 61 du CNU et ont pour domaine l'Automatique des Systèmes à Événements Discrets (SED). Elles sont conduites en vue d'accroître la sûreté de fonctionnement des systèmes automatisés comme ceux qu'il est possible de trouver dans le cadre de la production manufacturière, de la production d'énergie ou du transport. Une grande partie de mes recherches a concerné la conception sûre des systèmes de contrôle-commande à base d'Automates Programmables Industriels (API). J'ai ainsi eu l'occasion d'approfondir les thématiques suivantes :

- la vérification formelle de programmes de contrôle-commande,
- la synthèse algébrique de programmes de contrôle-commande à partir de spécifications informelles,
- le test de conformité d'un contrôleur logique vis-à-vis de sa spécification.

Fort de cette expérience dans le domaine des SED, je me suis part ailleurs intéressé à la formalisation des outils pour l'analyse de sûreté, utilisés dans le cadre de l'analyse prévisionnelle des risques d'un équipement ou d'une installation industrielle. Cette formalisation des outils utilisés en sûreté a été faite en examinant avec un point de vue SED une problématique qui ne l'était pas à son origine. Je me suis successivement penché sur :

- la modélisation algébrique des arbres de défaillances dynamiques,
- l'analyse prévisionnelle des risques d'un point de vue qualitatif pour les systèmes réparables à partir de Boolean logic Driven Markov Processes (BDMPs),

- l’analyse prévisionnelle des risques d’un point de vue quantitatif pour les systèmes réparables à l’aide de chaînes de Markov.

D'une manière générale, mes activités de recherche ont pour objectif de proposer des apports formels ou méthodologiques à des outils de modélisation généralement issus de l’industrie tout en répondant à des besoins industriels déjà présents ou sur le point de le devenir.

Ce mémoire comporte quatre chapitres. Le premier chapitre présente mon curriculum vitae et une synthèse de mes activités d’enseignement et de recherche. Les encadrements que j’ai réalisés, ma production scientifique et ma participation à la vie scientifique y sont également listés.

Les chapitres 2 et 3 constituent le cœur de ce mémoire. Il y est proposé une synthèse des résultats scientifiques obtenus dans les deux domaines étudiés. Le chapitre 2 est consacré aux travaux relatifs à la conception sûre des systèmes de contrôle-commande à base d’APIs. Le chapitre 3 est consacré aux travaux conduits dans le cadre de la formalisation des outils utilisés en sûreté en utilisant les paradigmes et modèles des SED.

Dans le quatrième et dernier chapitre, après un bilan succinct, j’envisage quelques perspectives pour ces travaux.

En annexe, quatre publications sont jointes pour que le lecteur puisse trouver plus de détails concernant ces travaux.

Sommaire

Chapitre 1

Présentation synthétique

1.1. Curriculum Vitae

1.1.1. État civil

Jean-Marc ROUSSEL
Né le 11 août 1967 à Port-de-Bouc (13), Marié, 2 enfants
LURPA, ENS Cachan
61, avenue du Président Wilson
94235 Cachan Cedex
01 47 40 29 97
jean-marc.roussel@lurpa.ens-cachan.fr

1.1.2. Titre et diplômes

- Doctorat de l'ENS Cachan en Automatique, Mention « Très honorable avec les félicitations du jury » délivré le 16 décembre 1994,
  Titre : *Analyse de grafcets par génération logique de l'automate équivalent*
  Jury : J.-P. Frachet (Président), F. Prunet & P. Lhoste (Rapporteurs), P. Bourdet (Directeur de thèse), J.-J. Lesage (Co-encadrant), O. Douchin (Examinateur)
- DEA Production Automatisée de l'ENS Cachan, Mention « Assez bien » septembre 1991,
  Titre : *Analyse et développement d'un atelier logiciel expérimental pour la modélisation par Grafcet*
  Responsable de la recherche : J.-J. Lesage
- Lauréat du concours de l'Agrégation Externe de Génie Mécanique, juin 1990 (2ième).
- Maîtrise de technologie de la Construction de l'Université Paris VI, Option fabrication mécanique, Mention « Assez bien », juin 1989
- Licence de technologie de la Construction de l'Université Paris VI, Mention « Bien », juin 1988

1.1.3. Situation actuelle

- Maître de Conférences (Section 61) au 8ème échelon à l'ENS Cachan
  - Enseignement au sein du Département Génie Mécanique (DGM) de l'ENS Cachan,
  - Recherche au LURPA (Laboratoire Universitaire de Production Automatisée - EA 1385) de l'ENS Cachan. Responsable de l'équipe ISA du LURPA depuis novembre 2010.

1.1.4. Emplois occupés

- Maître de Conférences à l'ENS Cachan (depuis 1995)
  - Enseignement au Département de Génie Mécanique de l'ENS Cachan (depuis 2000)
  - Enseignement au Département de Génie Mécanique de l'IUFM Créteil (1995-2000)
- Doctorant sur un statut Allocataire Moniteur Normalien (AMN), ENS Cachan (1991-1995)
  - Scientifique du contingent, École Militaire à Paris (1992-1993)
- Élève Professeur, ENS Cachan (1987-1991)

1.2. Synthèse des activités de recherche et d'enseignement

1.2.1. Synthèse des activités de recherche et d'administration de la recherche

- **Encadrement doctoral et de stages de recherche** (cf. page 10 pour plus de détails)
  - Thèses soutenues (% cumulé d'encadrement) : **5** (250%)
  - Thèse en cours (% cumulé d'encadrement) : **1** (50%)
  - Stages de recherche (DEA, Master...) : **13**
- **Production scientifique** (cf. page 13 pour plus de détails)
  - Revues internationales avec comité de lecture : **9**
  - Revues nationales avec comité de lecture : **3**
  - Conférences internationales avec comité de lecture : **36**
  - Conférences nationales avec comité de lecture : **16**
  - Conférences avec actes à diffusion restreinte : **8**
  - Brevet : **1**
  - Rapports de contrats : **11**
- Titulaire de la PEDR de septembre 1997 à juin 2001 et de septembre 2005 à juin 2009.
- **Animation scientifique** (cf. page 21 pour plus de détails)
  - Responsable de l'équipe ISA du LURPA depuis novembre 2010
  - Représentant national au comité technique « TC 3.1 Computers for Control » de l'IFAC depuis novembre 2011
  - Collaborations internationales : **5**
  - Collaborations nationales : **2**
  - Participation à des groupes de travail nationaux : **2**
  - Participation à des programmes de recherche ANR : **1**
  - Responsable ou participation à des contrats industriels : **4**
  - Participation au comité de programmes de conférences : **12**
  - Participation au comité d'organisation de conférences : **5**
  - Organisations de sessions invités : **1**
  - Organisation de tutoriaux : **1**
  - Évaluation de publications en revues : **23**
  - Évaluation de communications en conférences (hors conférences en tant qu'IPC) : **42**
  - Participation à des jurys de thèse : **1**

1.2.2. Synthèse des responsabilités d'enseignement et d'administration de l'enseignement

(cf. page 25 pour plus de détails)

- Responsabilités au sein de l'ENS Cachan dans le cadre de mes activités d'enseignement
  - Responsable d'un laboratoire d'enseignement au sein du DGM depuis septembre 2000
  - Responsable pédagogique pour l'ENS Cachan du master Recherche co-habilité avec l'Université de Lorraine, depuis septembre 2005 (Enseignement en visio-conférence)
- Participation au recrutement d'enseignants de l'Éducation Nationale
  - Membre du jury de l'Agrégation Interne de Génie Mécanique pour les sessions 2009 à 2011
- Participation au recrutement de Maîtres de Conférences
  - Membre de commissions de spécialistes dans 4 établissements distincts entre 1998 et 2008
  - Membre de 4 comités de sélection dont un en tant que président

1.3. Encadrement doctoral et de stages de recherche

J'ai eu la chance de co-encadrer 14 étudiants lors de leur formation scientifiques dont 5 à la fois pour leur thèse et leur DEA ou Master Recherche.

1.3.1. Co-Encadrements de thèses

1.3.1.1. Thèses soutenues

Ces co-encadrements de thèse ont été assurés sous la direction de Jean-Jacques Lesage ou de Jean-Marc Faure.

T1 Antonio Médina Rodriguez
Méthode de synthèse d'un contrôleur logique à partir de spécifications algébriques
Date de soutenance : 05 décembre 2007
Directeur de Recherche : J.-M. Faure (50%), Co-Encadrant : J.-M. Roussel (50%)
Jury : E. Rutten (Président), V. Carré-Ménétrier & H. Alla (Rapporteurs)
Financement : Bourse du Conacyt (Centre national des sciences et technologies du Mexique)
Publications : 1 revue : [A1], 2 conférences : [C44], [C45]
Situation actuelle : Ingénieur informatique pour la délégation mexicaine auprès de l'OCDE à Paris

T2 Yann Hietter
Synthèse algébrique de la loi de commande d'un système à événements discrets logique
Date de soutenance : 28 mai 2009
Directeur de Recherche : J.-L. Lesage (50%), Co-Encadrant : J.-M. Roussel (50%)
Jury : J.-L. Ferrier (Président), J. Zaytoon & J.-F. Petin (Rapporteurs), E. Craye (Examinateur)
Financement : Bourse ASN (Allocation Spécifique Normalien)
Publications : 3 conférences : [C18], [C19], [C48]
Situation actuelle : Professeur Agrégé à l'ENSISA, Mulhouse

T3 Guillaume Merle
Algebraic modelling of Dynamic Fault Trees, contribution to qualitative and quantitative analysis
Date de soutenance : 7 juillet 2010
Directeur de Recherche : J.-L. Lesage (50%), Co-Encadrant : J.-M. Roussel (50%)
Jury : A. Bobbio (Président), C. Bérenguer & F. Ortmeier (Rapporteurs), A. Rauzy (Examinateur)
Financement : Bourse MESR
Publications : 3 revues : [A3], [A5], [A7], 6 conférences : [C17], [C21], [C24], [C25], [C26], [C46]
Situation actuelle : Professeur agrégé à l'Ecole centrale de Pékin après un Post-Doc au LIAFA (INRIA, Académie des Sciences Chinoise), Pékin (Chine)

T4 Julien Provost
Test de conformité de contrôleurs logiques spécifiés en Grafcet
Date de soutenance : 8 juillet 2011
Directeur de Recherche : J.-M. Faure (50%), Co-Encadrant : J.-M. Roussel (50%)
Jury : J. Zaytoon (Président), H. Alla & T. Jéron (Rapporteurs), F. Corbier (Examinateur)
Financement : Bourse ASN (Allocation Spécifique Normalien)
Publications : 2 revues : [A6] [A9], 7 conférences : [C22], [C23], [C27], [C28], [C49], [C50], [C58]

Situation actuelle : Assistant Professor au TU Munich (Allemagne) après un Post-doc à l'université de Chalmers (Suède)

T5 Pierre-Yves Chaux
Analyse qualitative des Boolean Driven Markov Processes
Date de soutenance : 15 avril 2013
Directeur de Recherche : J.-L. Lesage (50%), Co-Encadrant : J.-M. Roussel (50%)
Jury : J.-F. Petin (Président), C. Bérenguer & A. Rauzy (Rapporteurs), M. Bouissou & G. Deleuze (Examinateurs)
Financement : Convention CIFRE avec EDF, Centre de recherche de Clamart
Publications : 4 conférences : [C29], [C31], [C34], [C51]
Situation actuelle : Post-doctorant au CRAN, Université de Lorraine

1.3.1.2. Thèses en cours

T6 Pierre-Antoine Brameret
Un modèle formel pour l'analyse prévisionnelle des défaillances des systèmes réparables
Début des travaux : septembre 2012
Co-direction avec A. Rauzy (Professeur à l'Ecole Centrale-Supelec & professeur associé à l'École Polytechnique)
Financement : Contrat Doctoral fléché pour Normaliens
Publications : 4 conférences : [C35], [C36], [C59], [C60]

1.3.2. Encadrements de stagiaires de DEA puis de Master Recherche

Tous les encadrements de stagiaires ont été réalisés en pleine responsabilité.

1.3.2.1. Stagiaires de DEA

S1 Christophe Thierry
Extension de l'automate équivalent à un Grafcet par association des variables de sortie, Introduction de contentions sur les variables d'entrées
DEA de Production Automatisée, ENS Cachan, 1996
Situation actuelle : Professeur Agrégé à l'IUT du Havres

S2 Olivier Grabinski
Apports du calcul symbolique pour la conception sûre de programmes de contrôle/ commande écrits en Ladder
DEA de Production Automatisée, ENS Cachan, 2002
Situation actuelle : Professeur Agrégé à l'IUT de Cachan

S3 Jérôme Antoine
Génération automatique de jeux de tests pour systèmes séquentiels et temporisés contenus dans un automate programmable industriel
DEA de Production Automatisée, ENS Cachan, 2003
Situation actuelle : Proviseur-adjoint au Lycée de Saint-Valery-en-Caux.

S4 Olivier Cardin
Apport de la réécriture pour la vérification d'un programme de commande par Model-Checking : Cas du Ladder Diagram
DEA de Production Automatisée, ENS Cachan, 2004
Situation actuelle : Maître de Conférences à l'IUT de Nantes

S5 Yann Hietter
Spécification d'un contrôleur logique permettant sa synthèse de manière algébrique
DEA de Production Automatisée, ENS Cachan, 2005
Situation actuelle : Professeur Agrégé à l'ENSISA à l'issue d'une thèse au LURPA [T2]

1.3.2.2. Stagiaires de Master 2 Recherche

S6 Matteo Cantarelli
Control system design using the Supervisory Control Theory: evaluations of possibilities and limits
Master IS en EEA-PR, ENS de Cachan, 2006
Publication : 1 conférence : [C20]
Situation actuelle : Ingénieur de développement au sein de la société PILZ en Irlande

S7 Guillaume Merle
Algebraic modelling of Fault Trees with Priority AND gates
Master IS en EEA-PR, ENS de Cachan, 2006
Situation actuelle : Professeur agrégé à l'Ecole centrale de Pékin après un Post-Doc au LIAFA (INRIA, Académie des Sciences Chinoise), à l'issue d'une thèse au LURPA [T3]

S8 Julien Provost
Du modèle logique séquentiel à la machine de Mealy à base d'alphabet : une transcription adaptée aux techniques de test
Master IS en EEA-PR, ENS de Cachan, 2008
Situation actuelle : Assistant Professor au TU Munich (Allemagne) après un Post-doc à l'université de Chalmers (Suède) après d'une thèse au LURPA [T4]

S9 Pierre-Yves Chaux
Apport du model-checking pour l'analyse qualitative de BDMP
Master IS en EEA-PR, ENS de Cachan, 2009
Situation actuelle : Post-doctorant au CRAN à Nancy après d'une thèse au LURPA [T5]

S10 Pierre-Antoine Brameret
Determining the availability of a system built with repairable components
Master ISC, ENS de Cachan, 2011
Situation actuelle : Doctorant au LURPA [T6]

S11 Anaïs Guignard
Symbolic generation of the automaton representing an algebraic description of a logic system
Master ISC, ENS de Cachan, 2011
Publication : 1 conférence : [C52]
Situation actuelle : Doctorante au LURPA

S12 Hélène Leroux
Algebraic synthesis of logical controllers with optimization criteria
Publication : 1 conférence : [C32]
Master ISC, ENS de Cachan, 2011
Situation actuelle : Doctorante au LIRMM à Montpellier

1.3.2.3. Stagiaires de Recherche étrangers

S13 Christiano Poddie
Analysis of grafcet models by automatic generation of the equivalent timed automaton
Master thesis in Electronic Engineering, Università di Caglari (Italie), 2009
Situation actuelle : Ingénieur de développement au sein de la société Trenitalia en Italie

1.4. Production scientifique

En faisant abstraction des étudiants et des membres du LURPA, mes différents co-auteurs sont :

- *au niveau local* : B. Bérard (LSV puis LIP6), N. Vayatis (CMLA)
- *au niveau national* : M. Bouissou (EDF R&D, Clamart), V. Carré-Ménétrier (CRESTIC, Reims), G. Deleuze (EDF R&D, Clamart), P. Lhoste (CRAN, Nancy), A. Rauzy (LIX, Palaiseau), B. Riéra (CRESTIC, Reims), J. Zaytoon (CRESTIC, Reims)
- *au niveau international* : A. Bobbio (Italie), J.-C. Ferreira Da Silva (Portugal), A. Giua (Italie), M. Kwiatkowska (Angleterre), E. Lopez-Mellado (Mexique)

1.4.1. Revues avec comité de lecture

Le facteur d'impact indiqué est celui du « ISI Web of Knowledge, Journal Citation Report » à la date de constitution ce mémoire. Les articles [A4] [A5] [A8] [A9] sont donnés en annexe.

1.4.1.1. Revues internationales

A1 Algebraic approach for dependable logic control systems design
J.-M. Roussel, J.-M. Faure, J.-J. Lesage, A. Médina
International Journal of Production Research, 42(14), pp. 2859-2876, 2004, (IF JCR : 1.46)

A2 Designing dependable controllers using algebraic specifications
J.-M. Roussel, J.-M. Faure
Control Engineering Practice, 14(10), pp. 1143-1155, 2006, (IF JCR : 1.669)

A3 Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events
G. Merle, **J.-M. Roussel**, J.-J. Lesage, A. Bobbio
IEEE Transactions on Reliability, 59(1), pp. 250-261, 2010, (IF JCR : 2.293)

A4 Verification of a timed multitask system with Uppaal
H. Bel Mokadem, B. Bérard, V. Gourcuff, O. De Smet, **J.-M. Roussel**
IEEE Transactions on Automation Science and Engineering, 7(4), pp. 921-932, 2010, (IF JCR : 1.674)

A5 Algebraic determination of the structure function of dynamic fault trees
G. Merle, **J.-M. Roussel**, J.-J. Lesage
Reliability Engineering and System Safety, 96(2), pp. 267-277, 2011, (IF JCR : 1.901)

A6 Translating Grafcet specifications into Mealy machines for conformance test purposes
J. Provost, **J.-M. Roussel**, J.-M. Faure
Control Engineering Practice, 19(9), pp. 947-957, 2011, (IF JCR : 1.669)

A7 Quantitative Analysis of Dynamic Fault Trees based on the Structure Function
G. Merle, **J.-M. Roussel**, J.-J. Lesage
Reliability Engineering International, 30(1), pp. 143-156, 2014 available online since January 2013, (IF JCR : 0,68)

A8 Design of logic controllers thanks to symbolic computation of simultaneously-asserted Boolean equations
J.-M. Roussel, J.-J. Lesage
Mathematical Problems in Engineering, 2014 (Article ID 726246), 16 pages, accepté le 7 février 2014 (IF JCR : 1,383)

A9 Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers
J. Provost, **J.-M. Roussel**, J.-M. Faure
IEEE Transactions on Industrial Informatics, 9 pages, accepté le 28 mars 2014 (IF JCR : 3.381)

1.4.1.2. Revues nationales

A10 Hierarchical approach to Grafcet using forcing order
J.-J. Lesage, **J.-M. Roussel**
APII- AFCET/CNRS, Ed. Hermès, 27(1), pp. 25-38, 1993

A11 Une algèbre de Boole pour l'approche événementielle des systèmes logiques
J.-M. Roussel, J.-J. Lesage
APII-AFCET/CNRS, Ed Hermès, 27(5), pp. 541-560, 1993

A12 Safety properties verification of ladder diagram programs
J.-M. Roussel, B. Denis
Journal Européen des Systèmes Automatisés, 36(7), pp. 905-917, 2002

1.4.2. Conférences

1.4.2.1. Conférences internationales avec comité de lecture et actes

C1 *A Boolean algebra for a formal expression of events in logical systems*
B. Denis, J.-J. Lesage, **J.-M. Roussel**
1st Mathmod Conference (MATHMOD'94), Vienna (Austria), pp. 859-862, February 1994

C2 *A method for design and valuation of manufacturing system control architecture*
B. Denis, J.-J. Lesage, **J.-M. Roussel**
IEEE Annual conference on Systems, Man, Cybernetics Society (SMC'95), Vancouver (Canada), pp. 4486-4491, October 1995

C3 *A theory of binary signal*
J.-J. Lesage, **J.-M. Roussel**, C. Thierry
Computational Engineering in Systems Applications EMACS-IEEE/SMC Multiconference (CESA'96), Lille (France), pp. 590-595, July 1996

C4 *Validation and verification of Grafcet using state machine*
J.-M. Roussel, J.-J. Lesage
Computational Engineering in Systems Applications EMACS-IEEE/SMC Multiconference (CESA'96), Lille (France), pp. 758-764, July 1996

C5 *On the supremal controllable Grafcet of a given Grafcet*
J. Zaytoon, C. Ndjab, **J.-M. Roussel**
2nd Mathmod Conference (MATHMOD'97), Vienna (Austria), pp. 371-376, February 1997

C6 *Formal validation of PLC programs : a survey*
S. Lampérière-Couffin, O. Rossi, J.-J. Lesage, **J.-M. Roussel**
5th European Control Conference (ECC'99), Karlsruhe (Germany), CD-Rom paper 741 6 pages, September 1999

C7 *A formal expression of time for discrete-events dynamic systems*
C. Thierry, **J.-M. Roussel**, J.-J. Lesage
3th Mathmod Conference (MATHMOD'2000), Vienna (Austria), pp. 445-448, February 2000

C8 *An extended boolean algebra for the control of logical systems*
C. Thierry, **J.-M. Roussel**, J.-J. Lesage
16th IMACS World Congress, Lausanne (Switzerland), CDRom paper 320 6 pages, August 2000

C9 *Modeling and implementing the control of automated production systems using statecharts and PLC programming languages*
J.-M. Machado, F. Louni, J.-M. Faure, J.-J. Lesage, J.-C. Ferreira Da Silva, **J.-M. Roussel**
6th European Control Conference (ECC'2001), Porto (Portugal), pp. 1019-1024, September 2001

C10 *Formal verification of industrial control systems*
O. De Smet, J.-J. Lesage, **J.-M. Roussel**
10th IFAC Symposium on Information Control Problems in Manufacturing (INCOM'2001), Vienna (Austria), CDRom paper 6 pages, September 2001

C11 *An algebraic approach for PLC programs verification*
J.-M. Roussel, J.-M. Faure
6th International Workshop on Discrete Event Systems (WODES'02), Zaragoza (Spain), pp. 303-308, October 2002

C12 *Towards automatic verification of ladder logic programs*
B. Zoubek, **J.-M. Roussel**, M. Kwiatkowska
IEEE International Conference on Computational Engineering in System Applications (CESA'03), Lille (France), CDRom paper S2-I-04-0169 6 pages, July 2003

C13 *Designing dependable logic controllers using algebraic specification*
J.-M. Roussel, J.-M. Faure
7th International Workshop on Discrete Event Systems (WODES'04), Reims (France), pp. 313-318, September 2004

C14 *Designing dependable logic controllers using the supervisory control theory*
J.-M. Roussel, A. Giua
16th IFAC World Congress, Praha (Czech Republic), CDRom paper 04427 6 pages, July 2005

C15 *Verification of a timed multitask system with UPPAAL*
H. Bel Mokadem, B. Bérard, V. Gourcuff, **J.-M. Roussel**, O. De Smet
10th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'05), Catania (Italy), CDRom paper CF-000606, September 2005

C16 *A methodology to design and check a plant model*
B. Rohée, B. Riéra, V. Carré-Ménétrier, **J-M. Roussel**
3rd IFAC Workshop on Discrete-Event System Design (DESDes'06), Rydzyna (Poland), pp. 246-250, September 2006

C17 *Algebraic modelling of fault trees with priority AND gates*
G. Merle, **J.-M. Roussel**
1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07), Cachan (France), pp. 175-180, June 2007

C18 *Algebraic synthesis of dependable logic controllers*
Y. Hietter, **J.-M. Roussel**, J.-J. Lesage
17th IFAC World Congress, Seoul (Korea), pp. 4132-4137, July 2008

C19 *Algebraic synthesis of transition conditions of a state model*
Y. Hietter, **J.-M. Roussel**, J.-J. Lesage
9th International Workshop On Discrete Event Systems (WODES'08), pp. 187-192, Göteborg (Sweden), May 2008

C20 *Reactive control system design using the Supervisory Control Theory: evaluation of possibilities and limits*
M. Cantarelli, **J.-M. Roussel**
9th International Workshop On Discrete Event Systems (WODES'08), pp. 200-205, Göteborg (Sweden), May 2008

C21 *Algebraic expression of the structure function of a subclass of dynamic fault trees*
G. Merle, **J.-M. Roussel**, J.-J. Lesage, A. Bobbio
2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09), Bari (Italy), pp. 129-134, June 2009

C22 *Test sequence construction from SFC specification*
J. Provost **J.-M. Roussel** J.-M. Faure
2nd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'09), Bari (Italy), pp. 341-346, June 2009

C23 *SIC-testability of sequential logic controllers*
J. Provost, **J.-M. Roussel**, J.-M. Faure
10th International Workshop On Discrete Event Systems (WODES'10), Berlin (Germany), pp. 203-208, August 2010

C24 *Analytical calculation of failure probabilities in dynamic fault trees including spare gates*
G. Merle, **J.-M. Roussel**, J.-J. Lesage, N. Vayatis
19th European Safety & Reliability Conference (ESREL'11), Rhodes (Greece), pp. 794-801, September 2010

C25 *Improving the efficiency of dynamic fault tree analysis by considering gates FDEP as static*
G. Merle, **J.-M. Roussel**, J.-J. Lesage
19th European Safety & Reliability Conference (ESREL'11), Rhodes (Greece), pp. 845-851, September 2010

C26 *Dynamic fault tree analysis based on the structure function*
G. Merle, **J.-M. Roussel**, J.-J. Lesage
Annual Reliability and Maintainability Symposium 2011 (RAMS 2011), Lake Buena Vista FL(USA), pp. 462-467, January 2011

C27 *Testing programmable logic controllers from finite state machines specification*
J. Provost, **J.-M. Roussel**, J.-M. Faure
3rd IFAC Workshop on Dependable Control of Discrete Systems (DCDS'11), Saarbrücken (Germany), pp. 3-8, June 2011

C28 *A formal semantics for Grafcet specifications*
J. Provost, **J.-M. Roussel**, J.-M. Faure
IEEE 7th International Conference on Automation Science and Engineering (CASE 2011), Trieste (Italy), pp. 488-494, August 2011

C29 *Qualitative analysis of a BDMP by finite automaton*
P.-Y. Chaux, **J.-M. Roussel**, J.-J. Lesage, G. Deleuze, M. Bouissou
20th European Safety & Reliability Conference (ESREL'11), Troyes (France), in "Advances in Safety Reliability and risk management", Taylor & Francis Ed., pp. 2055-2057, September 2011

C30 *Translation from Petri nets into Boolean equations for the algebraic design of logic controllers*
M. Diaz-Rodriguez, E. Lopez-Mellado, P.-A. Brameret, **J-M. Roussel**
8th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE 2011), Merida (Mexico), 6 pages, October 2011

C31 *Systematic extraction of Minimal Cut Sequences from a BDMP model*
P.-Y. Chaux, **J.-M. Roussel**, J.-J. Lesage
21th European Safety & Reliability Conference (ESREL'12), Helsinki (Finland), Paper 16B-We4-3 8 pages, June 2012

C32 Algebraic synthesis of logical controllers with optimization criteria
H. Leroux, **J.-M. Roussel**,
6th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS 2012), Paris (France), pp. 103-114, August 2012

C33 Algebraic synthesis of logical controllers despite inconsistencies in specifications
J.-M. Roussel, J.-J. Lesage
11th International Workshop On Discrete Event Systems (WODES'12), Guadalajara (Mexico), pp. 307-314, October 2012

C34 Towards an unified definition of Minimal Cut Sequences
P.-Y. Chaux, **J.-M. Roussel**, J.-J. Lesage, G. Deleuze, M. Bouissou
4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS'13), York (United Kingdom), Paper n°1, September 2013

C35 Preliminary System Safety Analysis with Limited Markov Chain Generation
P-A. Brameret, **J.-M. Roussel**, A. Rauzy
4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2013), York (United Kingdom), Paper n°3, September 2013

C36 The AltaRica 3.0 Project for Model-based Safety Assessment
T. Prosvirnova, M. Batteux, P.-A. Brameret, L. Kloul, A. Cherfi, T. Friedlhuber, **J.-M. Roussel**, A. Rauzy,
4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS'13), York (United Kingdom), Paper n°22, September 2013

1.4.2.2. Conférences nationales ou francophones avec comité de lecture et actes

C37 *Preuve de la cohérence d'une hiérarchie de forçage entre grafcets partiels*
J.-J. Lesage, **J.-M. Roussel**
Congrès GRAFCET'92, Paris, pp. 125-134, mars 1992

C38 *Définition d'un cadre formel Pour l'expression et la vérification de propriétés d'un modèle Grafcet*
J.-M. Roussel, J.-J. Lesage
1er Congrès MSR'96, Brest, pp. 229-237, mars 1996

C39 *Représentation et manipulation d'un automate équivalent à un Grafcet à l'aide de BDD*
M. Gataa, **J.-M. Roussel**
Congrès AGI'96, Tours, pp. 233-236, juin 1996

C40 *Réactivité et déterminisme du comportement temporel du grafcet*
J.-J. Lesage, **J.-M. Roussel**, J.-M. Faure, P. Lhoste, J. Zaytoon
3ème conférence ADPM'98, Reims, pp. 99-106, mars 1998

C41 *Identification de machine séquentielle binaire : application à un système réactif*
O. De Smet, **J.-M. Roussel**, N. Hévin
2ème congrès MSR'99, Cachan, pp. 351-360, mars 1999

C42 *Formalisation des opérateurs temporels utilisés pour la description des systèmes à événements dicrets*
C. Thierry, **J.-M. Roussel**
Journées Doctorales d'Automatique (JDA'99), Nancy, pp. 289-292, septembre 1999

C43 *Vérification de propriétés de sûreté dans les programmes Ladder Diagram*
J.-M. Roussel, B. Denis
3ème congrès MSR'01, Toulouse, pp. 225-240, octobre 2001

C44 *Modélisation d'un système logique séquentiel élémentaire à partir d'une spécification algébrique*
A. Médina, **J.-M. Roussel**
Journées Doctorales d'Automatique (JDA'03), Valenciennes, pp. 245-250, juin 2003

C45 *Synthèse d'un programme de commande d'un système logique à partir de l'expression algébrique de ses spécifications*
J.-M. Roussel, A. Médina, J.-M. Faure
4ème congrès MSR'03, Metz, pp. 77-93, octobre 2003

C46 *Modèle algébrique des arbres de défaillance intégrant des contraintes sur l'ordre d'occurrence des événements*
G. Merle, **J.-M. Roussel**
Journées Doctorales / Journées Nationales MACS (JD-JN-MACS'07), Reims, Papier 40, juillet 2007

C47 *Outil d'aide à l'élaboration de modèles hybrides de simulation pour les systèmes manufacturiers*
B. Rohée, B. Riéra, V. Carré-Ménétrier, **J.-M. Roussel**
Journées Doctorales / Journées Nationales MACS (JD-JN-MACS'07), Reims, Papier 62, juillet 2007

C48 *Calcul des conditions de transition d'un réseau de Petri par synthèse algébrique*
Y. Hietter, **J.-M. Roussel**, J.-J. Lesage
5ième Conférence Internationale Francophone d'Automatique (CIFA 2008), Bucarest (Roumanie), Papier 83, septembre 2008

C49 *Construction d'une séquence de test minimale à partir d'une spécification GRAFCET*
J. Provost, **J.-M. Roussel**, J.-M. Faure
Journées Doctorales / Journées Nationales MACS (JD-JN-MACS'09), Angers, Papier 19, mars 2009

C50 *Test exhaustif de contrôleurs logiques spécifiés en Grafcet : apports et limites d'une modélisation par machines de Mealy*
J. Provost, **J.-M. Roussel**, J.-M. Faure
7ième congrès MSR'09, Nantes, pp. 889-904, novembre 2009

C51 *Formalisation des scénarios de défaillance d'un BDMP par automate fini*
P.-Y. Chaux, **J.-M. Roussel**, J.-J. Lesage
Journées Doctorales / Journées Nationales MACS (JD-JN-MACS'11), Marseille, pp. 159-164, juin 2011

C52 *Génération d'une machine de Mealy à partir de spécifications algébriques à des fins de test de conformité*
A. Guignard, **J.-M. Roussel**, J.-M. Faure
7ième Conférence Internationale Francophone d'Automatique (CIFA 2012), Grenoble, pp.907-912, juillet 2012

1.4.2.3. Conférences avec ou sans comité de lecture avec actes à diffusion restreinte

C53 *Modèles de spécification fonctionnelle de la commande des systèmes de production : synthèse de trois études de Cas*
L. Piétrac, G. Timon, B. Denis, J.-J. Lesage, **J.-M. Roussel**
Journée PRIMECA les systèmes de production, Clermont-Ferrand, décembre 1994

C54 *AGGLAE : un outil d'aide à la validation des grafcets de spécification*
J.-M. Roussel, J.-J. Lesage
Journées d'Etude sur les Logiciels pour le traitement de l'Image, du Signal et l'Automatique (ELISA'97), Vandoeuvre-les-Nancy, 8 pages, mars 1997

C55 *Validation de spécifications : une expérience sur le grafcet*
J.-M. Roussel
Journée technique « Sûreté de fonctionnement des systèmes automatisés de production » du Lycée R. Dautry, Limoges, décembre 1998

C56 *IEC 60848 et IEC 61131-3 : deux normes complémentaires*
J.-M. Roussel, S. Lampérière-Couffin, J.-J. Lesage
Journée d'études « Nouvelles percées dans les langages pour l'automatique » de la SEE - Club 18, Amiens, novembre 1999

C57 *La norme IEC 61131-3, des possibilités d'ouverture pour les utilisateurs et les fournisseurs*
J.-M. Roussel
Journée technique du Club AUTOMATION, Paris, septembre 2000

C58 *Un démonstrateur pour le test de conformité de contrôleurs logiques*
J. Provost, **J.-M. Roussel**, J.-M. Faure
3èmes Journées Démonstrateurs du club EEA, décembre 2010

C59 Assessing the dependability of systems with repairable and spare components
P.-A. Brameret, **J-M. Roussel**, A. Rauzy
18ième Conférence Lambda-mu, Tours, Papier 3D-9, 9 pages, octobre 2012

C60 Availability assessment of a complex system using limited build of Markov chains
P.-A. Brameret, **J-M. Roussel**, A. Rauzy
3rd International Workshop on Model Based Safety Assessment, mars 2013

1.4.3. Mémoires

M1 *Analyse et développement d'un atelier logiciel expérimental pour la modélisation par le Grafcet*
J.-M. Roussel
DEA de Production Automatisée, ENS Cachan, 1991

M2 *Analyse de grafcets par génération logique de l'automate équivalent*
J.-M. Roussel
Doctorat de l'ENS Cachan, 218 pages, 16 décembre 1994

1.4.4. Valorisation industrielle

1.4.4.1. Brevet

B1 *Dispositif et procédé d'analyse de performances et d'identification comportementale d'un système en tant qu'automate à événements discrets et finis*
B. Denis, O. De Smet, J.-J. Lesage, **J.-M. Roussel**
Brevet FR 01 10 933, août 2001

1.4.4.2. Rapports de contrats de recherche

R1 *Approche objet et ingénierie des systèmes automatisés de production*
J.-J. Lesage, **J.-M. Roussel**
Rapport de fin d'étude, Réf. EXERA : S 3601X91, 58 pages, novembre 1991

R2 *Étude De Cas Sedem-91 « Cahier des charges au format guide d'analyse des besoins et modélisation Idef0, Merise, Grafcet Et SART »*
B. Denis, J.-J. Lesage, **J.-M. Roussel**, G. Timon
Rapport de fin d'étude, Réf. EXERA : S 3609X92, 178 pages, avril 1992

R3 *Étude de cas Rafale-X54 « Cahier des charges au format guide d'analyse des besoins et modélisation Idef0, Merise, Gemma et Objets »*
B. Denis, J.-J. Lesage, **J.-M. Roussel**, G. Timon
Rapport de fin d'étude, Réf. EXERA : S 3615X93, 194 pages, mai 1993

R4 *De l'Analyse des besoins au cahier des charges d'un système d'automatisation de la production : synthèse sur l'apport des techniques de modélisation fonctionnelle pour la conception d'architectures*
B. Denis, J.-J. Lesage, **J.-M. Roussel**, G. Timon
Rapport de synthèse Réf. EXERA : S 3649X94, 32 pages, Avril 1994

R5 *Validation des SFC (Sequential Function Charts)*
S. Lampérière-Couffin, J.-J. Lesage, **J.-M. Roussel**
Projet VULCAIN, Rapport de fin de tâche n° 1, Contrat n° UAI/1.98 avec Alcatel CRC, 151 pages, novembre 1999

R6 *Définition des fonctionnalités de l'atelier GRAFCET pour le logiciel CONTROCAD*
J.-M. Roussel
Rapport intermédiaire de contrat ALSTOM POWER - LURPA, 15 pages, avril 2001

R7 *Constitution d'un méta-modèle de l'atelier GRAFCET pour le logiciel CONTROCAD, méta-modèle rédigé au format Entité-Association*
J.-M. Roussel
Rapport intermédiaire de contrat ALSTOM POWER - LURPA, 25 pages, mai 2001

R8 *Définition du module de génération de code de l'atelier GRAFCET pour le logiciel CONTROCAD*
J.-M. Roussel
Rapport final de contrat ALSTOM POWER - LURPA, 15 pages, juin 2001

R9 *Étude de la démarche, des méthodes et outils pour réaliser le portage d'un automatisme à relais de contrale nucléaire vers un automate programmable industriel : Choix d'un langage de programmation d'automates*
J.-M. Roussel, J.-M. Faure
Rapport n°1 du contrat EDF R&D - LURPA n°P11/F01377/0, 40 pages, Décembre 2002

R10 *Étude de la démarche, des méthodes et outils pour réaliser le portage d'un automatisme à relais de contrale nucléaire vers un automate programmable industriel : Règles de programmation en Ladder Diagram*
J.-M. Roussel, J.-M. Faure
Rapport n°2 du contrat EDF R&D - LURPA n°P11/F01377/0, 25 pages, Décembre 2002

R11 *Étude de la démarche, des méthodes et outils pour réaliser le portage d'un automatisme à relais de contrale nucléaire vers un automate programmable industriel : Tests de comportements combinatoires, séquentiels et temporisés*
J.-M. Roussel, J.-M. Faure
Rapport n°3 du contrat EDF R&D - LURPA n°P11/F01377/0, 47 pages, Décembre 2003

1.5. Participation à la vie scientifique et responsabilités collectives

1.5.1. Participation à la vie scientifique

1.5.1.1. Participation à la vie scientifique du LURPA

- Membre du conseil de laboratoire du LURPA de septembre 2010 à mars 2014.
- **Responsable de l'équipe ISA du LURPA depuis novembre 2010** : Cette équipe se compose aujourd'hui de 6 permanents et accueille en moyenne 7 doctorants et 4 à 5 stagiaires en M2 Recherche.
- Responsable du contrat d'accompagnement de la thèse CIFRE de P.-Y. Chaux [T5]

1.5.1.2. Participation à des groupes de travail

- **Au niveau national**
  - **Groupe INCOS** (Ingénierie de la Commande et de la Supervision) du **GDR MACS** :
    Je participe à ce groupe de travail depuis 1991. Je l'ai connu en tant que Groupe Grafcet de l'Afcet jusqu'en 1997 puis du Club EEA jusqu'en décembre 1999, groupe COSED du club EEA de 2000 à 2002, puis du GDR Automatique de 2002 à 2005.
    J'ai **co-animé** avec Jean-Marc Faure, ce groupe de travail **de septembre 1998 à juin 2003** date de sa fusion avec le groupe ASSF du GRP.
    De février 2008 à décembre 2012, j'ai été membre du comité de pilotage de ce groupe.
  - **Groupe GELA** de **l'EXERA** : L'EXERA est une association des EXploitants d'Equipements de mesure, de Régulation et d'Automatismes, créée en 1974 avec le soutien du Ministère de l'Industrie. Sa principale vocation est d'apporter à ses membres une aide pratique pour l'expression du besoin, sa spécification et le choix de solutions ou de produits (matériels et logiciels). J'ai participé au groupe de travail GELA (Groupe d'Etude des Langages d'Automatisme) d'**avril 1998 à juin 2002** et plus particulièrement à l'élaboration d'une protocole de test de conformité du langage Ladder Diagram de la IEC 61 131-3.
- **Au niveau international**
  - Représentant national au comité technique « TC 3.1 Computers for Control » de la société scientifique IFAC depuis novembre 2011 (Chair : Marek Wegrzyn, University of Zielona Góra).

1.5.1.3. Collaboration avec d'autres équipes de recherche

- **Collaborations au sein de l'ENS Cachan**

Depuis 1998, je participe à différents projets entre les laboratoires de l'ENS Cachan :

- **Projet VULCAIN**, entre le LSV (Laboratoire Spécification et Vérification) et le LURPA, **d'avril 1998 à juin 2001**. L'objet de ce projet, en collaboration avec la société ALCATEL, était de définir une méthode de validation formelle de programmes de contrôle/commande multi-langages.
- **Co-Responsable** du **PPF VSMT** (Vérification de systèmes multi-tâches temps réel) proposé avec le LSV (P. Schnobelen puis B. Bérard) **de juin 2002 à décembre 2005**. L'objet de ce projet était d'intégrer les aspects temporels pour la vérification de propriétés des programmes de contrôle/commande. Ce projet a donné lieu à deux publications communes : la revue [A4] et la conférence [C15].

- **Projet Emoticon**, entre le LSV et le LURPA, de **janvier 2009 à décembre 2010**. L'objet de ce projet était d'étudier les équivalences possibles entre des modèles décrivant le temps et la concurrence. Nous nous sommes focalisés sur le modèle GRAFCET, les automates temporisés et les réseaux de Petri Temporel.
- **Projet CRAFT**, entre le CMLA (Centre de Mathématiques et de Leurs Applications), le LSV et le LURPA, de **janvier 2010 à décembre 2011**. L'objet de ce projet était d'étudier les possibilités d'analyses qualitative et quantitative des arbres de défaillance dynamiques en utilisant les possibilités du model-checking, pour la recherche de séquences d'événements conduisant aux états critiques des systèmes analysés, et la simulation aléatoire pour le calcul de la probabilité d'atteindre ces états critiques. Ce projet a donné lieu à une conférence commune [C24].

- **Collaborations nationales**
  - **Equipe SED, Groupe Auto du Crestic** (Reims) : De 1995 à 2008, j'ai collaboré régulièrement avec les membres permanents de cette équipe. Cette collaboration a débutée avec la mise à leur disposition des différentes versions du logiciel développé au LURPA pour l'analyse de Grafcets. Nous avons également travaillé sur l'élaboration de modèles de parties opératives. Cette collaboration a donné lieu à différentes publications communes [C40], [C5], [C16], [C47].
  - Dans le cadre du **projet ANR TESTEC** (RNTL 2007) de février 2008 à décembre 2011. Ce projet avait pour objectif le Test des Systèmes Temps réel Embarqués Critiques. Il a regroupé deux industriels (EDF Recherche et Développement, Geensoft) et trois autres universitaires (Laboratoire I3S (UPRES-A 6070) - Equipe CeP, INRIA Rennes - Projet VERTECS, LABRI (UMR 5800) - Equipe MVT).
  - **Equipe Sysmo, Laboratoire d'Informatique de l'École Polytechnique** (Palaiseau) : Depuis novembre 2010, je collabore avec le professeur A. Rauzy, autour l'analyse prévisionnelle des risques à l'aide de modèles dynamiques. Cette collaboration a donné lieu à 4 communications communes [C35], [C36], [C59], [C60].

- **Collaborations internationales**
  - **School of Computer Science, University of Birmingham** (Grande-Bretagne) : En 2002, j'ai collaboré avec le professeur M. Kwiatkowska et un de ses doctorants, B. Zoubek, autour de la vérification de programmes écrits en Ladder Diagram à l'aide de l'outil UPAAL. Cette collaboration a donné lieu à une communication commune [C12].
  - **Department of Electrical and Electronic Engineering, University of Cagliari** (Italie) : En 2004, j'ai travaillé avec le professeur A. Giua autour de la synthèse surpervisée de programmes de contrôle-commande. Ce travail a donné lieu à une communication commune [C14], l'encadrement en recherche de 2 stagiaires de cette université en 2006 [S6] et 2009 [S13]. L'un de ces encadrements a fait l'objet d'une communication commune [C20].
  - **Department of Signals and Systems, University of Chalmers** (Suède) : En 2005, j'ai travaillé avec M. Fabian autour de la cohérence des modèles de Process de bas niveau dans le cadre de la synthèse surpervisée.
  - **Dipartimento di Informatica, Universita` del Piemonte Orientale** (Italie) : En 2007, le LURPA a initié une collaboration avec le professeur A. Bobbio dans le cadre du travail de thèse de G. Merle [T3] autour de la modélisation algébrique des arbres de défaillance dynamiques. Cette collaboration a donné lieu à une conférence [C21] et une publication en revue [A3].

- **Equipe SED, Unité Guadalajara du Cinvestav** (Mexique) : En 2010, j'ai travaillé avec le Dr. E. López Mellado dans le cadre d'une expérimentation à Guadalajara de la technique de synthèse algébrique développée au LURPA. Cette collaboration a donné lieu à une communication commune [C30].

1.5.1.4. Organisation de manifestations scientifiques

- **Participation à des comités de programme de conférences nationales**
  - **CETSIS 2013** : 10ème colloque sur l'enseignement des technologies et des sciences de l'information et des systèmes, Caen, 20 au 22 mars 2013
  - **CETSIS 2014** : 11ème colloque sur l'enseignement des technologies et des sciences de l'information et des systèmes, Besançon, 27 au 29 octobre 2014

- **Participation à des comités de programme de conférences internationales**
  - **DCDS 2009** : 2nd IFAC Workshop on Dependable Control of Discrete Systems, Bari (Italie), 10 au 12 juin 2009
  - **VECOS 2010** : 4th International Workshop on Verification and Evaluation of Computer and Communication Systems, Paris, 1 et 2 juin 2010
  - **DCDS 2011** : 3rd International Workshop on Dependable Control of Discrete Systems, Saarbrücken (Allemagne), 15 au 17 juin 2011
  - **VECOS 2011** : 5th International Workshop on Verification and Evaluation of Computer and Communication Systems, Tunis (Tunisie), 15 et 16 septembre 2011
  - **VECOS 2012** : 6th International Workshop on Verification and Evaluation of Computer and Communication Systems, Paris, 27 et 28 août 2012
  - **IWMBSA 2013** : 3rd International Workshop on Model Based Safety Assessment, Versailles, 25 au 27 mars 2013
  - **DCDS 2013** : 4th International Workshop on Dependable Control of Discrete Systems, York (Angleterre), 4 au 6 septembre 2013
  - **VECOS 2013** : 7th International Workshop on Verification and Evaluation of Computer and Communication Systems, Florence (Italie), 21 et 22 novembre 2013
  - **WODES 2014** : 12th IFAC International Workshop on Discrete Event Systems, Cachan, 14 au 16 mai 2014
  - **IMBSA 2014** : 4rd International Symposium on Model Based Safety Assessment, Munich (Allemagne), 27 au 29 octobre 2014

- **Participation à des comités d'organisation de conférences**
  - **MSR 1996** : 1er Congrès « Modélisation des Systèmes Réactifs », Brest, 28 et 29 mars 1996
  - **MSR 1999** : 2nd Congrès « Modélisation des Systèmes Réactifs », Cachan, 24 et 25 mars 1999
  - **JAI 2004** : Journées Automatique et Informatique, Cachan, 11 et 12 mars 2004
  - **DCDS 2007** : 1st IFAC Workshop on Dependable Control of Discrete Systems, Cachan, 13 au 15 juin 2007
  - **WODES 2014** : 12th IFAC International Workshop on Discrete Event Systems, Cachan, 14 au 16 mai 2014

- **Organisation de tracks et sessions invités**
  - Co-organisation, avec L. Grunske (University of Queensland) et Y. Papadopoulos (University of Hull) de la session « Formal fault tree analysis » à **DCDS 2007**.

- **Organisation de tutoriaux**
  - Co-organisation (avec J.-F. Pétin, CRAN) du tutorial « Conception de la commande de

SED sûrs de fonctionnement » pour l'École des JDMACS (Reims, 12-13 juillet 2007), 4 conférenciers.

1.5.2. Critiques scientifiques

- **Jury de thèses**
  - **Benoit Rohée**
    Contribution à la conception d'applications de pilotage des systèmes manufacturiers
    Date de soutenance : 19 décembre 2008
    Jury : C. Moreno (Président), S. Debernard & B. Eynard (Rapporteurs), J.M. Roussel, B. Riéra (Co-directeur de thèse), V. Carré-Ménétrier (Co-directrice de thèse)

- **Articles soumis en revues internationales**
  - **Automatica** en 2012 (1) : Elsevier - IFAC Automatica
  - **CEP** depuis 2005 (6) : Elsevier - IFAC Control Engineering Practice
  - **DEDS** en 1997 (1) et 2007 (1) : Springer - Discrete Event Dynamic Systems
  - **IJPR** en 2004 (1) : Taylor & Francis - International Journal of Production Research
  - **IJRA** en 2009 (1) : Computer science Journals - International Journal of Robotics and Automation
  - **IJSS** en 2010 (1) : Inderscience - International Journal of Services and Standards
  - **JESA** entre 1998 et 2004 (4) : Hermès - Journal Européen des Systèmes Automatisés
  - **JIM** en 2005 (1) et 2007 (1) : Springer - Journal of Intelligent Manufacturing
  - **TCST** en 2012 (1) : IEEE Transactions on Control Systems Technology
  - **TASE** en 2009 (1) : IEEE Transactions on Automation Science and Engineering
  - **TII** depuis 2010 (3) : IEEE Transactions on Industrial Informatics

- **Communications soumis à des conférences (hors conférences en tant qu'IPC)**

En complément des conférences pour lesquelles j'ai été membre du comité de programme, il m'a été donné la possibilité d'évaluer des propositions pour les conférences :

- **ADHS 2003** (1), **ADHS 2009** (1) : IFAC Conference on Analysis and Design of Hybrid Systems
- **ADPM 1998** (3) : Congrès « Automatisation des Processus Mixtes »
- **CASE 2008** (2), **CASE 2013** (1) : IEEE Conference on Automation Science and Engineering
- **CIFA 2010** (3), **CIFA 2012** (1) : Conférence Internationale Francophone d'Automatique
- **ECC 2013** (1) : European Control Conference
- **IFAC 1998** (1), **IFAC 2005** (3), **IFAC 2011** (2), **IFAC 2014** (2) : IFAC World Congress
- **INCOM 2004** (2) : IFAC Symposium on Information Control Problems in Manufacturing
- **JDA 2001** (1), **JDMACS 2009** (1) : Journées doctorales MACS
- **MCPL 2004** (2) : IFAC Conference on Management and Control of Production and Logistics
- **MSR 1999** (2), **MSR 2001** (1), **MSR 2003** (4) : Congrès « Modélisation des Systèmes Réactifs »
- **SAFECOMP 2012** (1) : International Conference on Computer Safety, Reliability and Security
- **WODES 2004** (2), **WODES 2008** (2), **WODES 2010** (2), **WODES 2012** (1) : Workshop on Discrete Event Systems

1.5.3. Responsabilités collectives diverses

1.5.3.1. Participation à l'animation pédagogique au sein de l'ENS Cachan

- **Gestion d'un laboratoire d'enseignement**
  Depuis septembre 2000, je suis responsable du laboratoire d'Automatique du DGM. Cette responsabilité comprend la gestion complète des locaux et des moyens matériels (ordinateurs et systèmes physiques didactiques) pour tous les enseignements d'Automatique (séquentielle ou continue). Cette gestion des moyens matériels intègre la définition des besoins, l'achat d'équipements didactisés ou le développement complet de nouveaux supports, leur rénovation et leur maintenance (réalisée pratiquement par un technicien du DGM).

- **Gestion pédagogique d'un master M2 Recherche**
  Depuis septembre 2005, je suis le responsable pédagogique pour l'ENS Cachan d'un master Recherche co-habilité avec l'Université de Lorraine. Il s'agissait du Master IS en EEAPR « Ingénierie Système en Électronique, Électrotechnique, Automatique, Productique et Réseaux » de 2005 à 2009, puis du Master ISC « Ingénierie Systèmes Complexes », spécialité « Sûreté et Sécurité Actives des Systèmes » de 2009 à 2013 et maintenant du Master ISC « Ingénierie de Systèmes Complexes », spécialité « Systèmes et Technologies de l'Information et la Communication » Parcours « Automatique, Traitement du Signal et Génie Informatique » depuis la rentrée 2013.
  Cette responsabilité comprend la gestion des étudiants inscrits à Cachan, la coordination avec l'Université de Lorraine des enseignements dispensés en visio-conférence, et la gestion des différents examens. Cette responsabilité intègre le maintien en fonctionnement de la solution de visio-conférence (tableaux numériques synchronisés, vidéo...).

- **Participation aux concours nationaux de recrutements d'élèves-ingénieurs**
  L'ENS Cachan recrute principalement au niveau Bac+2 à l'issue des CPGE (Classes Préparatoires aux Grandes Écoles). Pour le DGM, les élèves de CPGE sont recrutés au sein des filières PT et PSI à travers des concours communs à différentes Écoles d'Ingénieurs. J'interviens sur ces concours pour l'épreuve orale « Manipulation de sciences industrielles ».
  - Concours commun pour la filière PT : Membre (depuis 1997) puis Responsable (depuis 2002) d'un des 8 jurys de l'épreuve orale « Manipulation de sciences industrielles ». Cette responsabilité regroupe la sélection des membres de jury (6 personnes), l'élaboration des sujets et la gestion matérielle du parc de manipulations (7 manipulations par jury).
  - Concours « ENS Cachan / Polytechnique » pour la filière PSI : Membre de jury de l'épreuve orale « Manipulation de sciences industrielles » depuis 2003.

1.5.3.2. Participation aux recrutements d'enseignants et d'enseignants-chercheurs

- **Recrutement de professeurs agrégés pour l'Éducation Nationale**
  - Membre du jury de l'**Agrégation Interne de Génie Mécanique** pour les sessions 2009 à 2011.
    Ce concours national de l'Éducation Nationale, qui était présidé par J.-P. Collignon (Inspecteur Général), comporte 2 épreuves écrites et 2 épreuves orales. Je suis intervenu sur l'épreuve écrite « Étude d'un problème d'automatisation » (rédaction du sujet et correction copies en 2009, correction copies en 2011) et sur l'épreuve orale « Travaux pratiques ».

- **Recrutement de Maîtres de Conférence via les Commissions de Spécialistes**
  De 1998 à 2008, j'ai été, chaque année, membre d'au moins deux commissions de spécialistes :
  - Membre nommé de la commission de spécialistes Établissement de l'ISMCM-CESTI de 1998 à 2003,
  - Membre nommé de la commission de spécialistes de l'Université de Reims Champagne Ardennes ($61^{ième}$ et $63^{ième}$ sections) de 1998 à 2008,
  - Membre élu de la commission de spécialistes de l'ENS Cachan ($61^{ième}$ et $63^{ième}$ sections) de 2000 à 2008,
  - Membre nommé de la commission de spécialistes de Nancy Université ($61^{ième}$section) de 2004 à 2008.

- **Recrutement de Maîtres de Conférence via les Comités de Sélection**
  Depuis 2009, j'ai participé à 4 comités de sélection, dont un en tant que président.
  - Membre du comité de sélection du poste 61 0422 à l'URCA (Université de Reims Champagne Ardennes) en 2009,
  - Membre du comité de sélection du poste 61 0422 à l'URCA en 2011,
  - Président du comité de sélection du poste 61 074 à l'ENS Cachan en 2011,
  - Membre du comité de sélection du poste 61 0817 à l'URCA en 2013.

Chapitre 2

Conception sûre des systèmes de contrôle-commande

Ce chapitre présente la partie des recherches que j'ai menées dans le domaine de la conception sûre du système de contrôle-commande à base d'Automates Programmables Industriels.

2.1. Introduction

La conception sûre du système de contrôle-commande d'une installation est une exigence forte à laquelle sont confrontés tous les industriels. C'est particulièrement le cas pour les systèmes critiques comme l'énergie et le transport. Comme le temps accordé au développement des systèmes est de plus en plus court, la garantie de sûreté est aujourd'hui un véritable challenge pour les industriels. Cependant, les industriels ne disposent pas encore de méthodes et d'outils suffisamment performants pour le faire.

Les travaux présentés dans ce chapitre concerne uniquement les systèmes de contrôle-commande réalisés à partir d'Automates Programmables Industriels (APIs). Ces équipements sont largement utilisés dans l'industrie pour leur robustesse et de leur capacité de connexion :

- Un API peut fonctionner sans discontinuité au pied de l'installation qu'il pilote sans être perturbé par l'environnement de cette installation (vibrations, champ magnétique, température...). Il n'est pas rare de trouver des APIs fonctionnant depuis plus de 10 ans.
- Un API peut être connecté à un très grand nombre d'équipements par l'intermédiaire de ses différentes cartes d'entrées/sorties ou de cartes de communication. Pour l'utilisateur, la partie « acquisition des données externes » est totalement transparente.

Si l'installation d'un API au sein d'une installation est technologiquement beaucoup plus simple à mettre en œuvre que pour les autres technologies de commande, le développement du code à implanter est délicat car il nécessite de tenir compte du fonctionnement interne de l'API

(fonctionnement à scrutation cyclique ou périodique, fonctionnement monotâche ou multi-tâche) et des spécificités des langages de programmation normalisés [30] utilisés.

Les travaux que j'ai menés dans le cadre de la conception sûre des systèmes de contrôle-commande ont toujours tenu compte des caractéristiques technologiques des composants matériels utilisés. Je considère que ce point est essentiel et cet attachement à prendre en compte cet aspect peut expliquer à lui seul certaines des spécificités des recherches présentées dans ce chapitre.

Ma contribution à la conception sûre des systèmes de contrôle-commande porte sur les trois problématiques suivantes :
- la *vérification formelle* de programmes de contrôle-commande,
- la *synthèse algébrique* de programmes de contrôle-commande à partir de spécifications informelles,
- le *test de conformité* d'un contrôleur logique vis-à-vis de sa spécification.

D'une manière générale, la *vérification formelle* d'un programme de contrôle-commande consiste à s'assurer que le programme établi par le concepteur (ou généré à partir des spécifications qu'il a écrites) satisfait les exigences attendues. Ce contrôle se fait à posteriori, une fois le programme établi. Pour mener à bien cette opération, il est nécessaire de pouvoir répondre aux questions suivantes :
- Comment établir un modèle du comportement global de l'API lors de l'exécution du programme étudié ?
- Comment exprimer les exigences attendues ?
- Comment vérifier si le modèle du comportement global de l'API lors de l'exécution du programme étudié satisfait les exigences attendues ?

Les travaux autour de la vérification formelle d'un programme de contrôle-commande ont été réalisés collégialement au LURPA dans le cadre de différents projets. Dans ce mémoire, j'ai choisi de ne présenter que les projets dans lesquels mon implication a été la plus significative et d'illustrer les stratégies que nous proposons au travers d'exemples.

La *synthèse algébrique* de programmes de contrôle-commande consiste à obtenir « automatiquement » le programme à partir de l'expression des exigences attendues. La méthode proposée a été spécifiquement développée pour les systèmes physiques disposant d'actionneurs à commande logique comme ceux qu'on peut rencontrer dans le domaine du manufacturier. Cette méthode exploite le fait que le fonctionnement global de ces systèmes peut être obtenu à partir d'une commande locale, parfois élémentaire, de chaque actionneur qui le compose. Cette thématique est à la fois la plus originale de mes travaux et celle à laquelle j'ai consacré le plus de forces. Les résultats présentés seront donc plus détaillés.

Le *test de conformité* d'un contrôleur logique vis-à-vis de sa spécification a été développé dans le cadre de l'ANR TESTEC (TEst des Systèmes Temps réel Embarqués Critiques). Notre partenaire industriel EDF souhaitait disposer d'une méthode garantissant que le comportement d'un API en fonctionnement était strictement conforme à sa spécification. L'API à tester devait être étudié en boite noire, c'est-à-dire, par la seule observation du comportement de ses entrées/sorties. Le comportement spécifié était un comportement séquentiel pour lequel il est demandé de réaliser un test exhaustif (100% du comportement spécifié). Pour réaliser ce type de test sur un équipement réel, un certain nombre de travaux théoriques ont été nécessaires pour établir la séquence de tests qui satisfasse la couverture à 100% du comportement spécifié et qui intègre toutes les contraintes induites par la réalisation matérielle du test sur un API.

2.2. Vérification formelle de programmes de contrôle-commande établis pour des APIs

Les activités de validation des SED ont démarré au LURPA dans le cadre de mes travaux de thèse [M2] qui portaient sur l'étude de spécifications établies en Grafcet conformément à la norme IEC 60 848 [29]. Je me suis ensuite intéressé à la validation formelle de programmes de contrôle-commande établis à l'aide des langages de programmation décrits dans la norme IEC 61 131-3 [30]. La validation formelle de programmes de contrôle-commande a été une activité importante de l'équipe ISA de 1998 à 2008.

L'objectif des travaux était de rendre les techniques de preuves formelles de propriétés, issues de l'informatique théorique, accessibles à l'ingénieur automaticien en montrant leur potentiel pour les systèmes de contrôle-commande réalisés à base d'APIs.

2.2.1. Positionnement scientifique

Dans un cycle de développement logiciel, il n'est pas rare de voir plus de la moitié du temps de développement dédié à la vérification de sa conformité aux spécifications [22]. Le test est encore aujourd'hui une activité largement répandue car elle permet de détecter de nombreuses erreurs tout en restant simple à mettre en œuvre. Cependant, comme cette technique est rarement exhaustive, elle ne peut donc apporter qu'une réponse partielle au problème de la détection d'erreurs. La vérification formelle de logiciels par model-checking ou par theorem-proving se veulent être des techniques complémentaires à celle du test en offrant la possibilité de conduire des analyses exhaustives.

La vérification formelle par model-checking ou par theorem-proving porte sur des modèles. Ces techniques permettent de garantir qu'un modèle M satisfait une propriété φ ($M \models \varphi$) [3]. Pour vérifier automatiquement un système par la méthode du model-checking, il est nécessaire d'en construire une modélisation formelle (M) sous la forme d'un automate (fini ou non) ou d'un réseau d'automates synchronisés. Il faut également énoncer formellement les propriétés à vérifier (φ). Enfin, il faut disposer d'un algorithme capable de dire si le modèle vérifie ou non les propriétés énoncées ($M \models \varphi$). Cet algorithme doit être implémenté dans un outil informatique pour être exploitable.

Aujourd'hui, les outils de model-checking sont nombreux[1] et très variés en raison des fonctionnalités offertes (analyse de code, prise en compte du temps, étude probabiliste...). Une grande partie d'entre-eux sont disponibles gratuitement pour un usage non commercial. Les outils les plus performants proposent également des versions commerciales attestant de leur maturité. Certains outils sont maintenant dédiés à un langage de modélisation ou de programmation donné. D'autres sont largement ouverts.

Lorsque nous avons démarré au LURPA les travaux sur la vérification formelle par model-checking, l'offre logicielle gratuite pour un usage non commercial se limitait à UPPAAL et NuSMV. Les travaux actuellement menés au CERN [16] montrent que ces outils restent aujourd'hui parmi les plus performants.

Nous souhaitions permettre au concepteur du programme d'un API de vérifier si sa commande respecte les propriétés intrinsèques propres à tout système de commande de ce type (absence de blocage, atteignabilité d'états, possibilité de réinitialisation...) ainsi que les propriétés

1. Voir : http://en.wikipedia.org/wiki/List_of_model_checking_tools ou http://anna.fi.muni.cz/yahoda/

extrinsèques requises pour l'application (absence de conflits entre commandes antagonistes, conformité de séquence d'événements...).

Quel que soit la pertinence des résultats obtenus en informatique, il n'en demeurait pas moins vrai que plusieurs verrous devaient être levés afin de rendre ces techniques de vérification accessibles aux ingénieurs automaticiens :

- Le premier verrou était la constitution du modèle M sur lequel la preuve sera réalisée automatiquement par le model-checker. Dans le cas de la vérification formelle de programmes de contrôle-commande établis pour des APIs, deux stratégies ont été envisagées :
  - Le modèle M recherché représente le fonctionnement global de l'API. Il intègre une description de son fonctionnement interne et de tous les éléments qui influent sur la valeur des variables du programme.
  - Le modèle M recherché représente le fonctionnement complet du système étudié. Il intègre une description du comportement de la commande et une description du comportement de la partie opérative.

  Pour pouvoir vérifier l'ensemble des propriétés exigées pour une installation, ces deux familles de modèles sont nécessaires. Une propriété de sûreté doit être vérifiée à l'aide d'un modèle représentant le fonctionnement global de l'API pour que le modèle de la partie opérative ne fausse pas le résultat. Une propriété d'atteignabilité doit être vérifiée à l'aide d'un modèle représentant le fonctionnement complet du système étudié pour tenir compte des limites d'évolution de la partie opérative.
- Le deuxième verrou concernait l'obtention des propriétés formelles φ. L'écriture d'expressions en logique temporelle correspondant aux propriétés à vérifier est une tâche difficilement envisageable dans un contexte industriel. Nous avons montré que l'ajout dans le modèle M d'automates observateurs chargés de détecter le comportement erroné permettait la vérification de propriétés très élaborées sans avoir recours à de complexes formulations en logique temporelle.
- Le troisième verrou résidait dans la capacité à traiter des systèmes de grande taille. L'application sans précaution des techniques de vérification formelle à des modèles conséquents conduit inexorablement à une explosion combinatoire rendant la preuve impossible. Le passage à l'échelle de ces techniques, indispensable pour leur diffusion dans le milieu industriel, nécessite une modélisation pertinente.
  - Le modèle M recherché doit être à la fois précis pour pouvoir distinguer une légère variation dans un programme à vérifier tout en étant suffisamment compact pour limiter sa sensibilité à l'explosion combinatoire. Pour obtenir ce résultat, nous avons cherché à exploiter au mieux les possibilités offertes par les outils UPPAAL et NuSMV.

Lever ces trois verrous a constitué l'objectif scientifique des travaux menés dans cette thématique.

2.2.2. Présentation globale des différents projets suivis

La figure 2.1 regroupe les différentes stratégies que j'ai suivies pour la validation formelle d'un modèle ou d'un programme de commande. J'ai envisagé des approches par model-checking et par theorem-proving.

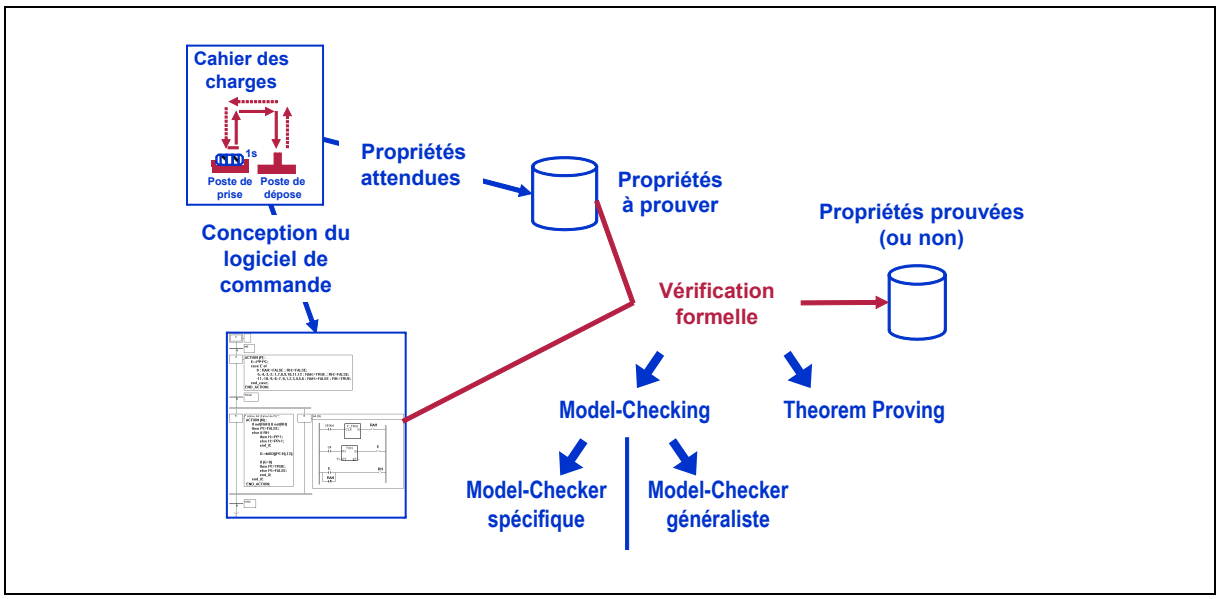


Figure 2.1. Approches développées pour la validation formelle de programmes de contrôle-commande

Les travaux réalisés à l'aide des model-checkers généralistes comme UPPAAL et NuSMV ont été menés avec les autres membres de l'équipe ISA comme ce thème était une activité importante de l'équipe de 1998 à 2008.

De 1998 à 2001, le LURPA était engagé dans le projet VULCAIN (Validation par l'Utilisateur de Logiciels de Commande d'Automatismes INdustriels) avec le LSV (Laboratoire Spécification et Vérification) et Alcatel Recherche (Site de Marcoussis). Ce projet regroupait 4 enseignants-chercheurs permanents, 2 doctorants et un post-doc.

Les langages de programmation étudiés dans le cadre du projet VULCAIN étaient le SFC (Sequential Function Chart), le LADDER et le langage textuel IL (Instruction List). L'apport majeur des travaux réside dans l'approche proposée pour valider par model-checking un programme établi avec ces différents langages. J'ai principalement travaillé sur la validation de modèles SFC [C56] [C6] [R5] [C10].

La collaboration avec le LSV s'est poursuivie, de juin 2002 à décembre 2005, dans le cadre du PPF VSMT (Vérification de Systèmes Multi-tâches Temps réel) dont j'avais la co-responsabilité. Dans le cadre du projet VSMT, nous nous sommes intéressés à la validation de programmes par model-checking à l'aide de l'outil UPPAAL [C15] [A4]. La modélisation proposée intègre une modélisation du comportement cyclique de l'API, de ses primitives temporelles et un modèle élémentaire de son environnement.

En parallèle de ces travaux, je me suis intéressé à la validation de programmes de contrôle-commande à l'aide d'une approche purement algébrique basée sur le principe du theorem-proving. Cette approche exploite le fait que de nombreuses propriétés de sûreté peuvent être prouvées sans avoir à construire l'espace d'état complet des variables du programme API. La vérification des propriétés peut être établie en s'intéressant uniquement à la fonction mathématique qui permet d'établir la valeur de ces variables. Je me suis ainsi particulièrement intéressé à des programmes écrits en LADDER.

Pour ces travaux, je me suis appuyé sur une algèbre spécifique que j'avais développée pour les signaux binaires afin de modéliser le comportement des opérateurs temporels utilisés en SED comme la détection d'un changement d'état [A11] [C1] [C3], la mesure d'un temps d'activité [C7] [C8], et les mémorisations [C11]. J'ai montré qu'une sous-classe de programmes

LADDER pouvait être représentée algébriquement grâce à cette algèbre et qu'il était possible de démontrer le respect de différentes propriétés de sûreté par calcul symbolique réalisé à partir des théorèmes établis dans cette algèbre [C43] [A12]. Une instrumentation à l'aide de l'outil Mathematica a été testée dans le cadre d'un travail de DEA [S2].

Durant cette période, j'ai également collaboré avec le professeur M. Kwiatkowska (School of Computer Science, University of Birmingham) et un de ses doctorants, B. Zoubek. L'objectif était de proposer une technique de traduction de programmes écrits en LADDER vers les automates temporisés proposés par l'outil UPPAAL afin de permettre la vérification formelle de propriétés temporelles. Cette collaboration a donné lieu à une communication commune [C12].

2.2.3. Détails des principaux résultats obtenus

Le principal apport de l'équipe ISA dans le cadre de la vérification formelle de programmes de contrôle-commande à l'aide de model-checkers comme UPPAAL et NuSMV réside dans les techniques proposées pour obtenir la description d'entrée de ces outils, description qui doit représenter le comportement de l'API lors de l'exécution du programme à valider.

Les techniques proposées exploitent les spécificités des deux model-checkers utilisés afin de limiter au mieux la sensibilité des descriptions proposées à l'explosion combinatoire. Les concepts sur lesquels reposent ces techniques vont être principalement présentés à l'aide d'un exemple (Figure 2.2) traité avec le model-checker NuSMV. Certains aspects seront illustrés en s'appuyant sur l'étude de cas présentée dans la publication [A4] donnée en annexe, étude de cas qui a été traitée avec UPPAAL.

Pour utiliser un model-checker comme UPPAAL ou NuSMV, la première activité à réaliser consiste à exprimer le comportement à vérifier à l'aide des primitives du langage d'entrée du model-checker considéré. Il s'agit d'un travail de modélisation demandant une forte expertise pour que les conclusions obtenues sur le modèle M soient exploitables vis-à-vis du comportement à vérifier. Dans notre cas, nous cherchons à établir automatiquement cette modélisation à partir du programme de contrôle-commande à valider. Concrètement, l'objectif de cette activité de modélisation est de proposer un agencement pertinent des primitives du langage d'entrée du model-checker, agencement à partir duquel le model-checker construit automatiquement une structure de Kripke globale. C'est sur cette structure de Kripke globale que sera vérifiée chaque propriété exprimée en logique temporelle. La génération du modèle M doit se faire en tenant compte de la manière suivie par le model-checker pour construire la structure de Kripke globale.

Dans un outil comme NuSMV, la structure de Kripke calculée fait référence aux différentes valuations atteignables d'un ensemble de variables à valeurs discrètes. Pour calculer toutes ces valuations, NuSMV considère que chaque variable déclarée évolue librement et peut prendre à chaque instant toutes les valeurs autorisées. Le comportement d'une variable peut être restreint en précisant la ou les valeurs prises par la variable dans le prochain état en fonction de la valeur des différentes variables pour l'état courant.
Dans un outil comme UPPAAL, la structure de Kripke calculée fait référence aux différentes valuations atteignables pour un réseau d'automates temporisés, d'horloges et de variables discrètes. La valeur de ces variables discrètes est fixée lors du changement d'état d'un des automates temporisés.

Considérons le programme LADDER proposé sur la figure 2.2 comportant 10 réseaux exécutés séquentiellement. L'interprétation de chaque réseau permet à l'API de calculer la valeur courante des variables internes. Une fois le code entièrement exécuté, la dernière valeur calculée pour chacune des sorties est transmise aux cartes de sortie. Pour ce programme, il est demandé

de vérifier si les sorties « L1_pump » et « L2_pump » peuvent être simultanément émises par l'API.

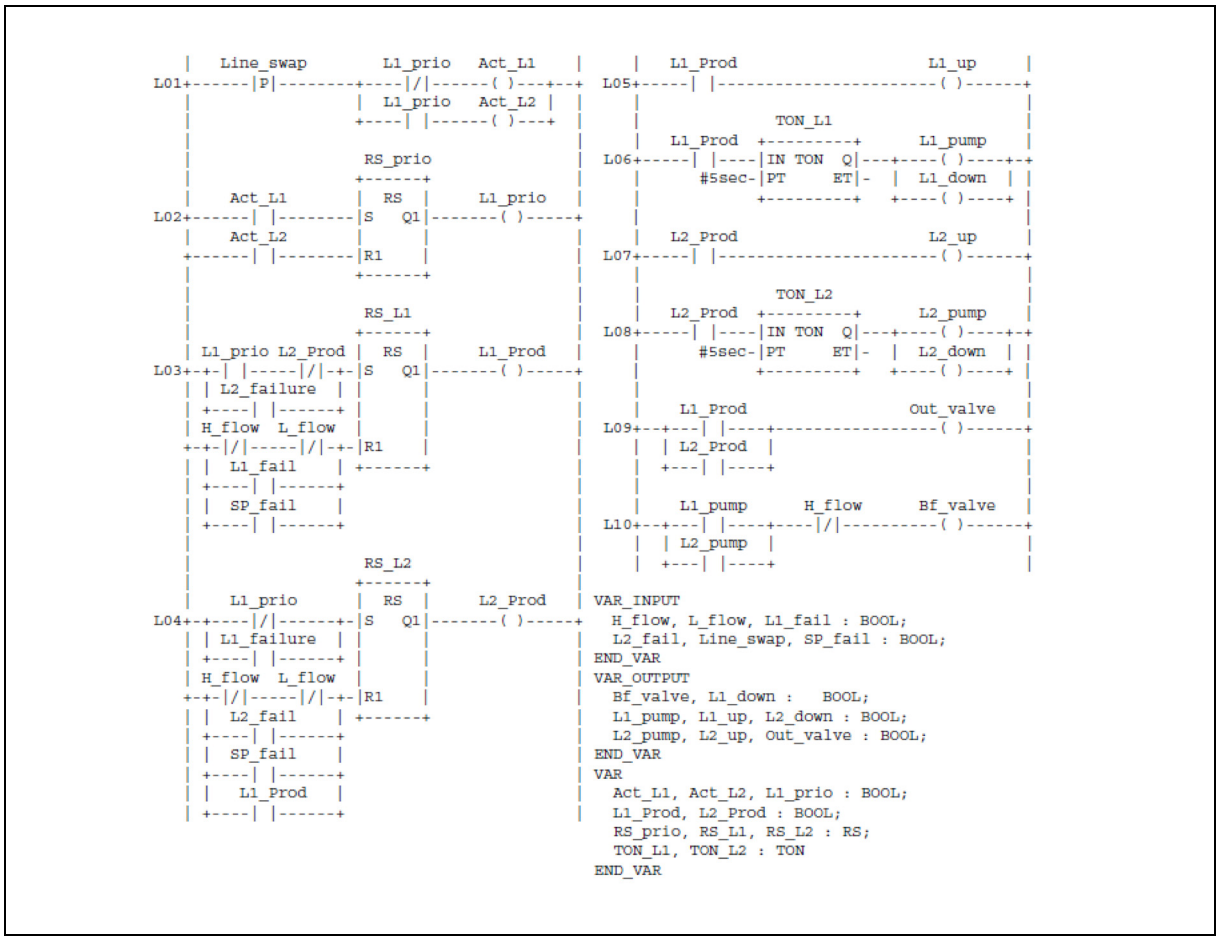


Figure 2.2. Programme LADDER à valider

La modélisation recherchée doit intégrer la description du comportement de toutes les variables du programme, du fonctionnement interne de l'API et de tous les éléments qui influent sur les variables comme les temporisations. Le modèle NuSMV obtenu directement à partir du programme est partiellement représenté sur la figure 2.3. Dans cette représentation, le comportement de l'API est décrit via les variables discrètes « PLC » et « PLC_cp_line ». Les évolutions de la variable « PLC » décrivent les évolutions du moniteur d'exécution de l'API tandis que la variable « PLC_cp_line » permet de spécifier la ligne du programme en cours d'exécution. Ces deux variables sont utilisées pour cadencer les évolutions des entrées de l'API (comme « Line_swap ») et des variables internes du programme (comme « Act_L1 » ou « RS_prio »).

Dans la structure de Kripke construite par le model-checker à partir de ce modèle, nous obtenons :

- Une évolution de la variable discrète « PLC » entre les états : « system », « read », « execute » et « write »,
- Une évolution de la variable discrète « PLC_cp_line » entre les valeurs 0 et 10 lorsque la variable « PLC » est dans l'état « execute »,
- Une évolution libre de chaque variable d'entrée lorsque la variable « PLC » est dans l'état « read » et un maintien de la valeur courante dans les autres cas.
- Une évolution conforme au programme API pour chaque variable calculée lorsque la variable « PLC » est dans l'état « execute » et que la variable « PLC_cp_line » a atteint la

valeur correspondante à la ligne de programme. Il y a un maintien de la valeur courante dans les autres cas.

Le programme API fait référence à 2 blocs temporisations de type TON. Le modèle retenu pour décrire leur comportement est un modèle à seulement 3 états (« inactive », « en_cours », « ecoulee ») qui ne fait pas référence à la valeur exacte de la temporisation. L'évolution du temps correspond au passage de la valeur « en_cours » à la valeur « ecoulee » qui peut librement apparaître dès que la temporisation est activée. Ce modèle abstrait d'une temporisation s'avère suffisant pour prouver de nombreuses propriétés comme celle recherchée.

```
VAR
-- Variables propres au moniteur d execution
  PLC : { system, read, execute, write };
  PLC_cp_line : 0 .. 10 ;

-- Entrees du programme LADDER
  Line_swap : boolean;
...
--
-- Sorties du programme LADDER
  Out_valve : boolean;
  Bf_valve : boolean;
...
--
-- Variables internes du programme LADDER
  Act_L1 : boolean;
  L1_prio : boolean;
--
-- Temporisations du programme LADDER
--
  TON_L1 : { inactive, en_cours, ecoulee };
--
-- Fronts du programme LADDER
--
  Line_swap_old : boolean;


ASSIGN

-- INITIALISATION
-- Variables propres au moniteur d execution
--
  init(PLC) := system;
  init(PLC_cp_line) := 1;
--
-- Sorties du programme LADDER
--
  init(Out_valve) := FALSE;
  init(Bf_valve) := FALSE;
...
--
-- Variables internes du programme LADDER
--
  init(Act_L1) := FALSE;
  init(TON_L1) := inactive;
  init(TON_L2) := inactive;
  init(Line_swap_old) := FALSE;
...

-- EVOLUTION DES VARIABLES
-- Variables propres au moniteur d execution
--

next(PLC) := case
  (PLC=system) : read ;
  (PLC=read) : execute ;
  (PLC=execute) & (PLC_cp_line=10) : write ;
  (PLC=execute) & ! (PLC_cp_line=10) : execute ;
  (PLC=write) : system ;
  esac;

next(PLC_cp_line) := case
  (PLC=execute) & (PLC_cp_line<10) : PLC_cp_line+1
;
  (PLC=execute) & (PLC_cp_line=10) : 0 ;
  (PLC=read) : 1;
  TRUE : PLC_cp_line;
esac;

--
-- Evolutions des entrées
--
next(Line_swap) := case
  (PLC=read) : {FALSE,TRUE};
  TRUE : Line_swap ;
esac;
--
-- Evolutions des variables internes
--

--
-- Ligne 1
--
  next(Act_L1) := case
    (PLC_cp_line=1) & Line_swap & !Line_swap_old &
!L1_prio : TRUE ;
    (PLC_cp_line=1) : FALSE ;
    TRUE : Act_L1 ;
  esac;
--
  next(Act_L2) := case
    (PLC_cp_line=1) & Line_swap & !Line_swap_old &
L1_prio  : TRUE ;
    (PLC_cp_line=1) : FALSE ;
    TRUE : Act_L2 ;
  esac;
--
-- Ligne 2
--
  next(L1_prio) := case
    (PLC_cp_line=2) & Act_L2 : FALSE ;
    (PLC_cp_line=2) & Act_L1  : TRUE ;
    TRUE : L1_prio ;
  esac;
...

--
-- Evolutions des temporisations
--
next(TON_L1) := case
  (PLC_cp_line=6) & !L1_prod : inactive;
  (PLC_cp_line=6) & L1_prod & (TON_L1=inactive) :
en_cours;
  (PLC_cp_line=6) & L1_prod & (TON_L1=en_cours) : {
en_cours, ecoulee } ;
  TRUE : TON_L1 ;
esac;
--
next(L1_pump) := case
  (PLC_cp_line=6) & (next(TON_L1)=ecoulee)  : TRUE ;
  (PLC_cp_line=6) : FALSE ;
  TRUE : L1_pump ;
esac;
...
--

-- Evolution des variables fronts (sur un cycle)
--
next(Line_swap_old) := case
  PLC=system : Line_swap ;
  TRUE : Line_swap_old ;
esac;
```

Figure 2.3. Modèle NuSMV du programme LADDER proposé Figure 2.2

Pour cet exemple, la structure de Kripke construite automatiquement par NuSMV comporte 17 642 états. C'est sur cette structure que sont vérifiées les propriétés exprimées en logique temporelle via la recherche d'un état, d'un chemin ou d'un circuit ayant des caractéristiques données.

Pour vérifier la propriété attendue (les sorties « L1_pump » et « L2_pump » ne doivent jamais être émises simultanément par l'API), il est nécessaire de ne s'intéresser qu'aux états de la structure de Kripke correspondant à la phase d'émission des sorties de l'API. Pour le modèle établi, la formulation en logique temporelle de la propriété est :

AG ((PLC=write) -> !(L1_pump & L2_pump))

L'utilisation de la variable « PLC » permet d'éviter de pointer sur les états de la structure de Kripke correspondants aux états où l'exécution du programme LADDER est incomplète.

Dans le cadre de son travail de DEA [S4], Olivier Cardin a montré qu'il était possible d'obtenir pour le logiciel NuSMV, une structure de Kripke qui décrive le comportement de l'API qu'au seul moment où celui-ci émet ses sorties. La structure de Kripke correspondant à cet exemple comporte seulement 300 états. Cette modélisation exploite une des spécificités du langage d'entrée de NuSMV qui permet de conditionner le futur état d'une variable à partir des futurs états des autres variables. Pour obtenir ce modèle abstrait, le programme à valider subit une phase de réécriture symbolique. Ce principe est à l'origine d'un des mécanismes d'abstraction retenus dans la thèse de Vincent Gourcuff [24].

Dans l'article [A4] fourni en annexe, le lecteur trouvera une étude de cas traitée avec le logiciel UPPAAL. Les propriétés à vérifier sont relatives au déplacement d'un convoyeur. Le modèle d'entrée intègre une description du comportement de la commande (Figure 2.4) et une description du comportement de la partie opérative (Figure 2.5).

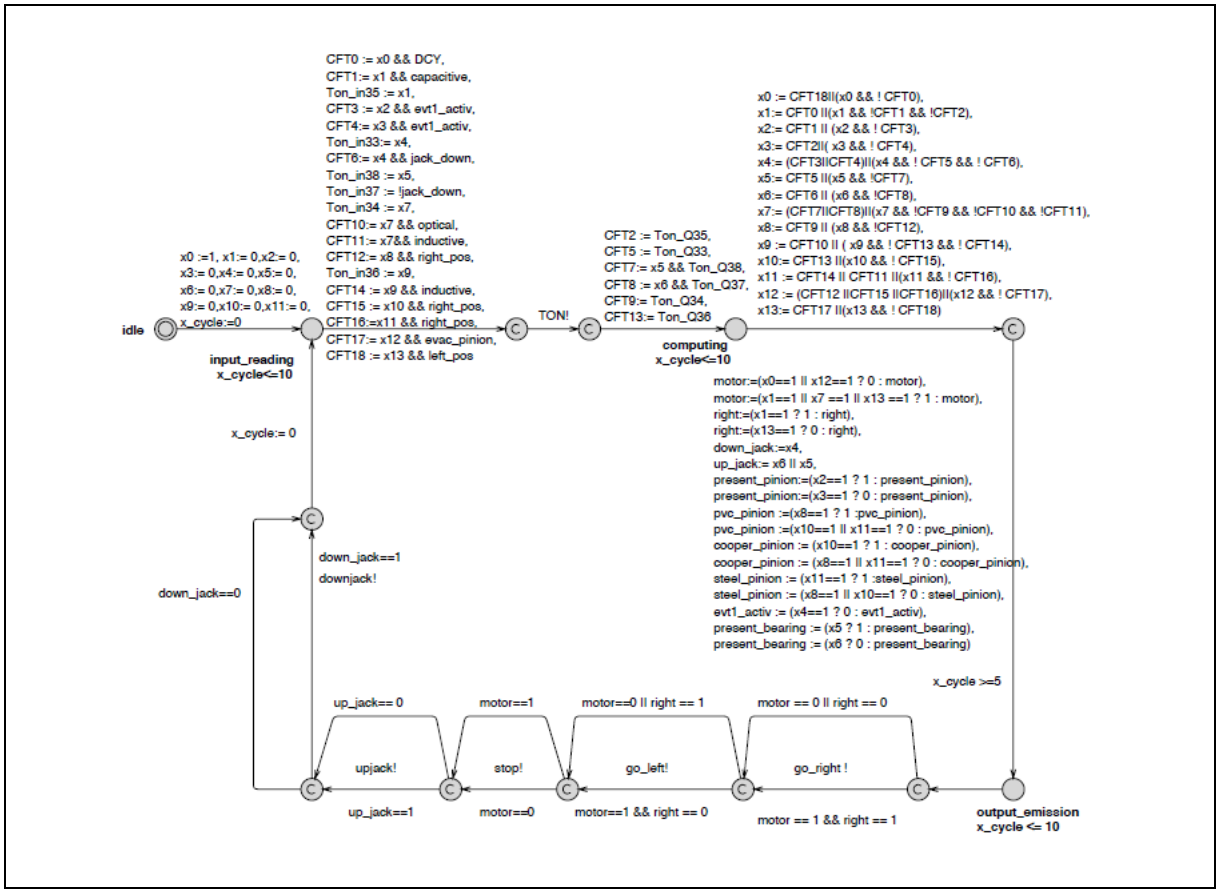


Figure 2.4. Modèle UPPAAL d'un API exécutant un code SFC (extrait)

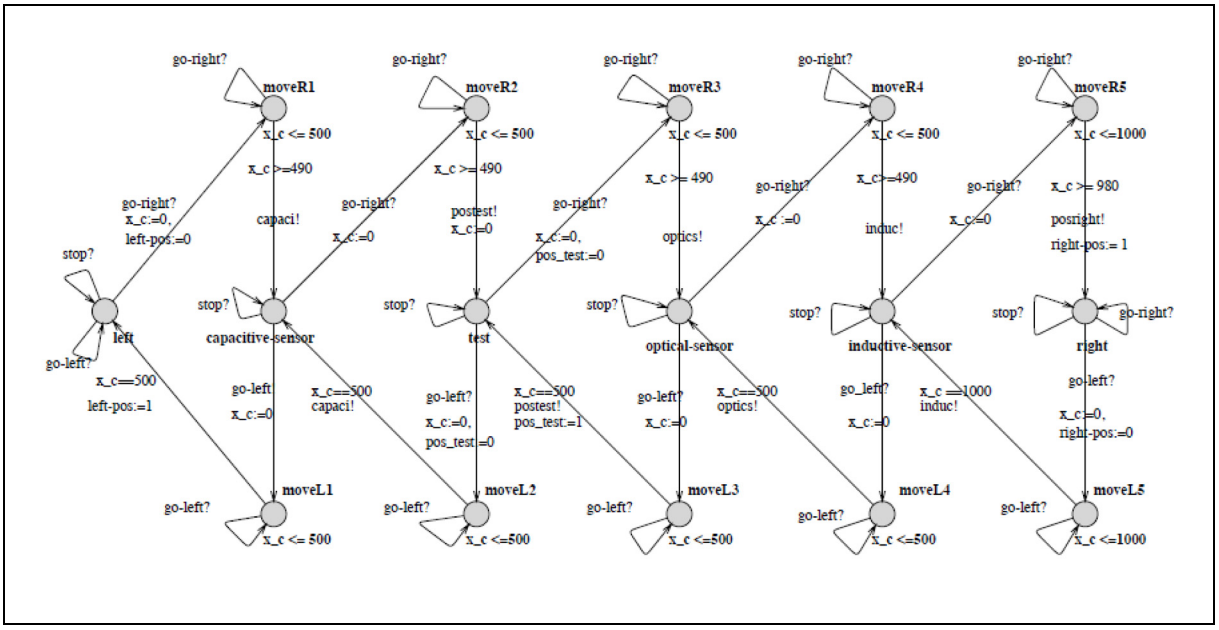


Figure 2.5. Modèle UPPAAL pour le convoyeur

Le modèle UPPAAL (Figure 2.6) proposé pour les blocs temporisations intègre la référence à la valeur exacte de la temporisation. L'évolution du temps, correspond au passage de la valeur « running » à la valeur « Timeout », est ici conditionnée par le compteur « x_Ton ». Contrairement au modèle NuSMV, ce modèle permet de vérifier des propriétés faisant référence à des valeurs explicites du temps.

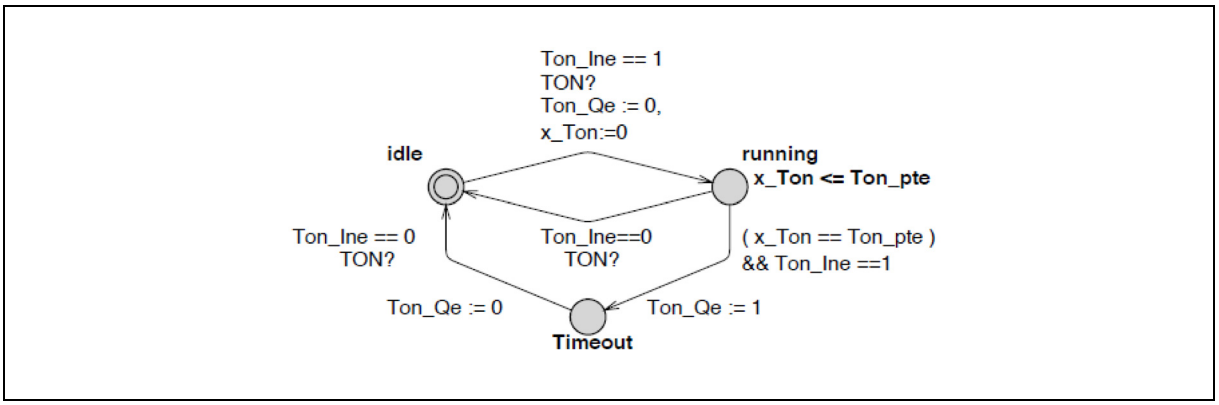


Figure 2.6. Modèle UPPAAL d'une temporisation

Pour la synchronisation des différents automates proposés, il a été utilisé le mécanisme de broadcast (émission : c! et réception : c?) proposé par UPPAAL ainsi que des variables discrètes. Nous avons eu recours à un automate observateur (Figure 2.7) pour exprimer la propriété à vérifier : « La commande doit réagir en moins de 5 ms pour arrêter le convoyeur lors de son passage au poste de test. »

Cette modélisation est représentative des possibilités offertes par le logiciel UPPAAL. Cependant, contrairement à NuSMV, un tel modèle ne peut pas être obtenu automatiquement.

Pour chacune de ces modélisations, nous avons cherché à tirer profit des capacités du langage d'entrée des outils UPPAAL et NuSMV. Pour NuSMV, nous avons fait évoluer progressivement nos modélisations afin de minimiser la structure de Kripke générée. Pour UPPAAL, nous avons privilégié la lisibilité du modèle puisqu'il intègre encore une partie construite manuellement.

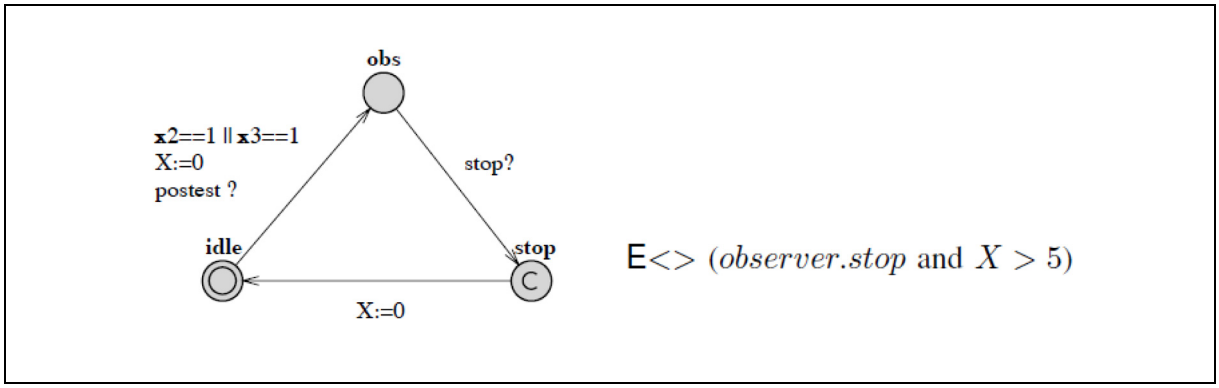


Figure 2.7. Vérification d'une propriété à l'aide d'un automate observateur

2.2.4. Conclusions

La vérification de programmes de contrôle-commande établis pour des APIs est une activité de recherche que l'équipe ISA a fortement réduite ces dernières années car nous avons considéré que les résultats obtenus par la communauté scientifique étaient suffisamment matures pour être transférés dans le milieu industriel. En 2007, à partir des résultats obtenus dans le cadre de la thèse de Vincent Gourcuff [24], la société Alstom a développé au sein de son outil CONTROCAD un module « Vérification formelle » utilisant le model-checker NuSMV. L'outil CONTROCAD est utilisé dans le groupe Alstom pour le développement des systèmes de contrôle-commande des systèmes de production d'énergie (hydraulique ou gaz).

La vérification formelle par model-checking d'un programme de contrôle commande reste encore aujourd'hui hors de la portée de la grande majorité des utilisateurs en raison de l'absence d'outils pour le faire concrètement et de l'expertise que cette activité demande.

Pour tirer le meilleur parti de la vérification formelle dans le cadre de la conception de programmes de contrôle-commande, il est nécessaire que cette activité soit directement intégrée dans le processus de conception afin que la phase de validation ne soit pas vécue par le concepteur du programme comme une remise en cause de son travail mais comme une technique d'assistance à l'écriture de son code. Les méthodes de conception de programmes basées sur l'assemblage de modules logiciels paramétrables sont celles pour lesquelles la vérification formelle peut apporter rapidement des résultats concluants. Ces méthodes sont généralement bien acceptées par les concepteurs car elles permettent de développer une application sans avoir à tout reprogrammer. La vérification formelle de tels programmes peut se faire par niveaux successifs en partant des blocs élémentaires. Dans une telle approche, il peut être envisagé que seuls les modules validés formellement soient réutilisables. Une fois validé, un module peut donner lieu à un modèle abstrait exploitable pour valider les modules qui l'utilisent.

L'obtention d'un programme de contrôle-commande qui puisse être validé formellement nécessite donc que son processus de conception soit parfaitement structuré. Cependant, seules certaines sociétés disposent aujourd'hui d'un tel processus de conception.

La vérification formelle par model-checking d'un programme de contrôle commande n'est plus hors de la portée des utilisateurs pour des raisons théoriques mais essentiellement méthodologiques. De nombreux travaux sont encore nécessaires pour permettre son utilisation.

2.3. Synthèse algébrique de programmes de contrôle-commande à partir de spécifications informelles

Cette activité de recherche est celle à laquelle j'ai consacré le plus de forces. Son objectif est d'obtenir directement le programme de contrôle-commande à partir de spécifications formalisées sous forme algébrique. Les principaux résultats mathématiques nécessaires à sa formalisation ont été obtenus dans le cadre des thèses d'Antonio Médina [T1] et de Yann Hietter [T2].

La conception du programme de contrôle-commande d'une installation automatisée est une tâche difficile qui repose encore aujourd'hui sur la seule expertise des équipes de conception [35]. Pour limiter les effets désastreux des erreurs de programmation, le monde industriel a toujours cherché à fiabiliser son processus de conception en se dotant de méthodologies très strictes pour le développement de ses installations. On peut citer par exemple les recommandations CNOMO[1] définies par les constructeurs automobiles français.

Pour les systèmes critiques, les normes actuelles, comme la norme *IEC 61508 : Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* [31], recommandent l'utilisation de méthodes formelles. Ces techniques ont cependant du mal à pénétrer le secteur industriel, en raison de la forte expertise qu'elles demandent. Les industriels se limitent principalement à l'utilisation de techniques de vérification et abordent très peu les méthodes de synthèse.

2.3.1. Positionnement scientifique

Dans le cadre de l'automatique des SED, la technique de synthèse actuellement la plus étudiée est la « Supervisory Control Theory » (SCT) proposée par Ramadge and Wonham en 1987 [45]. Depuis plus de 25 ans, la recherche académique explore ce cadre mathématique afin de disposer d'un cadre formel pour la synthèse de modèles SED. Si les résultats théoriques obtenus sont remarquables, il convient de souligner que le transfert vers le monde industriel, et plus particulièrement celui du contrôle-commande, est très limité, voire exploratoire seulement.

La « Supervisory Control Theory » est un cadre mathématique permettant de construire un superviseur permettant de contrôler un SED labellisé Procédé afin de le contraindre à satisfaire un ensemble de spécifications comportementales données (Figure 2.8).

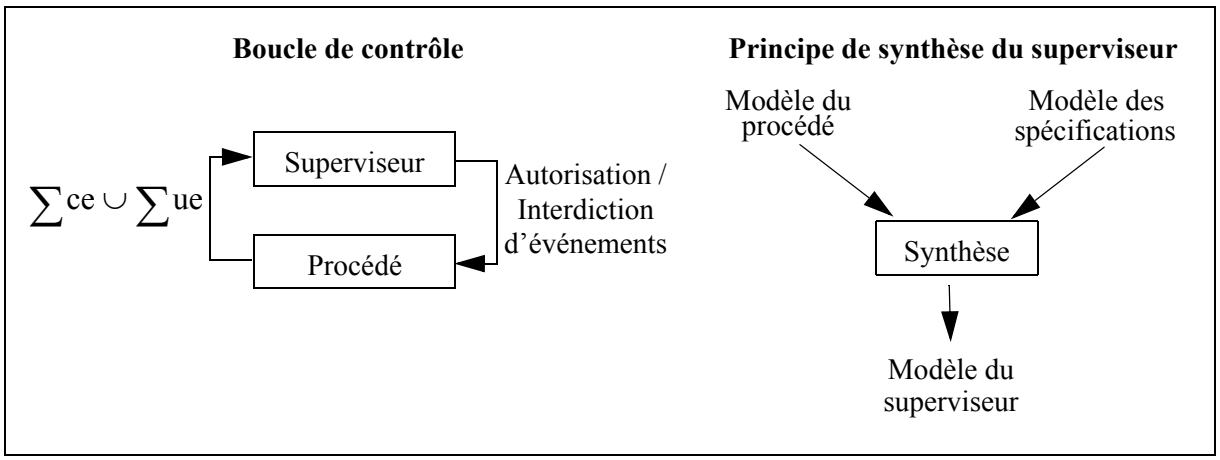


Figure 2.8. Principe de fonctionnement de la SCT

Le procédé est modélisé par un automate fini qui représente tout les états possibles que peut atteindre le SED étudié à contrôler. Les événements produits par le procédé sont scindés en 2

1. Des informations complémentaires sont disponibles sur : http://www.cnomo.org/

catégories : $\sum$ce : les événements contrôlables et $\sum$ue : les événements incontrôlables. Le rôle du superviseur est d'empêcher l'apparition de certains événements contrôlables afin de maintenir le procédé dans la partie de son comportement compatible avec les spécifications données. L'action exercée par le superviseur sur le procédé est une restriction de son comportement et non un contrôle complet puisse que le superviseur ne peut pas forcer l'occurrence d'un événement même contrôlable. Le superviseur agit comme un filtre vis-à-vis du procédé.

Les algorithmes de base de la SCT permettent d'obtenir le superviseur le plus permissif (c'est-à-dire, le moins contraignant vis-à-vis du procédé) à partir de la donnée du modèle du procédé et du modèle des spécifications. Ces algorithmes comporte les phases suivantes :

- Composition synchrone des automates correspondant aux modèles du procédé et des spécifications,
- Détermination des états défendus : un état défendu est tel qu'il existe un événement non contrôlable admis dans le modèle du procédé, mais interdit dans la spécification,
- Détermination des états faiblement défendus : un état faiblement défendu est tel qu'il existe une séquence d'événements incontrôlables dans le produit synchrone procédé × spécification qui conduit à un état défendu,
- Élimination de tous les états défendus et faiblement défendus. Le résultat obtenu correspond au superviseur.

L'utilisation de la SCT pour obtenir un programme de contrôle-commande a fait l'objet de différents travaux en France [25] comme à l'étranger [2] [21]. Dans sa thèse [26], David Gouyon précise clairement quels sont les différents problèmes qui doivent être résolus pour que la SCT soit opérationnelle et les pistes envisagées par les différentes équipes de recherche. Nous y retrouvons :

- La maitrise de l'explosion combinatoire lors la composition synchrone des automates,
- Le passage du superviseur au contrôleur : par construction, le superviseur obtenu grâce à la SCT représente le comportement maximal permissif pour le système étudié. Cependant, ce comportement ne correspond pas au fonctionnement d'un contrôleur dont le rôle est de piloter l'installation de manière déterministe. Le comportement recherché pour le contrôleur n'est qu'un sous-ensemble du comportement du superviseur qu'il est nécessaire d'extraire.
- La génération du code à implanter au sein de l'équipement de contrôle-commande.

Pour évaluer de manière concrète les possibilités offertes par la SCT pour le contrôle-commande d'une partie opérative, j'ai tenu à expérimenter cette approche lors de 2 études de cas pour lequel nous disposions du système physique à commander. Il s'agissait :

- d'un préhenseur pneumatique comportant 3 actionneurs : ce préhenseur permet le déplacement d'une pièce depuis un poste de prise vers un poste de dépose. Le mouvement doit se faire selon un cycle en « U » afin d'éviter toutes collisions.
- d'un portail automatique de parking entraîné par un moteur électrique.

Ces travaux ont été initialement menés avec Alessandro Giua [C14] et poursuivis dans le cadre de l'encadrement d'un de ses étudiants lors d'un travail de DEA [S6] qui s'est concrétisé par une publication [C20]. Les exemples avaient été choisis pour que les modèles générés ne soient pas triviaux tout en restant lisibles pour pouvoir les analyser.

Lors de ces expérimentations, Il est apparu clairement que la SCT n'était pas utilisable pour synthétiser le contrôleur d'une partie opérative en raison de la nécessité :

- de manipuler les signaux échangés entre l'API et la partie opérative et non des événements théoriques dénués de sens physique :
  - Le choix des événements contrôlables et incontrôlables est imposé par le câblage de l'API pilotant le système physique à commander : nous avons 2 événements incontrôlables par entrée logique (passage de 0 à 1 et passage de 1 à 0) et 2 événements contrôlables par sortie.
- d'intégrer les algorithmes de traitement des API et leur incidence sur le traitement des informations échangées avec la partie opérative :
  - Lors du même cycle de scrutation, un API peut faire varier simultanément plusieurs de ses sorties. Il peut également accepter plusieurs variations successives de ses entrées sans faire évoluer ses sorties.
- de l'extrême difficulté à établir le modèle du procédé représentatif de la partie opérative à contrôler :
  - Pour que le système de contrôle-commande puisse contrôler de manière sûre la partie opérative, il faut que le modèle du procédé soit extrêmement détaillé afin qu'aucune évolution possible de la partie opérative ne soit oubliée. Pour ces 2 systèmes, les premiers modèles du procédé étaient non représentatifs du système à contrôler car trop simplistes. Comme certaines évolutions du système réel n'avaient pas été envisagées dans le modèle du procédé, la commande qui aurait été obtenue n'aurait pas pu réagir à ces évolutions.
- des nombreuses difficultés rencontrées pour établir un modèle satisfaisant pour chaque spécification comportementale :
  - Dans le cas de la synthèse d'un programme de contrôle commande, il ne suffit pas d'interdire l'apparition d'événements contrôlables mais il faut également pouvoir forcer leur apparition (arrêt en fin course d'un mobile entraîné par un moteur électrique par exemple).
  - Certaines spécifications comportementales ne peuvent pas être écrites indépendamment des unes des autres. il a été parfois nécessaire de regrouper certaines d'entre-elles au sein du même automate.
- d'extraire automatiquement le comportement attendu pour le contrôleur du modèle du superviseur :
  - Une méthode systématique basée sur la recherche du plus court chemin entre états marqués a été proposée dans [C20]. Cependant, cette technique n'est opérationnelle que si les spécifications et le modèle du procédé sont écrits en tenant compte de ce critère.

Le tableau 2.1 extrait de [C20] regroupent les données techniques relatives à ces 2 expérimentations. Le superviseur a été obtenu à l'aide de l'outil Supremica. L'exploitation du superviseur pour obtenir le code à implanter (Code en Structured Text généré pour TSX Premium) a fait l'objet du développement d'un module spécifique écrit en Python. Il convient de remarquer, que même sur des exemples élémentaires, cette méthode nécessite un important travail de modélisation pour obtenir les modèles pour le procédé et les spécifications.

	Portail automatique	Préhenseur pneumatique
Modèle du procédé	11 automates (4 pour les entrées, 2 pour les sorties, 1 pour l'API, 4 pour la partie opérative)	17 automates (6 pour les entrées, 3 pour les sorties, 1 pour l'API, 7 pour la partie opérative)
Automate final du procédé	481 états, 1 330 transitions	62 721 états, 237 890 transitions.
Spécifications	11 automates	9 automates
Aut. final des spécifications	276 états, 1 215 transitions	416 états, 2 616 transitions
Superviseur obtenu	368 états, 646 transitions	173 états, 251 transitions
Superviseur après réduction	110 états, 194 transitions	108 états, 160 transitions
Contrôleur généré	15 états, 40 transitions	30 états, 58 transitions

Tableau 2.1: Données relatives à 2 études de cas traitées à l'aide de la SCT

Les résultats obtenus pour ces deux études de cas montrent que la SCT n'est pas exploitable pour obtenir le code à implanter pour contrôler directement une partie opérative. Lorsque la SCT est utilisée pour synthétiser des modèles de plus haut niveau (comme l'étude des modes de marche d'un système), la plupart des limites qui viennent d'être listées disparaissent. La SCT doit donc pouvoir donner des résultats nettement plus intéressants.

Face à ce constat, je me suis fortement impliqué dans la recherche et la mise au point d'une autre technique de synthèse.

2.3.2. Objectifs des travaux

Cette méthode de synthèse algébrique permet d'obtenir la commande des systèmes physiques dont l'état global peut être majoritairement reconstruit à partir des informations fournies par les détecteurs présents sur le système. Pour ces systèmes, en s'appuyant sur la connaissance de l'état de la partie opérative et de l'état des produits manipulés, le fonctionnement global peut être obtenu à partir d'une commande locale, parfois élémentaire, de chaque actionneur qui compose le système.

La logique de commande recherchée n'est plus basée sur une description des enchaînements des opérations à réaliser mais sur la donnée explicite des conditions nécessaires et suffisantes pour que chaque opération ait lieu. Cette logique de commande a été très longtemps exploitée dans certaines méthodologies industrielles en raison de la robustesse des programmes qu'elle permettait d'obtenir (recommandations CNOMO).

La méthode de synthèse proposée repose sur ce principe de conception. Nous souhaitons extraire le programme de contrôle-commande à partir de la formulation explicite des différentes exigences retenues pour le système. Ces exigences correspondent à :

- des attentes fonctionnelles relatives à la commande du système, comme la séquence des opérations à réaliser,
- des règles de sécurité liées aux modes de marche,
- des règles de sécurité ou de programmation liées à la technologie des composants commandés,
- des règles de sécurité liées aux positionnements des composants commandés.

Cependant, nous déchargeons entièrement le concepteur du difficile travail de mise au point des lois de commande locales qui satisfont l'ensemble des exigences exprimées. Celles-ci sont obtenues automatiquement lors de la résolution d'un système d'équations (Figure 2.9).

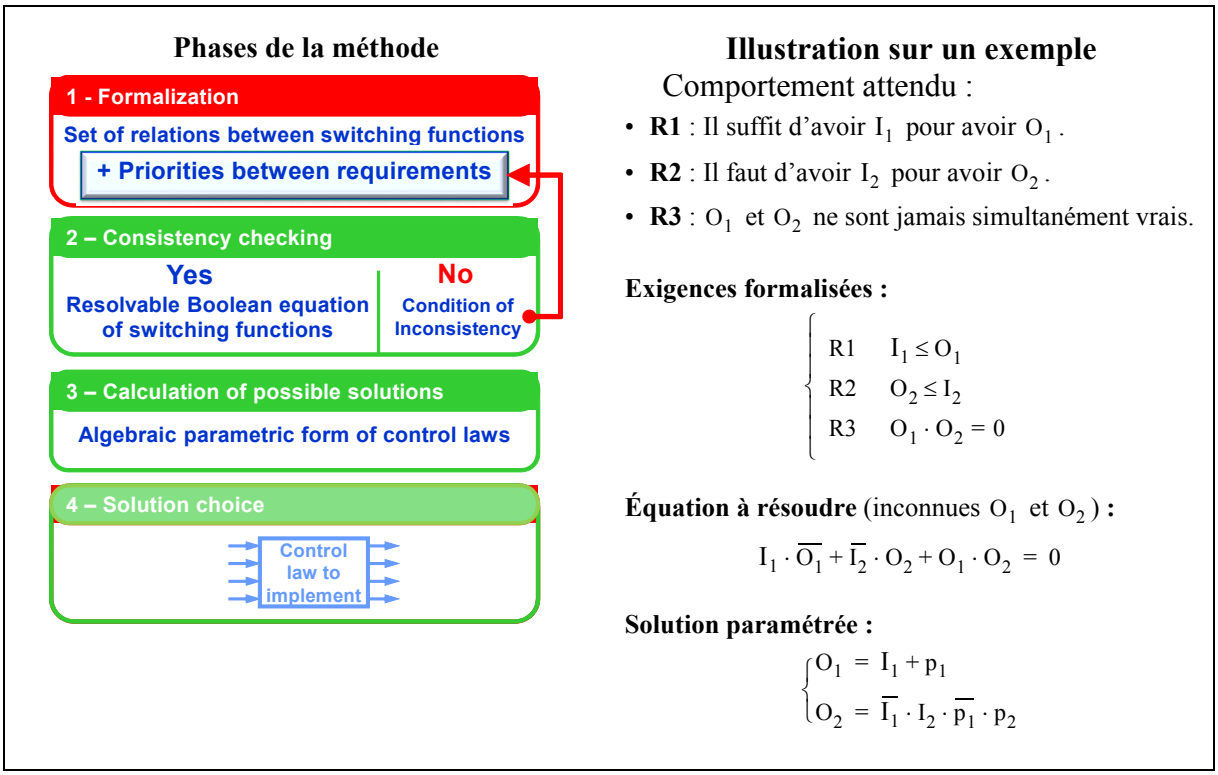
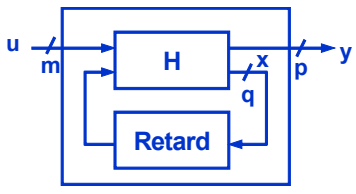


Figure 2.9. Principe de la synthèse algébrique

Sur le plan mathématique, les éléments manipulés lors de cette résolution sont des fonctions booléennes (Switching functions) de dimension n :

$$f : B^n \rightarrow B \text{ avec } B = \{{}_b0, {}_b1\}$$

Pour travailler algébriquement avec des fonctions booléennes afin d'obtenir la commande d'un système logique, nous sommes repartis d'une description proposée par Huffman en 1954 [28]. La figure 2.10 présente le modèle SED que nous utilisons. Ce modèle comporte m entrées logiques u_i, p sorties logiques y_j, et q variables internes logiques x_l introduites pour exprimer le comportement séquentiel du système. Le temps est discrétisé : [k] représente l'instant présent et [k-1] l'instant précédent. Dans cette modélisation, le comportement est décrit par la définition de (p+q) fonctions booléennes de (m+q) variables.



$$\begin{cases} x_l[k] = H_l(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_q[k-1]) \\ y_j[k] = H_{q+j}(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_q[k-1]) \end{cases}$$

$(2^{2^{(m+q)}})^{(p+q)}$ lois de commande exprimables

Figure 2.10. Modèle retenu pour la loi de commande d'un système logique

Cette description est très expressive puisque le modèle de la figure 2.10 peut modéliser un système comportant 2^q états différents. Bien que construit sur seulement (p+q) fonctions booléennes, il permet de décrire $(2^{2^{(m+q)}})^{(p+q)}$ comportements différents.

Le modèle proposé par Huffman en 1954 a rapidement été abandonné au profit des modèles proposées par Mealy [41] ou Moore [42] en raison de la difficulté à établir les (p+q) fonctions booléennes de (m+q) variables. Dans notre approche, ces (p+q) fonctions booléennes sont définies automatiquement par résolution d'un système d'équations qui représente les différentes exigences retenues pour la commande.

Cette opération est aujourd'hui possible car nous savons résoudre dans l'algèbre de Boole des fonctions booléennes tout système d'équations entre ces fonctions quelque soit le nombre d'inconnues, la forme et le nombre d'équations [T2]. Grâce à la relation d'ordre partiel qui existe dans cette algèbre, nous avons montré qu'il est possible d'exprimer formellement des conditions nécessaires ou suffisantes entre ces fonctions. C'est cette caractéristique qui permet de formaliser simplement les attentes exprimées dans un cahier des charges. Une fois ces attentes formalisées, nous sommes en mesure de déterminer, par calcul symbolique, tous les éléments qui les satisfont et de les décrire à l'aide d'une forme paramétrée.

Les derniers développements mathématiques ont été obtenus pour faciliter le travail de formalisation des exigences attendues pour la commande. Ils permettent d'obtenir automatiquement les solutions qui satisfont des critères d'optimisation donnés [S12] [C32]. Il est également possible de hiérarchiser les exigences exprimées pour la commande afin de tenir compte de leur priorités relatives pour l'établissement de la logique de commande [C33].

Les différents essais [A2] [C18] [C19] [C32] [C33] réalisés sur les systèmes physiques auxquels nous avons accès, ont montré que les programmes de contrôle-commande obtenus avec cette approche étaient à la fois très compacts et conduisaient à un fonctionnement sûr de l'installation.

2.3.3. Détails des principaux résultats obtenus

Les principaux résultats obtenus sont présentés en deux temps. Les résultats mathématiques sont donnés globalement puis seront listés les résultats méthodologiques propres à la synthèse de lois de commande.

Le lecteur pourra trouver dans [T2] [C32] [C33] et [A8] la démonstration des résultats mathématiques. Pour éviter toute confusion due à certaines définitions, les bases mathématiques sur lesquelles s'appuient ces travaux font l'objet d'un rappel succinct.

2.3.3.1. Bases mathématiques

Les définitions et théorèmes présentés dans cette section sont majoritairement extraits de [14] et de [27]. Ils permettent d'introduire les notions et les notations utilisées dans ces travaux.

Définition 1 Algèbre de Boole[1]

Soit B un ensemble non vide d'éléments contenant deux éléments particuliers 0 (élément Zéro) et 1 (élément Un) sur lequel sont définies deux lois de composition internes binaires notées $(+, .)$ et une loi de composition interne unaire notée $(\bar{})$.

1. Définition 15.5 de [27]

$(B, +, ., \bar{}, 0, 1)$ est une **algèbre de Boole** si les neuf axiomes suivants sont satisfaits pour tout élément x, y et z de B :

$x + y = y + x$	$x \cdot y = y \cdot x$	Commutativité
$x \cdot (y + z) = (x \cdot y) + (x \cdot z)$	$x + (y \cdot z) = (x + y) \cdot (x + z)$	Distributivité
$x + 0 = x$	$x \cdot 1 = x$	Élément neutre
$x + \bar{x} = 1$	$x \cdot \bar{x} = 0$	Élt complémentaire
$0 \neq 1$		

Il existe de nombreuses algèbres de Boole. La plus connue est l'algèbre de Boole à 2 éléments : $(\{{}_b0, {}_b1\}, \vee, \wedge, \neg, {}_b0, {}_b1)$.

Définition 2 Formule booléenne[1]
Une formule booléenne (ou expression booléenne) sur B est une formule qui représente une combinaison des éléments de B par les lois de composition $+$, $.$ et $\bar{}$.

Par construction, toute formule booléenne sur B représente un et un seul élément de B. Deux expressions sont équivalentes si elles représentent le même élément de B. Toute formule booléenne sur B peut être reformulée sous une forme canonique grâce au théorème 1.

Théorème 1 Expansion booléenne d'une formule booléenne
Soient $(\alpha_1, \ldots, \alpha_n)$ n éléments de $B - \{0, 1\}$. toute formule booléenne $F(\alpha_1, \ldots, \alpha_n)$ peut être développée de la façon suivante :

$$F(\alpha_1, \ldots, \alpha_n) = F_0(\alpha_2, \ldots, \alpha_n) \cdot \overline{\alpha_1} + F_1(\alpha_2, \ldots, \alpha_n) \cdot \alpha_1$$

où $F_0(\alpha_2, \ldots, \alpha_n)$ et $F_1(\alpha_2, \ldots, \alpha_n)$ sont des formules booléennes de $(\alpha_2, \ldots, \alpha_n)$ uniquement.

$$\begin{cases} F_0(\alpha_2, \ldots, \alpha_n) = F(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 0} = F(0, \alpha_2, \ldots, \alpha_n) \\ F_1(\alpha_2, \ldots, \alpha_n) = F(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 1} = F(1, \alpha_2, \ldots, \alpha_n) \end{cases}$$

La relation égalité n'est pas la seule relation qui existe dans une algèbre de Boole. Il existe également une relation d'ordre partiel.

Définition 3 Relation Inclusion[2]
Soient $x, y \in B$, définissons $x \leq y$ si et seulement si $x \cdot y = x$

Pour l'algèbre des classes $(2^S, \cup, \cap, \bar{}, \varnothing, S)$, cette relation correspond à $\subseteq$ car $a \subseteq b \Leftrightarrow a \cap b = b$.

1. A partir de la section 3.6 de [14]
2. Définition 15.6 de [27]

Théorème 2 Réduction d'un ensemble de relations[1]

Tout ensemble de relations entre des éléments $(\alpha_1, \ldots, \alpha_n)$ de B peut s'exprimer sous la forme d'une unique relation $F(\alpha_1, \ldots, \alpha_n) = 0$.

L'algèbre de Boole que nous utilisons est l'algèbre de Boole des fonctions booléennes.

Définition 4 Fonction booléenne de n variables

Un fonction booléenne de n variables est une fonction de la forme

$$\begin{array}{l} f : \quad B^n \to B \\ \quad ({}_b b_1, \ldots, {}_b b_n) \to f({}_b b_1, \ldots, {}_b b_n) \end{array} \quad \text{avec } B = \{{}_b 0, {}_b 1\}$$

Soit $F_n(B)$ l'ensemble des fonctions booléennes de n variables. Par construction, $F_n(B)$ comporte 2^{2^n} éléments et comporte $(n + 2)$ éléments particuliers :

- 2 fonctions constantes (0,1) définies par :

$$\begin{array}{ll} 0 : \quad B^n \to B & 1 : \quad B^n \to B \\ \quad ({}_b b_1, \ldots, {}_b b_n) \to {}_b 0 & \quad ({}_b b_1, \ldots, {}_b b_n) \to {}_b 1 \end{array}$$

- n fonctions projections (f^i_{Proj}) définies par :

$$\begin{array}{l} f^i_{Proj} : \quad B^n \to B \\ \quad ({}_b b_1, \ldots, {}_b b_n) \to {}_b b_i \end{array}$$

$F_n(B)$ peut être muni de trois lois de compositions internes (2 lois binaires, 1 loi unaire).

$$\begin{array}{lll} \text{Loi} + : F_n(B)^2 \to F_n(B) & \text{Loi} . : F_n(B)^2 \to F_n(B) & \text{Loi } \bar{} : F_n(B) \to F_n(B) \\ \quad (f, g) \to f + g & \quad (f, g) \to f \cdot g & \quad f \to \bar{f} \end{array}$$

avec $\forall ({}_b b_1, \ldots, {}_b b_n) \in B^n$,

$$(f + g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \vee g({}_b b_1, \ldots, {}_b b_n)$$

$$(f \cdot g)({}_b b_1, \ldots, {}_b b_n) = f({}_b b_1, \ldots, {}_b b_n) \wedge g({}_b b_1, \ldots, {}_b b_n)$$

$$\bar{f}({}_b b_1, \ldots, {}_b b_n) = \neg f({}_b b_1, \ldots, {}_b b_n)$$

Théorème 3 Algèbre de Boole des fonctions booléennes de n variables

$(F_n(B), +, ., \bar{}, 0, 1)$ est une algèbre de Boole.

C'est dans cette algèbre de Boole que les résultats mathématiques nécessaires à cette méthode de synthèse ont été démontrés. Ils s'appuient sur le fait que tout élément de $F_n(B)$ peut être exprimé sous la forme d'une composition des $(n + 2)$ éléments particuliers.

1. Théorème 5.3.1 de [14]

2.3.3.2. Résultats mathématiques

F. M. Brown consacre une partie de son livre « Boolean reasoning » [14] à la résolution d'équations dans une algèbre Boole. on peut trouver dans ce livre :

- le théorème 6 (Solution paramétrée d'une équation à une inconnue),
- le principe de démonstration retenu pour le théorème 7 (Solution paramétrée d'une équation à k inconnues). Cependant, dans ce livre, il n'est pas proposé de solution paramétrée pour exprimer les solutions.

Notre méthode de synthèse repose entièrement sur la capacité a résoudre tout système d'équations entre des fonctions booléennes quelques soient le nombre d'inconnues, le nombre d'équation et la forme de ces équations. Cette résolution est possible grâce aux théorèmes suivants :

- Théorème 4 : Réduction d'un ensemble de relations entre des fonctions booléennes de n variables,
- Théorème 5 : Forme canonique d'une équation booléenne,
- Théorème 7 : Solution paramétrée d'une équation à k inconnues.

Le théorème 8 (Solution d'une équation en tenant compte d'hypothèses sur les fonctions de projections) a été démontré pour être capable de travailler sur un sous-ensemble de $F_n(B)$ lorsqu'aucune solution sur $F_n(B)$ existe.

Le théorème 9 (Solution d'une équation en tenant compte de priorités entre les exigences) a été démontré pour être capable de hiérarchiser les exigences d'un problème donné.

Dans le cadre de ces travaux, cette capacité à établir la solution paramétrée pour une équation à k inconnues (Théorème 7) a permis de mettre en place des techniques de recherche de solutions suivant des critères d'optimisation donnés (Théorème 11). Cette recherche de solutions optimales s'appuie sur l'ordre partiel existant entre des fonctions booléennes et sur l'existence d'un extremum pour une formule booléenne (Théorème 10) lorsque cette formule fait référence à des variables libres $F_n(B)$.

L'expression de ces différents théorèmes nécessite l'introduction de quelques notations spécifiques. Considérons $(F_n(B), +, ., \bar{}, 0, 1)$ l'algèbre de Boole des fonctions booléennes de n variables :

- Soient $(f^1_{Proj}, \ldots, f^n_{Proj})$ les n fonctions projections de $F_n(B)$. Notons Proj le vecteur $(f^1_{Proj}, \ldots, f^n_{Proj})$.
- Soient $(x_1, \ldots, x_k)$ les k éléments de $F_n(B)$ considérés comme inconnus. Notons X_k le vecteur $(x_1, \ldots, x_k)$.
- Pour $x \in F_n(B)$ et $a \in \{0, 1\}$, x^a est défini par : $x^0 = \bar{x}$ et $x^1 = x$
- Pour $X_k \in (F_n(B))^k$ et $A_k \in \{0, 1\}^k$, ${X_k}^{A_k}$ est défini par :

$$ {X_k}^{A_k} = \prod_{i=1}^{i=k} x_i^{a_i} = x_1^{a_1} \cdot \ldots \cdot x_k^{a_k} $$

Théorème 4 Réduction d'un ensemble de relations entre des fonctions booléennes de n variables

Tout ensemble de relations entre les éléments $(f^1_{Proj}, \ldots, f^n_{Proj})$ et $(x_1, \ldots, x_k)$ de $F_n(B)$ peut s'exprimer sous la forme d'une unique relation $F(X_k, Proj) = 0$.

Théorème 5 Forme canonique d'une équation booléenne

Toute équation booléenne $Eq(X_k, Proj) = 0$ peut être exprimée sous la forme canonique suivante :

$$\sum_{A_k \in \{0,1\}^k} Eq(A_k, Proj) \cdot X_k^{A_k} = 0$$

Théorème 6 Solution paramétrée d'une équation à une inconnue

L'équation booléenne $Eq(x, Proj) = 0$ sur $F_n(B)$ qui a pour forme canonique

$$Eq(0, Proj) \cdot \bar{x} + Eq(1, Proj) \cdot x = 0$$

admet des solutions si et seulement si :

$$Eq(0, Proj) \cdot Eq(1, Proj) = 0$$

Dans ce cas, la forme générale des solutions est

$x = Eq(0, Proj) + p \cdot \overline{Eq(1, Proj)}$ où p est un élément quelconque de $F_n(B)$.

Théorème 7 Solution paramétrée d'une équation à k inconnues

L'équation booléenne $Eq_0(X_k, Proj) = 0$ sur $F_n(B)$ qui a pour forme canonique

$$\sum_{A_k \in \{0,1\}^k} Eq_0(A_k, Proj) \cdot X_k^{A_k} = 0$$

admet des solutions si et seulement si :

$$\prod_{A_k \in \{0,1\}^k} Eq_0(A_k, Proj) = 0$$

Dans ce cas, la forme générale du k-uplet solution $(S(x_1), \ldots, S(x_k))$ est

$$S(x_i) = \prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(0, A_{k-i}, Proj) + p_i \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(1, A_{k-i}, Proj)}$$

où

- $Eq_i(x_{i+1}, \ldots, x_k, Proj) = Eq_{i-1}(x_i, x_{i+1}, \ldots, x_k, Proj)\big|_{x_i \leftarrow S(x_i)}$
- p_i est un élément quelconque de $F_n(B)$.

Théorème 8 Solution d'une équation en tenant compte d'hypothèses sur les fonctions de projections

Le problème

$$\left[\begin{array}{l}\text{Equation à résoudre}\\ \qquad \mathrm{Eq}_0(\mathrm{X_k}, \mathrm{Proj}) = 0\\ \text{Hypothèses}\\ \qquad H(\mathrm{Proj}) = 0\end{array}\right.$$

admet les même solutions que l'équation suivante :

$$\mathrm{Eq}_0(\mathrm{X_k}, \mathrm{Proj}) \leq H(\mathrm{Proj})$$

Théorème 9 Solution d'une équation en tenant compte de priorités entre les exigences

Le problème

$$\left[\begin{array}{l}\text{Equation à résoudre}\\ \qquad \left\{\begin{array}{l}\mathrm{HR}\ F_H(\mathrm{X_k}, \mathrm{Proj}) = 0\\ \mathrm{LR}\ F_L(\mathrm{X_k}, \mathrm{Proj}) = 0\\ \mathrm{OR}\ F_O(\mathrm{X_k}, \mathrm{Proj}) = 0\end{array}\right.\\ \text{Priorités entre les exigences}\\ \qquad \mathrm{HR} \gg \mathrm{LR}\end{array}\right.$$

où

- $F_H(\mathrm{X_k}, \mathrm{Proj}) = 0$ est l'expression des exigences ayant la priorité maximale (HR),
- $F_L(\mathrm{X_k}, \mathrm{Proj}) = 0$ est l'expression des exigences ayant la priorité minimale (LR),
- $F_O(\mathrm{X_k}, \mathrm{Proj}) = 0$ est l'expression des autres exigences (OR),
- $\mathrm{HR} \gg \mathrm{LR}$ est la règle de priorités entre les exigences incohérentes entre-elles,

admet les même solutions que le problème suivant :

$$\left\{\begin{array}{l}F_H(\mathrm{X_k}, \mathrm{Proj}) = 0\\ F_L(\mathrm{X_k}, \mathrm{Proj}) \leq I(\mathrm{Proj})\\ F_O(\mathrm{X_k}, \mathrm{Proj}) = 0\end{array}\right.$$

où $I(\mathrm{Proj})$ est la condition d'incohérence entre les exigences HR et LR :

$$I(\mathrm{Proj}) = \prod_{\mathrm{A_k} \in \{0,1\}^k} (F_H(\mathrm{A_k}, \mathrm{Proj}) + F_L(\mathrm{A_k}, \mathrm{Proj}))$$

Théorème 10 Extremum d'une formule booléenne

Toute formule booléenne $F(P_k, \text{Proj})$ où P_k représente un vecteur de fonctions booléennes libres de $F_n(B)$ admet un élément maximum et un élément minimum :

$$\text{Max}_{P_k \in (F_n(B))^k}(F(P_k, \text{Proj})) = \prod_{A_k \in \{0,1\}^k} F(A_k, \text{Proj})$$

$$\text{Min}_{P_k \in (F_n(B))^k}(F(P_k, \text{Proj})) = \sum_{A_k \in \{0,1\}^k} F(A_k, \text{Proj})$$

Théorème 11 Solution optimale d'une équation selon un critère donné

Le problème

$$\begin{cases} \text{Equation à résoudre} \\ \quad \text{Eq}(X_k, \text{Proj}) = 0 \\ \text{Critère d optimisation} \\ \quad \text{Extremum de } F_C(X_k, \text{Proj}) \end{cases}$$

admet les même solutions que le problème suivant :

$$\begin{cases} \text{Eq}(X_k, \text{Proj}) = 0 \\ F_C(X_k, \text{Proj}) = \text{Ext}_{\{X_k \mid \text{Eq}_0(X_k, \text{Proj}) = 0\}}(F_C(X_k, \text{Proj})) \end{cases}$$

Ces différents théorèmes permettent de faire tous les calculs nécessaires à cette approche de synthèse algébrique.

2.3.3.3. Résultats méthodologiques propres à la synthèse de lois de commande

La synthèse des lois de commande d'un système logique séquentiel comportant m entrées logiques u_i, p sorties logiques y_j, et q variables internes logiques x_l repose sur l'association d'une fonction booléenne de dimension $(m + p)$ à chacune des variables utilisées selon le processus suivant :

- la valeur à l'instant $[k]$ de l'entrée u_i est décrite par la fonction booléenne :
$$U_i(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_p[k-1])$$
- la valeur à l'instant $[k]$ de la sortie y_j est décrite par la fonction booléenne :
$$Y_j(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_p[k-1])$$
- la valeur à l'instant $[k]$ de la variable interne x_l est décrite par la fonction booléenne :
$$X_l(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_p[k-1])$$
- la valeur à l'instant $[k\text{-}1]$ de la variable interne x_l est décrite par la fonction booléenne :
$${}_pX_l(u_1[k], \ldots, u_m[k], x_1[k-1], \ldots, x_p[k-1])$$

Les fonctions booléennes U_i et ${}_pX_l$ sont supposées connues. Elles correspondent aux fonctions projections de l'algèbre de Boole des fonctions booléennes de dimension $(m + p)$. Les fonctions booléennes Y_j et X_l correspondent aux inconnues du problème que nous souhaitons

exprimer sous la forme d'une formule booléenne des seules fonctions booléennes U_i et ${}_pX_l$ en tenant compte des différentes relations liant les fonctions booléennes U_i, ${}_pX_l$, Y_j et X_l.

En complément des résultats mathématiques déjà présentés, il a été réalisé :
- une bibliothèque d'assertions type pour l'expression des spécifications élémentaires,
- une mise en œuvre logicielle pour dispenser le concepteur des calculs fastidieux.

Résultats complémentaires : Bibliothèque d'assertions type

Une bibliothèque d'assertions type a été mise au point pour faciliter la synthèse de lois de commande. Cette bibliothèque permet la formalisation sous forme de relations entre des fonctions booléennes de spécifications intemporelles ou faisant référence à des contraintes de changement d'état pour les variables internes comme :
- Il faut avoir « a » pour avoir « b » : $B \leq A$
- Il suffit d'avoir « a » pour avoir « b » : $A \leq B$
- « a » et « b » ne peuvent pas avoir lieu simultanément : $A \cdot B = 0$
- Lorsque « c » est vrai, il faut avoir « a » pour avoir « b » : $(C \cdot B) \leq A$
- Lorsque « c » est vrai, il suffit d'avoir « a » pour avoir « b » : $(C \cdot A) \leq B$
- Il faut avoir « a » pour avoir la mise à 1 de « x_1 » : $({}_p\overline{X_l} \cdot X_l) \leq A$
- Il suffit d'avoir « a » pour avoir la mise à 1 de « x_1 » :[1] $({}_p\overline{X_l} \cdot A) \leq ({}_p\overline{X_l} \cdot X_l)$
- Il faut avoir « a » pour avoir la mise à 0 de « x_1 » : $({}_pX_l \cdot \overline{X_l}) \leq A$
- Il suffit d'avoir « a » pour avoir la mise à 0 de « x_1 » : $({}_pX_l \cdot A) \leq ({}_pX_l \cdot \overline{X_l})$

Sur le plan mathématique, toute spécification exprimable sous la forme de relations entre des formules booléennes des fonctions booléennes U_i, ${}_pX_l$, Y_j et X_l peut être prise en compte par cette méthode de synthèse. Il convient de souligner que la formulation mathématique d'une spécification n'est pas unique. Cependant, comme il est toujours possible de vérifier par calcul symbolique si deux formulations sont mathématiquement équivalentes, le concepteur peut a tout moment contrôler la compatibilité des formulations qu'il propose.

La relation mathématique $\leq$ est certainement la pierre angulaire de cette approche car elle permet d'exprimer des conditions nécessaires et des conditions suffisantes.

Résultats complémentaires : Mise en œuvre logicielle

Les théorèmes précédents permettent aujourd'hui d'obtenir la forme paramétrée d'un k-uplet solutions de $F_n(B)$ satisfaisant un ensemble de relations entre n fonctions booléennes supposées connues (f^i_{Proj}) et k fonctions booléennes supposées inconnues (x_i) en tenant compte d'un

1. La formulation mathématique proposée correspond à la spécification en langage naturelle « Lorsque la variable x_1 est à 0, il suffit d'avoir « a » pour avoir la mise à 1 de « x_1 ». La première partie de la sentence est implicite en langage naturel mais doit être intégrée dans la formulation mathématique pour que cette relation puisse donner lieu à des solutions.

ou plusieurs critères d'optimisation. L'obtention de cette forme paramétrée n'est pas triviale car elle nécessite de nombreux calculs symboliques impossibles à réalisés manuellement.

Le logiciel « BESS »[1] a été développé en Python pour décharger complètement l'analyste des différents calculs nécessaires à l'obtention de la forme paramétrée des solutions. Les opérations réalisées par ce logiciel sont :

- la lecture du problème (à partir de données fournies dans un fichier texte) intégrant la vérification de la syntaxe et de la cohérence des données,
- la détermination de l'équation à résoudre comprenant :
  - la prise en compte des hypothèses,
  - la prise en compte des priorités,
  - la prise en compte des critères d'optimisation,
- la résolution de l'équation,
- l'export du résultat au format texte de la forme paramétrée des solutions.

Depuis septembre 2010, ce logiciel est utilisé en enseignement principalement par les étudiants inscrits au module « Algebraic approaches for analysis and synthesis of Discrete Event Systems » du master ISC. Il leur permet de traiter en autonomie les différentes études de cas support du cours.

Parmi les différentes études de cas traitées avec le logiciel « BESS », certaines d'entre-elles ont donné lieu à un rapport détaillé disponible depuis le site WWW du laboratoire à partir de la page dédiée : http://www.lurpa.ens-cachan.fr/-171294.kjsp

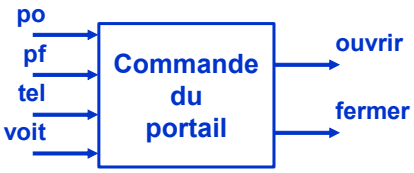
Les éléments donnés sur la figure 2.11 correspondent à la synthèse de la loi de commande d'un portail automatique en exploitant les différentes possibilités offertes par cette méthode[2]. La loi de commande obtenue est présentée sur la figure 2.12 et peut être implémentée en utilisant seulement 3 réseaux LADDER.

Dans le logiciel « BESS », tous les calculs symboliques se font à l'aide de BDD. Lors du traitement de cet exemple, il a été construit un unique BDD comportant 186 nœuds pour coder simultanément l'ensemble des données du problème, l'équation à résoudre et les solutions calculées. L'utilisation d'un BDD partagé pour coder les différentes formules booléennes manipulées a permis une implémentation efficace des différentes opérations de calcul symbolique nécessaires à cette approche.

Le lecteur pourra se reporter à [A8], publication jointe en annexe, pour plus détails sur cette méthode de synthèse. Une étude de cas différente y est présentée.

1. BESS : Boolean Equations System Solver
2. Pour cet exemple, il est également possible d'obtenir la même loi de commande sans recourir à des critères d'optimisation.

Entrées/sorties du contrôleur



Forme générale de la loi de commande recherchée

$$\begin{cases} \text{ouvrir}[k] = \text{Ouvrir}(\text{po}[k], \text{pf}[k], \text{tel}[k], \text{voit}[k], \text{ouvrir}[k-1], \text{fermer}[k-1]) \\ \text{fermer}[k] = \text{Fermer}(\text{po}[k], \text{pf}[k], \text{tel}[k], \text{voit}[k], \text{ouvrir}[k-1], \text{fermer}[k-1]) \end{cases}$$

Modèle algébrique

Fonctions booléennes de dimension 6

- Fonctions booléennes considérées comme connues :

$$\text{Po, Pf, Tel, Voit, } {}_{p}\text{Ouvrir, } {}_{p}\text{Fermer}$$

- Fonctions booléennes considérées comme inconnues :

$$\text{Ouvrir, Fermer}$$

Spécifications prises en compte

- **F1** : Il suffit que la télécommande soit activée pour que le portail s'ouvre.

$$\text{Tel} \leq \text{Ouvrir}$$

- **F2** : Pour arrêter l'ouverture du portail, il faut qu'il soit entièrement ouvert (toute ouverture doit être complète).

$$({}_{p}\text{Ouvrir} \cdot \overline{\text{Ouvrir}}) \leq \text{Po}$$

- **T1** : Le moteur ne doit pas être commandé simultanément dans les 2 sens.

$$\text{Ouvrir} \cdot \text{Fermer} = 0$$

- **T2** : L'ouverture du portail est impossible lorsqu'il est entièrement ouvert.

$$\text{Ouvrir} \cdot \text{Po} = 0$$

- **T3** : La fermeture du portail est impossible lorsqu'il est entièrement fermé.

$$\text{Fermer} \cdot \text{Pf} = 0$$

- **S1** : La fermeture du portail est impossible lorsqu'une voiture est détectée.

$$\text{Fermer} \cdot \text{Voit} = 0$$

- **S2** : La fermeture du portail est impossible lorsque la télécommande est activée.

$$\text{Fermer} \cdot \text{Tel} = 0$$

- **S3** : Lorsque le portail est fermé, pour démarrer son ouverture, il faut une demande à l'aide de la télécommande.

$$\text{Pf} \cdot ({}_{p}\overline{\text{Ouvrir}} \cdot \text{Ouvrir}) \leq \text{Tel}$$

Priorité prise en compte

- Priorité des contraintes technologiques sur les attentes fonctionnelles

$$\text{T2} \gg \text{F1}$$

Critère d'optimisation

- La loi de commande recherchée doit maximiser la fermeture du portail en absence de voiture et son ouverture lorsqu'une voiture est détectée.

$$\text{Max}(\text{Voit} \cdot \text{Ouvrir} + \overline{\text{Voit}} \cdot \text{Fermer})$$

Figure 2.11. Synthèse de la commande d'un portail automatique

Solution obtenue

$$\begin{cases} \text{Ouvrir} = \overline{\text{Po}} \cdot (\text{Tel} + \overline{\text{Pf}} \cdot \text{Voit} + {}_{\text{p}}\text{Ouvrir}) \\ \text{Fermer} = \overline{\text{Pf}} \cdot \overline{\text{Tel}} \cdot \overline{\text{Voit}} \cdot (\text{Po} + {}_{\text{p}}\overline{\text{Ouvrir}}) \end{cases}$$

Code LADDER Correspondant

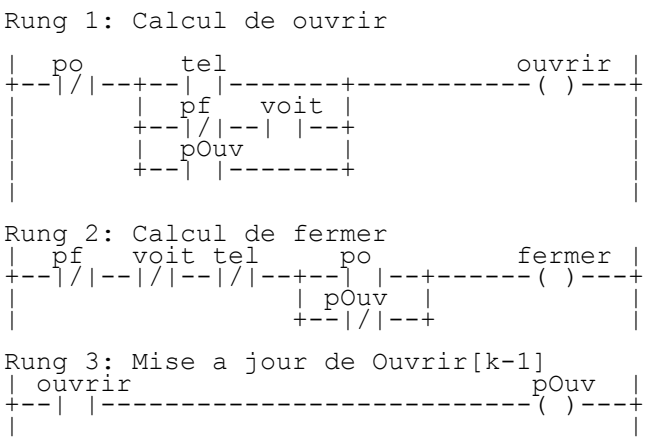


Figure 2.12. Loi de commande synthétisée

2.3.4. Conclusions

La méthode de synthèse proposée a été spécifiquement développée pour obtenir la loi de commande d'un système logique séquentiel. Cette méthode repose sur la résolution d'un système d'équations entre des fonctions booléennes.

Sur le plan mathématique, les résultats obtenus permettent de résoudre tout système d'équations entre des fonctions booléennes quelques soient le nombre d'inconnues, le nombre de relations et la formulation de celles-ci. La forme paramétrée proposée pour les solutions permet d'exprimer l'ensemble des solutions du problème et non une unique solution comme cela est fait par les approches de satisfaisabilité booléenne (SAT).

Le principal intérêt de la forme paramétrée est de permettre la recherche d'une solution optimale selon un ou plusieurs critères d'optimisation. Ces critères d'optimisation peuvent être traités simultanément ou séquentiellement [C32]. Dans cette méthode, la seule contrainte pour ces critères d'optimisation est qu'ils soient exprimables sous la forme d'une formule booléenne à minimiser ou à maximiser, formule booléenne des différentes fonctions booléennes utilisées dans le problème.

D'un point de vue purement mathématique, les résultats obtenus sont applicables à toutes les structures d'algèbre de Boole ayant pour seules lois de composition les lois ET, OU et NON. Ils peuvent donc être appliqués directement à la théorie des ensembles ou pour résoudre des problèmes de satisfaisabilité booléenne.

On pourrait envisager d'étudier l'extension de ces résultats mathématiques pour des structures d'algèbre de Boole ayant des lois de composition non exprimables à l'aide des seules lois ET, OU et NON. L'algèbre « I », que j'ai proposée dès 1992 [A11] pour modéliser des signaux binaires dépendants du temps, en est un exemple. C'est dans le cadre de cette algèbre que les premiers essais de synthèse ont été réalisés [A1] [A2]. Cependant, la définition retenue pour les opérateurs modélisant le temps (fronts, temporisations) ne permettent pas aujourd'hui de rame-

ner tout système de relations entre des signaux binaires à une unique équation disposant d'une forme canonique. Sans cette forme canonique, la résolution d'équations ne peut se faire qu'au cas par cas. Les avancées possibles s'avèrent donc limitées.

Sur le plan méthodologique, et en ne considérant que la synthèse de lois de commande pour des systèmes logiques séquentiels, il est évident que des travaux complémentaires doivent être développés pour rendre cette approche réellement opérationnelle et/ou en identifier clairement les limites. A ce stade de développement de la méthode, plusieurs actions complémentaires peuvent être menées :

- Traitement de nouvelles études de cas par des acteurs externes afin d'analyser le processus d'acquisition de la méthode,
- Enrichissement de la bibliothèque d'assertions type pour tenir compte des technologies des équipements commandés, de règles de fonctionnement métier...
- Étude des possibilités d'importation des spécifications depuis des modèles de haut niveau afin de faciliter l'expression des séquences de fonctionnement,
- Étude d'un processus de synthèse par partie afin d'offrir aux spécificateurs la possibilité de travailler par niveau hiérarchique (organisation des modes de marche, détails des différents modes)...

Sur le plan du logiciel, l'outil actuel permet la résolution du système d'équations proposé par un utilisateur et l'expression de la forme paramétrée des solutions. Pour faciliter l'utilisation de cette méthode dans le cadre de la synthèse de lois de commande, il serait intéressant d'optimiser les algorithmes utilisés et d'ajouter de nouvelles fonctionnalités comme la génération automatique de code pour les API, la génération de documentation, l'expression des solutions avec mise en facteur des termes communs...

2.4. Test de conformité d'un contrôleur logique vis-à-vis de sa spécification

Les travaux relatifs au test de conformité d'un contrôleur logique vis-à-vis de sa spécification ont été obtenus dans le cadre de la thèse de Julien Provost [T4]. Cette thèse a été réalisée dans le cadre de l'ANR TESTEC (TEst des Systèmes Temps réel Embarqués Critiques) dont le LURPA était le porteur de projet. Ce projet s'est déroulé de février 2008 à décembre 2011. Il regroupait deux industriels (EDF Recherche et Développement, Geensoft) et trois autres universitaires (Laboratoire I3S (UPRES-A 6070) - Equipe CeP, INRIA Rennes - Projet VERTECS, LABRI (UMR 5800) - Equipe MVT).
Ce projet s'est appuyé sur une précédente collaboration liant l'équipe STEP d'EDF Recherche et le LURPA autour du portage d'un automatisme à relais de centrale nucléaire vers un API [R9] [R10] [R11]. Cette collaboration portait sur le test de fonctions combinatoires et séquentielles spécifiées à l'aide de Diagrammes Fonctionnels Logiques (DFL).

Dans le cadre de la rénovation de ces centrales, EDF souhaite disposer d'une méthode garantissant que le comportement réel d'un API soit strictement conforme à sa spécification. L'API ne doit être testé que par la seule observation du comportement de ses entrées/sorties logiques. Le comportement à tester est un comportement séquentiel pour lequel il est demandé de réaliser un test exhaustif couvrant 100% du comportement spécifié.

2.4.1. Positionnement scientifique

Le test de conformité est une technique de test fonctionnel qui a pour objectif de s'assurer que le comportement d'une implantation, considérée comme une boîte noire dont on ne peut observer que les entrées/sorties, est identique au comportement spécifié (Figure 2.13). De manière générale, le test de conformité d'une implantation comporte deux phases :

- Construction d'une séquence de test à partir de la spécification, chaque élément de cette séquence étant composé d'un vecteur d'entrée et du vecteur de sortie escompté en réponse au vecteur d'entrée associé. La construction de cette séquence nécessite que la spécification soit exprimée sous la forme d'un modèle formel et qu'un taux de couverture pour ce modèle ait été préalablement défini.
- Exécution proprement dite du test de conformité. Durant cette phase, pour chaque élément de la séquence précédemment élaborée, la réponse produite par l'implantation est comparée à celle attendue. La non- conformité de cette réponse conduit à un verdict de test négatif.

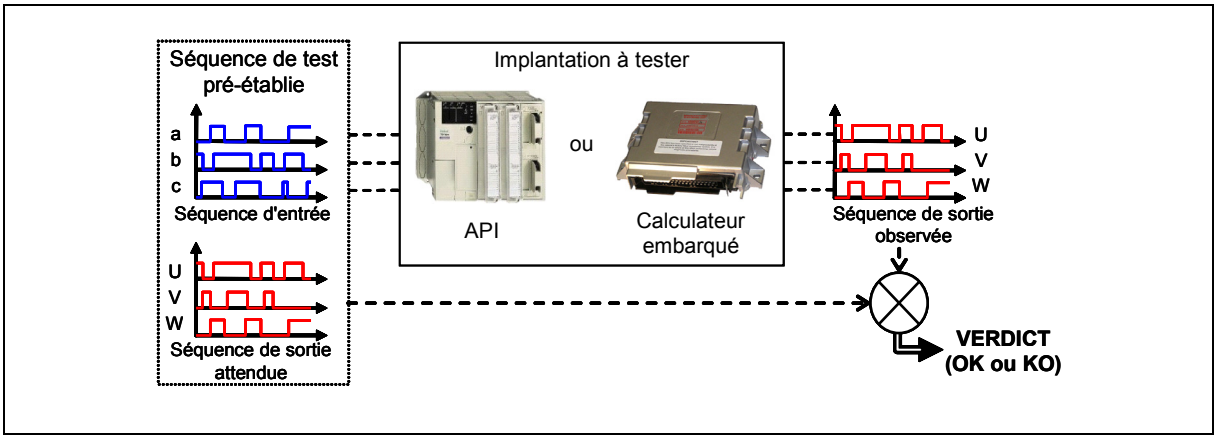


Figure 2.13. Principe du test de conformité

De nombreux résultats théoriques ont été obtenus dans ce domaine pour les systèmes non temporisés. Il a été envisagé de s'intéresser à des spécifications exprimées sous la forme de ma-

chines de Mealy [34], de systèmes de transitions [32] [52] ou de certaines classes de réseaux de Petri [54]. Ces travaux ont permis de mettre au point des méthodes efficaces de construction de la séquence de test et de l'utilisation de cette séquence sur un modèle de l'implantation, en mettant en évidence en particulier les cas de test non-concluant. Lors de ces travaux, les auteurs se sont intéressés à l'analyse de l'exécution sur un modèle de l'implantation, et non sur un contrôleur logique réel. Il n'a donc pas été pris en compte l'influence possible des caractéristiques technologiques du contrôleur réel sur le verdict du test de conformité.

La thèse de Julien Provost [T4] avait comme objectif d'intégrer les caractéristiques technologiques du contrôleur dans ce type de travaux afin de faciliter l'acceptation des techniques de test de conformité par l'industrie. Nous nous sommes particulièrement intéressés au test de conformité d'un API à scrutation cyclique des entrées dont la spécification du comportement est décrite dans le langage Grafcet.

Le test de conformité d'un API à l'aide d'un banc de test nécessite que le comportement spécifié soit exprimé en extension (représentation exhaustive sous une forme énumérée) afin de lister et d'exécuter chaque comportement élémentaire à tester. Cependant, ce n'est pas sous cette forme que sont données les spécifications dans l'industrie. Le succès du Grafcet repose sur sa capacité d'exprimer un comportement séquentiel complexe d'un manière très compacte grâce aux réceptivités associées aux transitions. Pour pouvoir exécuter un test exhaustif à partir d'un grafcet, il est nécessaire que le comportement décrit soit entièrement énuméré pour déterminer chaque comportement élémentaire. Nous avons choisi d'exprimer ce comportement à l'aide d'une machine de Mealy pour exploiter les résultats obtenus sur ce modèle (Figure 2.14).

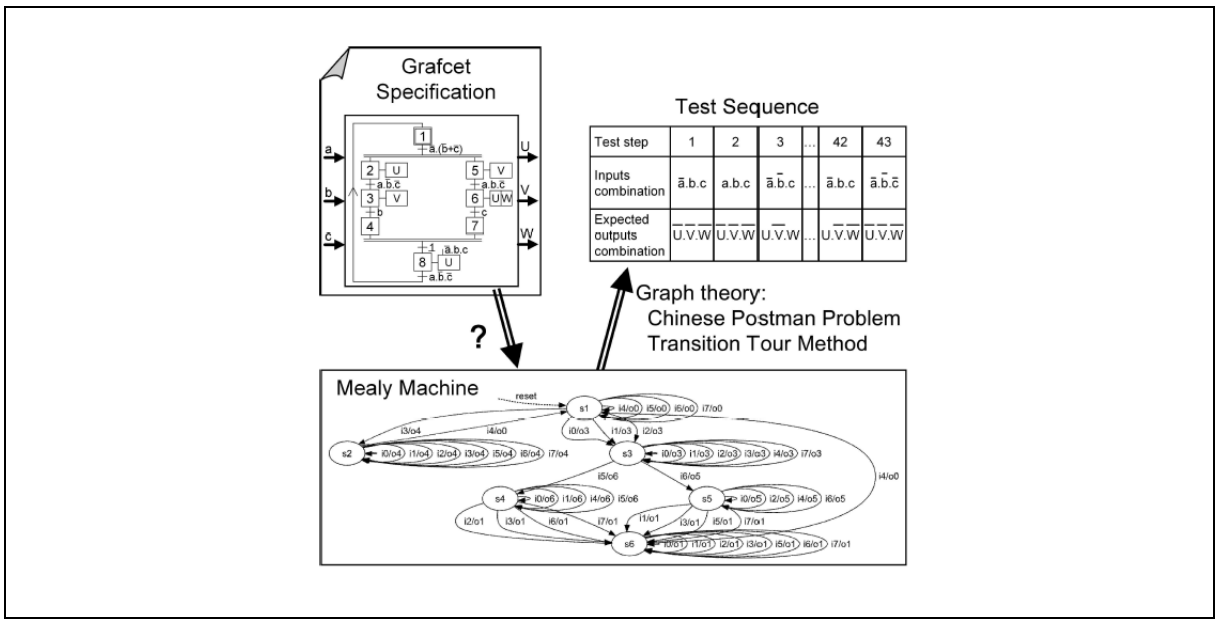


Figure 2.14. Principe de génération d'une séquence de test pour une spécification Grafcet

A notre connaissance le test de conformité d'un API par la seule observation du comportement de ses entrées/sorties n'a jamais fait l'objet d'une étude systématique avant ces travaux. Pour mettre en œuvre cette technique, les contraintes technologiques liées à l'exécution de la séquence de test ont du être clairement identifiées :

- Seul le comportement stable de l'API est observable sans erreur depuis l'extérieur de l'équipement. En raison du temps de filtrage des cartes de sortie d'un API, l'émission d'une sortie est différée dans le temps et son apparition sur seulement quelques cycles est difficilement percevable depuis l'extérieur de l'API.

- Le comportement séquentiel à tester doit donc intégrer la recherche des états stables de la commande car seuls ceux-ci sont observables.
- L'acquisition des sorties émises par un API doit se faire de manière synchrone après un délai d'attente donné. Ce délai d'attente dépend de la technologie retenue pour les cartes de sortie et du temps maximum d'évolution entre deux états stables.

- Un API n'est pas capable de détecter le synchronisme d'évolution de ses entrées [R11] car son fonctionnement interne intègre une lecture séquentielle des informations produites par les cartes d'entrée.
  - La réponse d'un API à une variation synchrone de ses entrées peut être biaisée par la non perception de cette synchronisation. Un API ne doit donc être sollicité qu'en faisant varier une seule entrée à la fois si on veut être certain que la sollicitation prise en compte par l'API soit celle prévue.

2.4.2. Détails des principaux résultats obtenus

Parmi les résultats obtenus dans le cadre de ce travail, certains d'entre-eux ont une portée théorique (passage du Grafcet à la machine de Mealy) et d'autres ont une portée plus opérationnelle (exploitation de la machine de Mealy pour la génération de la séquence de tests). L'apport majeur des travaux de Julien Provost réside dans la méthode proposée pour déterminer et mettre en œuvre un test de conformité d'un contrôleur logique par l'intermédiaire de ses entrées-sorties qui tienne compte des différentes spécificités de fonctionnement d'un API. Cette méthode a été validée par des essais sur des équipements réels.

Le passage du Grafcet à la machine de Mealy repose sur l'exploitation d'un modèle intermédiaire, dénommé Automate des Localités Stables, qui regroupent l'ensemble des états stables du grafcet et des évolutions entre ces états (Figure 2.15).

L'automate des localités stables (où « Stable Location Automaton ») qui décrit le comportement stable du grafcet est défini formellement de la manière suivante [A6] :

Définition 5 Automate des Localités Stables équivalent à un grafcet

Un Automate des Localités Stables (SLA) est un 5-uplet $(I_{SLA}, O_{SLA}, L, l_{Init}, Evol)$ où :

- I_{SLA} est l'ensemble des entrées du grafcet,
- O_{SLA} est l'ensemble des sorties du grafcet,
- L est l'ensemble des localités stables l,
- l_{Init} est la localité initiale ($l_{Init} \in L$),
- $Evol$ est l'ensemble des évolutions e.

Une localité stable $l \in L$ est définie par le triplet $(S_{Act}, O_{Em}, E_{Stab(I_{SLA})})$:

- S_{Act} est le sous-ensemble des étapes du grafcet actives,
- O_{Em} est le sous-ensemble des sorties émises,
- $E_{Stab(I_{SLA})}$ est une expression booléenne des entrées I_{SLA}. Cette expression est vraie pour les seules combinaisons des entrées pour lesquelles la localité est stable.

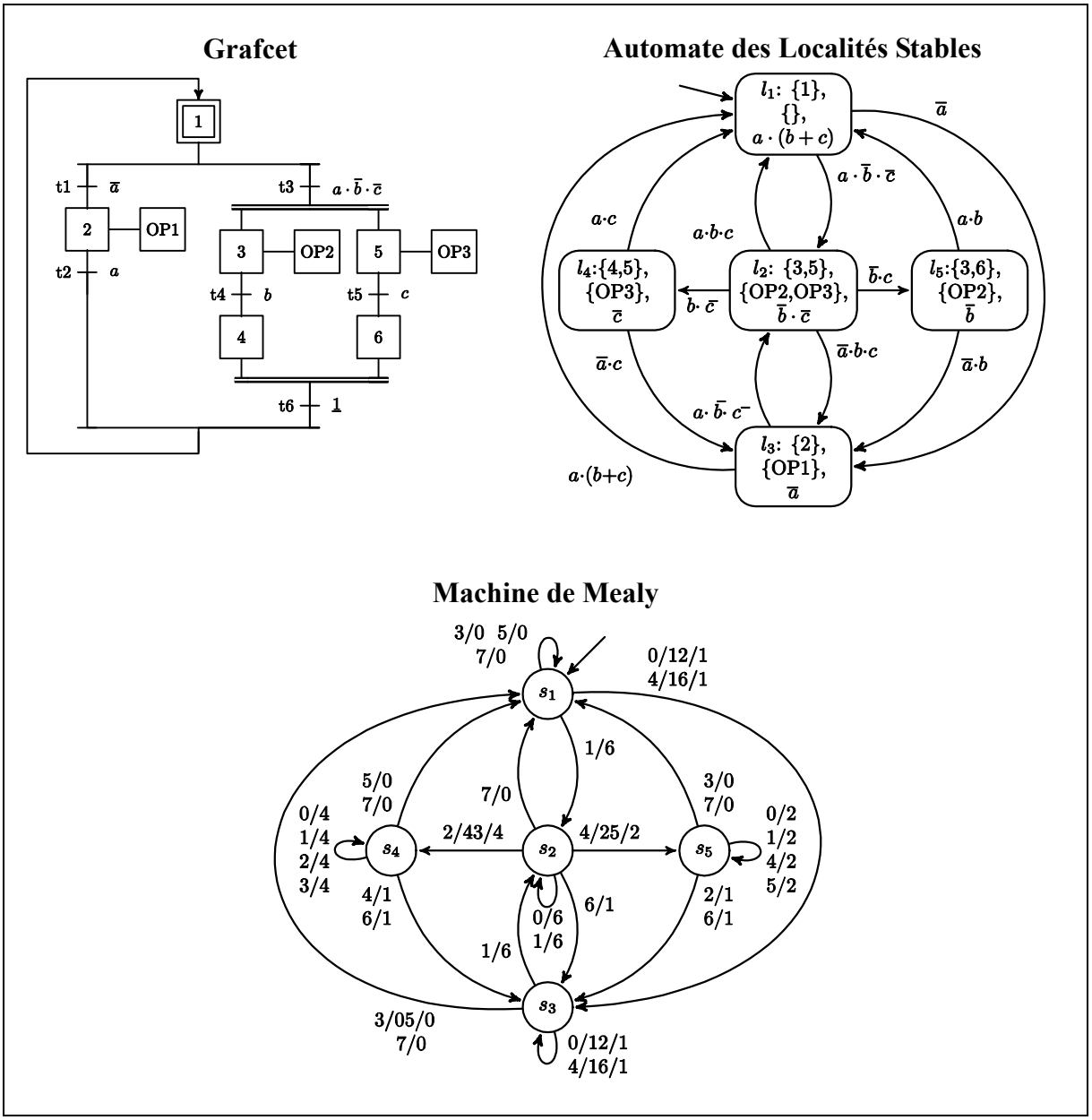


Figure 2.15. Détail du passage du Grafcet à la machine de Mealy

Une évolution $e \in Evol$ entre 2 localités stables est définie par le triplet $(l_U, l_D, E_{Evol(I_{SLA})})$:

- l_U est la localité Amont (Up) de e ($l_U \in L$),
- l_D est la localité Aval (Down) de e ($l_D \in L$),
- $E_{Evol(I_{SLA})}$ est une expression booléenne des entrées I_{SLA}. Cette expression est vraie pour les seules combinaisons des entrées pour lesquelles le SLA évolue de l_U à l_D.

D'une manière générale, l'obtention du SLA à partir d'un grafcet s'obtient en calculant de proche en proche les situations atteintes lors du franchissement simultané de tous les ensembles de transitions simultanément franchissables [C28] [A6]. La détermination de ces ensembles de transitions franchissables se fait par calcul symbolique à partir des réceptivités des transitions simultanément validées.

Le SLA obtenu à l'aide de cet algorithme possède différentes propriétés que nous avons regroupées sous un critère de « définition cohérente ».

Définition 6 Condition de cohérence d'un SLA

Un SLA est dit bien défini lorsque les sept propriétés suivantes sont satisfaites :

- Distinction des localités :

$$\forall (l_1, l_2) \in L^2, \qquad S_{Act}(l_1) \neq S_{Act}(l_2) \qquad \text{OU} \qquad O_{Em}(l_1) \neq O_{Em}(l_2)$$

- Distinction des évolutions :

$$\forall (e_1, e_2) \in Evol^2, \qquad l_U(e_1) \neq l_U(e_2) \qquad \text{OU} \qquad l_D(e_1) \neq l_D(e_2)$$

- Absence de boucles :

$$\forall e \in Evol, \qquad l_U(e) \neq l_D(e)$$

- Déterminisme des évolutions :

$$\forall (e_1, e_2) \in Evol^2, \qquad \text{Si } (l_U(e_1) = l_U(e_2)) \text{ alors } E_{Evol(I_{SLA})}(e_1) \cdot E_{Evol(I_{SLA})}(e_2) = 0$$

- Exclusion entre la condition de stabilité et les possibilités d'évolutions :

$$\forall l_e \in L, \qquad E_{Stab(I_{SLA})}(l) \cdot \left(\sum_{\substack{e_i \in Evol \\ l_U(e_i) = l_e}} E_{Evol(I_{SLA})}(e_i) \right) = 0$$

- Complémentarité entre la condition de stabilité et les possibilités d'évolutions :

$$\forall l \in L, \qquad E_{Stab(I_{SLA})}(l) + \left(\sum_{\substack{e_i \in Evol \\ l_U(e_i) = l_e}} E_{Evol(I_{SLA})}(e_i) \right) = 1$$

- Absence de localité instable :

$$\forall (e_1, e_2) \in Evol^2, \qquad \text{Si } (l_D(e_1) = l_U(e_2)) \text{ alors } E_{Evol(I_{SLA})}(e_1) \cdot E_{Evol(I_{SLA})}(e_2) = 0$$

Le SLA ainsi calculé correspond à la représentation du comportement stable atteignable du grafcet étudié. Chacune des évolutions décrites est conditionnée par une expression booléenne. Cette expression booléenne regroupe toutes les combinaisons d'entrées qui induisent la même évolution.

La machine de Mealy recherchée correspond à une représentation exhaustive et entièrement énumérée des comportements élémentaires du SLA. Pour représenter tous ces comportements élémentaires, la machine de Mealy doit avoir :

- autant d'états que le SLA a de localités,
- autant de lettres dans l'alphabet d'entrée qu'il y a de combinaisons possibles pour les entrées du SLA,
- autant de lettres dans l'alphabet de sorties qu'il y a de combinaisons possibles pour les sorties du SLA,
- autant de transitions qu'il existe de couples (localité, combinaison des entrées) dans le SLA.

Une fois la machine de Mealy générée, il est possible de construire une séquence de test couvrant la totalité du comportement spécifié, c'est-à-dire parcourant toutes ses transitions. La séquence de test de longueur minimale est obtenue par implantation d'un des algorithmes

proposés pour la résolution sur un graphe orienté du problème du postier chinois [37]. C'est lors de cette dernière étape, que la sensibilité à l'explosion combinatoire est la plus grande.

La séquence générée est une séquence initialisable de tests élémentaires $t^i = (s_U, c_I, s_D, c_O)$ où :

- s_U est un état stable de la spécification,
- c_I est une combinaison des entrées logiques,
- s_D est l'état stable atteint depuis s_U pour la combinaison des entrées c_I,
- c_O est la combinaison des sorties logiques émises dans l'état stable s_D,

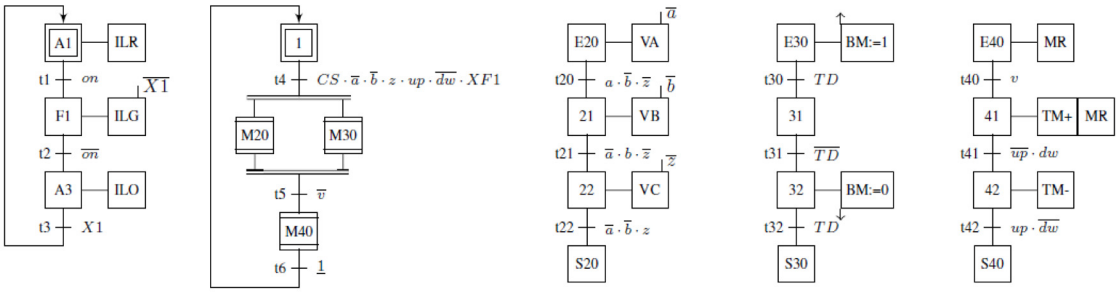
pour laquelle :

- $s_U(t^0) = s_{Init}$ (condition d'initialisation),
- $s_D(t^i) = s_U(t^{i+1})$ (condition de consistance nécessaire à l'enchaînement des tests).

Les données présentées sur la figure 2.16 ont été obtenues à l'aide du démonstrateur « Teloco » [C58] développé en Python. Ce démonstrateur permet de construire automatiquement le SLA, de représenter de manière exhaustive son comportement à l'aide d'une machine de Mealy et de générer une séquence de test permettant de parcourir chaque transition de cette machine de Mealy

Grafcet étudié

- 9 entrées, 10 sorties
- 2 grafcets connexes et 3 macro-étapes : 16 étapes, 15 transitions



Automate des Localités Stables équivalent

- 9 entrées, 10 sorties
- 64 localités, 389 évolutions

Machine de Mealy

- 2^9 entrées, 2^{10} sorties
- 64 états, 32 768 (64×2^9) transitions

Séquence de test

- Une séquence unique de 73 528 pas

Figure 2.16. Du grafcet à la séquence de test

Pour le grafcet proposé, les différents temps de calcul sur un PC portable sont :

- Génération du SLA : 260 ms,

- Génération de la machine de Mealy : 135 ms,
- Génération de la séquence de test : 390 ms.

La durée d'exécution de cette séquence de test dépend de sa longueur et du temps nécessaire à l'exécution d'un test élémentaire sur le banc de test utilisé. Pour le banc développé au LURPA (Figure 2.17), son automatisation permet de solliciter l'API à tester à l'aide d'une nouvelle variation des entrées toutes les 20 ms. Pour le grafcet de la figure 2.16, la durée totale du test de conformité est de 25 minutes.

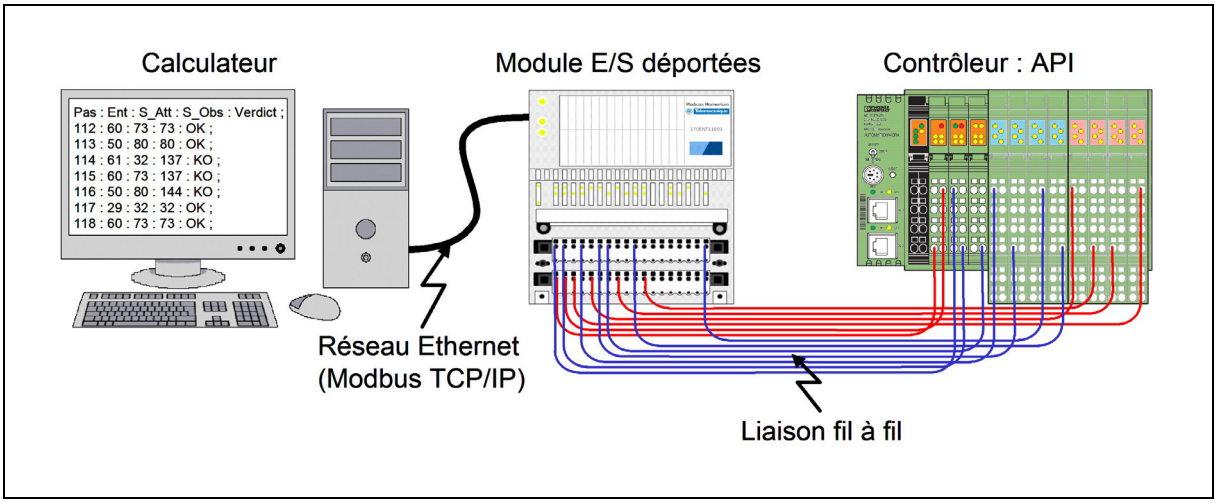


Figure 2.17. Détail du banc de test développé au LURPA

Le démonstrateur « Teloco » est disponible en téléchargement depuis le site WWW du laboratoire à l'adresse suivante : http://www.lurpa.ens-cachan.fr/-211030.kjsp.

Les expérimentations faites à l'aide de ce banc de test ont montré que la réponse d'un API à une variation synchrone de ses entrées pouvait être biaisée par la non perception de cette synchronisation (Figure 2.18).

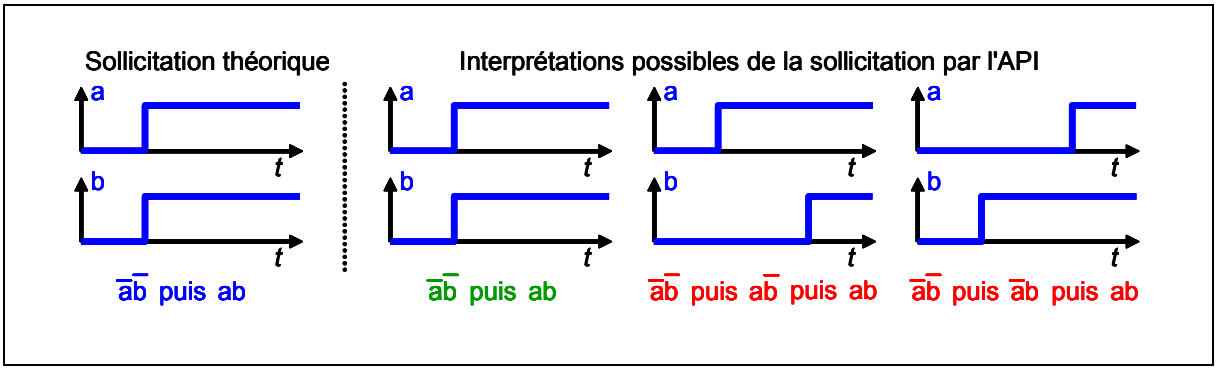


Figure 2.18. Interprétations possibles par un API d'une évolution synchrone de 2 entrées logiques

Pour garantir l'absence de biais, il est nécessaire de ne solliciter l'API qu'en ne faisant varier qu'une seule de ses entrées logiques à la fois. Cependant, la génération d'une séquence SIC (Single Input Change [56]) pour la totalité du comportement à tester n'est pas toujours possible [C23] [C27].

Définition 7 SIC-testabilité du comportement séquentiel d'un système logique
La partie SIC-testable du comportement séquentiel d'un système logique est le sous-ensemble des comportements élémentaires (s_U, c_I, s_D, c_O) qui peuvent être testés en ne faisant varier qu'une entrée logique entre 2 pas de tests.

En considérant que le comportement élémentaire d'un système logique séquentiel correspond au quadruplet (s_U, c_I, s_D, c_O), il a été montré dans [A9] que l'ensemble des comportements élémentaires SIC-testables peut être obtenu à l'aide d'un calcul de point fixe.

Soit R_{Gray} la relation d'adjacence entre 2 combinaisons des entrées c_I et c_I' basée sur le calcul de la distance entre les mintermes qui les représentent. Deux tests élémentaires t^i et t^j satisfont une évolution SIC si et seulement si :

$$s_D(t^i) = s_U(t^j) \text{ et } c_I(t^i) \; R_{Gray} \; c_I(t^j)$$

En considérant qu'une séquence SIC est une séquence initialisable composées de tests élémentaires satisfaisant une évolution SIC, tous les comportements testés au cours de cette séquence sont SIC-testables. Si s_D est l'état atteignable à l'aide de cette séquence SIC pour la valeur courante des entrées est c_I, alors les comportements élémentaires $(s_D, c_I', *, *)$ avec $c_I' \; R_{Gray} \; c_I$ sont également SIC-testable.

A partir de l'ensemble des comportements élémentaires initialisables, le calcul de point fixe suivant permet de calculer la partie SIC-testable du comportement séquentiel étudié :

$$P_{SIC}(k+1) = P_{SIC}(k) \cup \{(s_D, c_I', *, *) \mid \exists (s_U, c_I, s_D, c_O) \in P_{SIC}(k), c_I' \; R_{Gray} \; c_I\}$$

Le lecteur pourra se reporter à la publication [A9] jointe en annexe pour plus détails sur ce calcul de point fixe et son implémentation.

La séquence SIC permettant de tester cette partie SIC-testable s'obtient à partir des algorithmes de la théorie des graphes utilisés pour le problème du voyageur de commerce [17]. Le graphe à étudier est tel que :

- Chaque sommet du graphe représente un couple (s_U, c_I) distinct,
- Les arcs représentent l'exécution d'un comportement élémentaire ou la variation SIC du vecteur d'entrée :
  - A chaque comportement élémentaire (s_U, c_I, s_D, c_O) instable $(s_D \neq s_U)$, correspond un arc orienté $((s_U, c_I), (s_D, c_I))$,
  - A l'issue de chaque comportement élémentaire (s_U, c_I, s_D, c_O) stable $(s_D = s_U)$, il est possible d'enchaîner une des $n_{I_{SLA}}$ variations possibles du vecteur d'entrée. Chacune de ces variations du vecteur d'entrée correspond à un arc orienté $((s_U, c_I), (s_U, c_I'))$ où $c_I \; R_{Gray} \; c_I'$.

La limite actuelle pour la génération de la séquence SIC est notre capacité à résoudre de manière efficace le problème du voyageur de commerce pour des graphes de taille conséquente. Les graphes manipulés peuvent présenter jusqu'à $9 \cdot 10^5$ nœuds.

2.4.3. Conclusions

Les résultats obtenus dans le cadre du test de conformité de contrôleurs logiques industriels ayant un comportement séquentiel ont été validés par des expérimentations conduites sur des équipements réels. Lors de ces essais, nous avons pu constater que les contraintes technologiques citées liées à la mise en œuvre des tests ne pouvaient pas être ignorées [C50]. Le test d'un API par la seule observation du comportement de ses entrées/sorties ne peut pas se faire sans tenir compte de son fonctionnement interne et des contraintes induites par ses cartes d'entrée et de sortie. Cela a des conséquences sur le choix de la représentation de son comportement théorique.

Nos travaux ont montré que le test exhaustif d'un comportement séquentiel par la seule observation du comportement de ses entrées/sorties nécessite que la spécification soit SIC-testable. Un test exhaustif de la partie SIC-testable d'une spécification est possible mais il s'avère très coûteux lorsque le nombre d'entrées augmente. Cette approche doit donc être réservée pour des systèmes hautement critiques pour lequel un test exhaustif est exigé par un organisme de sûreté.

Ces premiers travaux sur le test de conformité peuvent être poursuivis dans plusieurs directions complémentaires en fonction des résultats attendus pour cette technique :

- Test exhaustif en intégrant la connaissance du code implanté dans le contrôleur industriel :
  - Lorsque la mise en place d'un test exhaustif est exigée, le meilleur moyen de contourner l'explosion combinatoire induite par une analyse globale est de partitionner la spécification globale et de ne générer qu'un test exhaustif que pour chacune des parties. Pour que cela soit valide, il est nécessaire que le code implanté dans le contrôleur industriel soit analysé afin de garantir le respect des hypothèses de travail (cohérence du partitionnement, indépendance des parties de code...).
- Test à taux de couverture garanti :
  - Pour la grande majorité des installations un test exhaustif ne peut être envisagé et il nécessaire de recourir à des tests partiels suivants des critères donnés comme la garantie d'un taux de couverture fixé. Cependant dans le cas des systèmes de contrôle-commande et des langages de programmation qu'ils utilisent, la simple définition de ce taux de couverture présente déjà une certaine difficulté...

La poursuite de ce type de travaux doit cependant se faire en privilégiant une collaboration industrielle afin que les critères retenus correspondent à une attente d'un secteur d'activité donné.

Chapitre 3

Formalisation des outils pour les analyses de sûreté

Ce chapitre présente la partie des recherches que j'ai menées dans le cadre de la formalisation des outils pour les analyses de sûreté en utilisant les paradigmes et modèles des SED.

3.1. Introduction

L'analyse de sûreté d'un système complexe repose encore aujourd'hui principalement sur l'utilisation d'outils et de méthodes issues du monde industriel. Ces outils et méthodes sont variés tant sur le plan des possibilités offertes, que des modèles mathématiques sous-jacents.

Le groupe de travail M2OS (Management, Méthodes, Outils, Standards) de l'IMdR[1] (Institut pour la Maîtrise des Risques) propose un recueil de 30 fiches synthétiques détaillant les outils et méthodes employés lors d'une analyse de sûreté. À la lecture de ces fiches, il apparait clairement que l'analyse de sûreté d'un système est une activité complexe nécessitant à la fois une culture pluridisciplinaire pour intégrer les différents aspects à prendre en compte et très pointue pour établir le modèle avec le niveau de détails adéquat.

Conçus initialement pour ne prendre en compte que des comportements élémentaires, les outils utilisés en analyse de sûreté s'appuient le plus souvent sur des modèles mathématiques simples. Pour répondre aux attentes pressantes des utilisateurs, de nouvelles primitives ont été progressivement introduites. Dans certains outils, l'ajout de ces nouvelles primitives s'est malheureusement fait sans remettre en cause le choix initial des modèles mathématiques utilisés. Cela a conduit à des définitions informelles de ces primitives pouvant remettre en cause les résultats obtenus.

1. http://www.imdr.fr/

Les travaux que je mène dans cette thématique ont pour objectif de renforcer les fondements mathématiques de ces outils métiers afin d'en éliminer les possibilités d'incohérence et ainsi permettre une analyse plus pertinente des modèles établis par les experts. Comme je m'appuie essentiellement les compétences acquises dans le domaine des SED, je me suis naturellement focalisé sur les outils d'analyse de sûreté qui font référence aux évolutions discrètes des systèmes.

Ma contribution à la formalisation des outils pour les analyses de sûreté a porté sur les trois problématiques suivantes :

- la modélisation algébrique des arbres de défaillance dynamiques,
- la formalisation de la cohérence et le calcul des séquences de coupe minimales pour les systèmes dynamiques réparables,
- l'analyse prévisionnelle des risques d'un point de vue quantitatif pour les systèmes réparables à l'aide de chaînes de Markov.

Les modèles construits décrivent l'état du système en fonction de l'occurrence des défaillances et des réparations des composants qui le composent. Une fois établis, les ingénieurs fiabilistes mènent :

- des analyses qualitatives pour connaître les différents scénarios qui conduisent à la défaillance du système,
- des analyses quantitatives pour chiffrer le taux d'indisponibilité du système.

Les travaux réalisés dans le cadre de la modélisation algébrique des arbres de défaillance dynamiques ont portés sur des systèmes ne comportant que des composants non réparables. Les résultats obtenus sont exploitables pour des analyses qualitatives et quantitatives.

Les travaux réalisés dans le cadre de la formalisation de la cohérence et le calcul des séquences de coupe minimales ont été développés pour renforcer les fondements scientifiques des analyses qualitatives des systèmes dynamiques à base de composants réparables.

Les travaux réalisés dans le cadre de l'analyse prévisionnelle des risques à l'aide de chaînes de Markov ont pour objectif d'accroître la taille des systèmes qu'il est possible d'étudier lors d'analyses quantitatives.

3.2. Modélisation algébrique des arbres de défaillance dynamiques

Les travaux portant sur la modélisation algébrique des arbres de défaillance dynamiques ont été les premiers que j'ai conduit dans le cadre de la formalisation des outils pour les analyses de sûreté. L'événement déclencheur est une présentation de Yiannis Papadopoulos de ses activités de recherche en 2005 lors d'un séjour à l'ENS Cachan. Il avait présenté la logique temporelle PANDORA développée pour tenir compte de l'ordre d'occurrence d'événements dans un arbre de défaillance dynamique. Son intervention s'était conclue sur l'impossibilité actuelle de simplifier des arbres de défaillance comportant des portes PAND en raison de l'absence de résultats mathématiques pour le faire.

En reprenant l'approche suivie dans le cadre de la modélisation des opérateurs temporels utilisés en automatique discrète [A11], il a été possible de modéliser algébriquement la porte PAND à l'aide d'une représentation permettant d'établir les théorèmes nécessaires à la simplification d'un arbre de défaillance dynamique. Ce travail de modélisation s'est déroulé dans le cadre du travail de master Recherche de Guillaume Merle [S7] (qui a donné lieu à [C17]) puis de ses travaux de thèse [T3].

Pour ces travaux, Guillaume Merle a obtenu le prix 2009-2010 des meilleures thèses en automatique du GDR MACS et de la section automatique du Club EEA, dans la catégorie « Systèmes à Événements Discrets et Systèmes hybrides ».

3.2.1. Positionnement scientifique

Les arbres de défaillance [49] [53] sont largement utilisés dans l'industrie pour conduire des études prévisionnelles de fiabilité d'un système. Ils sont établis par les experts fiabilistes pour identifier les différentes causes qui conduisent à la défaillance du système analysé. Leur construction suit un processus itératif partant de l'événement redouté (TE pour Top Event) placé à la racine de l'arbre. Cette construction consiste à décomposer chaque événement étudié en événements intermédiaires jusqu'à ce que cette décomposition soit impossible ou jugée superflue. Les événements non décomposés sont les événements de base (BE) de la modélisation. Ils forment les feuilles de l'arbre de défaillance et doivent correspondre à des processus stochastiquement indépendants. Une fois établis, ces arbres de défaillance sont exploités pour déterminer les conditions minimales pour que la défaillance ait lieu (étude qualitative) ou quantifier la probabilité d'occurrence de cette défaillance (étude quantitative) en fonction des événements de base.

Lorsque chaque décomposition est décrite à l'aide d'une porte OR ou d'une porte AND, l'arbre de défaillance est dit statique (ADS). La forme simplifiée de la fonction booléenne qu'il décrit est dite fonction de structure. Sa connaissance permet de mener efficacement les analyses qualitatives ou quantitatives à l'aide de diagrammes de décision binaires (BDD) [46] [47].

Pour compléter le pouvoir d'expression des ADS, de nouvelles portes ont été ajoutées telles que les portes PAND, FDEP et SPARE. les arbres de défaillance qui les contiennent sont appelés arbres de défaillance dynamiques (ADD) [19] en raison la relation entre l'événement sommet et les événements de base. Dans un ADD, la défaillance de l'événement sommet ne dépend pas seulement de la défaillance des événements de base mais également de l'ordre d'occurrence de ces défaillances (Figure 3.1).

Cependant, comme l'ordre d'occurrence des défaillances des événements de base ne peut pas être pris en compte dans le modèle booléen des défaillances, il a fallu proposer différentes stra-

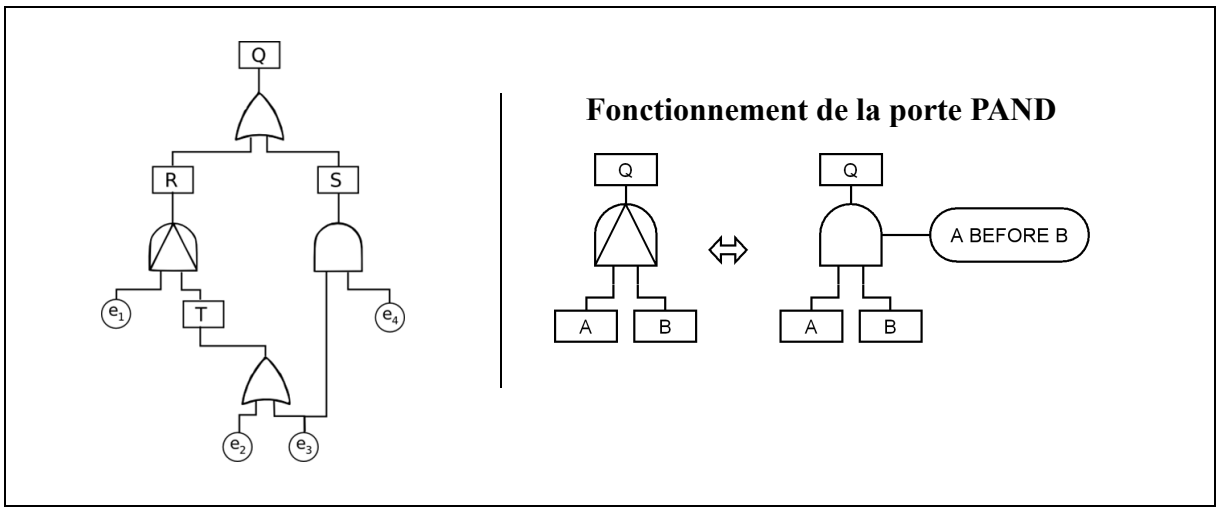


Figure 3.1. Arbre de défaillance comportant une porte PAND (Priority AND)

tégies pour pouvoir continuer à mener les analyses qualitatives et quantitatives des systèmes. Ces stratégies sont généralement basées sur le principe suivant :

- Division de l'ADD en sous-arbres (ou modules) purement statiques ou dynamiques en cherchant à maximiser les parties statiques,
- Analyse séparée des différents modules,
- Combinaison des résultats obtenus pour obtenir le résultat global.

L'idée était de conserver la possibilité d'exploiter les BDD pour l'étude des parties statiques de l'arbre et d'avoir seulement recours aux techniques plus complexes pour les parties faisant référence aux aspects dynamiques. Selon les auteurs, les modules dynamiques sont analysés à l'aide de chaines de Markov [9] [10] [11], de Réseaux de Petri Stochastiques [5] [15] ou de réseaux bayésiens temporels [7] [8] [39]. Les auteurs reconnaissent que cette approche n'est possible que si les différents modules ne présentent aucun élément de base en commun. Dans le cas contraire, l'arbre doit être étudié dans sa globalité comme un seul module dynamique pour lequel il faut identifier toutes les séquences possibles d'occurrences des événements de base.

L'analyse qualitative d'un ADS repose sur l'étude de ses coupes minimales [53]. Les coupes minimales correspondent aux combinaisons les plus courtes entre les événements de base qui provoquent la défaillance de l'événement sommet pour l'arbre étudié. Dans le cas des ADS cohérents [4], ces coupes minimales s'obtiennent automatiquement à partir de la fonction de structure de cet arbre car chaque coupe minimale correspond à un terme de la forme simplifiée de la fonction booléenne correspondant à la fonction de structure [47].

L'analyse qualitative d'un ADD repose quant-à-elle sur l'étude de ses séquences minimales. Dans le cadre des ADD ne comportant pas de composants réparables, ces séquences minimales correspondent aux séquences les plus courtes des événements de base qui provoquent la défaillance de l'événement sommet de l'arbre étudié. Pour identifier ces séquences avec des approches basées sur des modèles à états, une représentation exhaustive des états du système est nécessaire. Comme cette représentation est fortement sensible à l'explosion combinatoire, la taille des modèles manipulables s'avère limitée (Figure 3.2).

A l'exception de l'équipe de Yiannis Papadopoulos qui a poursuivi l'approche PANDORA [55], aucune autre équipe n'a envisagé de proposer une modélisation algébrique des portes PAND, FDEP et SPARE afin d'établir la fonction de structure d'un ADD et de l'exploiter pour les analyses qualitatives et quantitatives des systèmes.

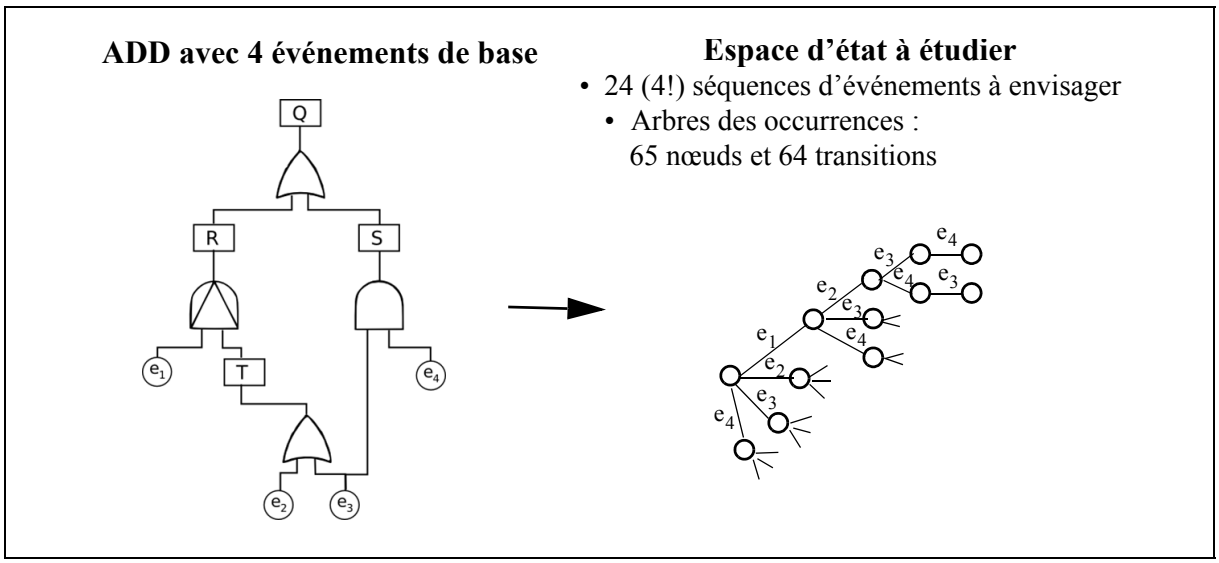


Figure 3.2. Approches traditionnelles pour l'étude qualitative d'un ADD

3.2.2. Détails des principaux résultats scientifiques obtenus

La contribution majeure des travaux de Guillaume Merle a été dans la définition d'un cadre algébrique homogène permettant d'établir la fonction de structure d'un ADD comportant les portes OR, AND, PAND, FDEP et SPARE. Ce cadre algébrique est fondé sur un modèle temporel des événements présents dans un ADD et des portes qui les manipulent [A3] [A5].

3.2.2.1. Cadre temporel utilisé

Le modèle temporel retenu pour ce cadre algébrique repose entièrement sur l'hypothèse de non réparabilité des événements présents dans un ADD. Nous considérons que :
- un événement représente l'occurrence d'une faute,
- un événement est faux tant que la faute n'est pas apparue,
- lorsque la faute apparait, l'événement devient vrai et restera vrai ensuite,
- un événement est instantané : son changement de valeur est immédiat.

Pour prendre en compte l'ordre d'occurrence des événements, il a été nécessaire de faire référence à leur unique date d'occurrence. Dans notre approche, tous les événements présents dans un ADD sont décrits par une fonction temporelle définie sur $\Re^+ \cup \{+\infty\}$ dans $B = \{0, 1\}$ continue à droite et admettant un seul point de discontinuité (Figure 3.3). Chaque événement est caractérisé par sa date d'occurrence : $d(e)$.

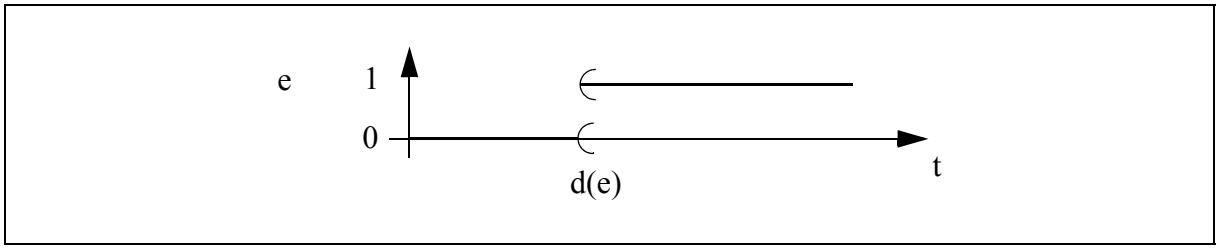


Figure 3.3. Modèle d'un événement non réparable

Soit ξ_{nr} l'ensemble des événements non-réparables. Soient $\perp$ et $\top$ les événements non-réparables dont les dates d'occurrences sont $d(\perp) = +\infty$ et $d(\top) = 0$.

ξ_{nr} peut être muni de deux lois internes AND et OR permettant de combiner des événements non-réparables entre-eux.

$$\text{Loi } . : \quad \xi_{nr}{}^2 \to \xi_{nr} \qquad (a, b) \to a \cdot b \qquad\qquad \text{Loi } + : \quad \xi_{nr}{}^2 \to \xi_{nr} \qquad (a, b) \to a + b$$

$$\text{où } d(a \cdot b) = \begin{cases} d(b) \text{ si } d(a) < d(b) \\ d(a) \text{ si } d(a) = d(b) \\ d(a) \text{ si } d(a) > d(b) \end{cases} \quad \text{et } d(a + b) = \begin{cases} d(a) \text{ si } d(a) < d(b) \\ d(a) \text{ si } d(a) = d(b) \\ d(b) \text{ si } d(a) > d(b) \end{cases}$$

Sur le plan mathématique $(\xi_{nr}, . , +)$ est un dioïde abélien et grâce aux théorèmes suivants, ce dioïde peut être utilisé pour modéliser et simplifier les ADS :

$a \cdot b = b \cdot a$	$a \cdot a = a$	$a \cdot (b \cdot c) = (a \cdot b) \cdot c$
$a + b = b + a$	$a + a = a$	$a + (b + c) = (a + b) + c$
$a \cdot (b + c) = (a \cdot b) + (a \cdot c)$	$a + (b \cdot c) = (a + b) \cdot (a + c)$	
$a \cdot (a + b) = a$	$a + (a \cdot b) = a$	
$a + \bot = a$	$a \cdot \bot = \bot$	
$a + \top = \top$	$a \cdot \top = a$	

ξ_{nr} a été muni de trois lois internes complémentaires pour intégrer des aspects temporels. Ces lois BEFORE ($\lhd$), SIMLUTANEOUS (Δ) et Inclusive BEFORE ($\trianglelefteq$) sont définies ainsi :

$$\text{Loi } \lhd : \quad \xi_{nr}{}^2 \to \xi_{nr} \qquad (a, b) \to a \lhd b \qquad\qquad \text{Loi } \Delta : \quad \xi_{nr}{}^2 \to \xi_{nr} \qquad (a, b) \to a \Delta b \qquad\qquad \text{Loi } \trianglelefteq : \quad \xi_{nr}{}^2 \to \xi_{nr} \qquad (a, b) \to a \trianglelefteq b$$

$$\text{où } d(a \lhd b) = \begin{cases} d(a) \text{ si } d(a) < d(b) \\ +\infty \text{ si } d(a) = d(b) \\ +\infty \text{ si } d(a) > d(b) \end{cases} \quad d(a \Delta b) = \begin{cases} +\infty \text{ si } d(a) < d(b) \\ d(a) \text{ si } d(a) = d(b) \\ +\infty \text{ si } d(a) > d(b) \end{cases} \text{ et}$$

$$d(a \trianglelefteq b) = \begin{cases} d(a) \text{ si } d(a) < d(b) \\ d(a) \text{ si } d(a) = d(b) \\ +\infty \text{ si } d(a) > d(b) \end{cases}$$

Ces nouvelles lois permettent d'étudier la séquentialité d'occurrence entre les 2 événements a et b. 60 théorèmes ont été démontrés pour pouvoir développer et simplifier toutes les compositions possibles des éléments de ξ_{nr} à l'aide de cinq lois internes [T3].

3.2.2.2. Modèles algébrique et probabiliste d'un arbre de défaillance dynamique

Ce cadre algébrique a permis de modéliser les portes PAND [C21], FDEP [C25] et SPARE [C24] (Figure 3.4) et d'établir et simplifier la fonction de structure de tout ADD comportant ces portes [A3] [A5]. Pour les systèmes ne comportant que des composants non réparables, Il est maintenant possible d'analyser un ADD sans passer par une représentation explicite des séquences.

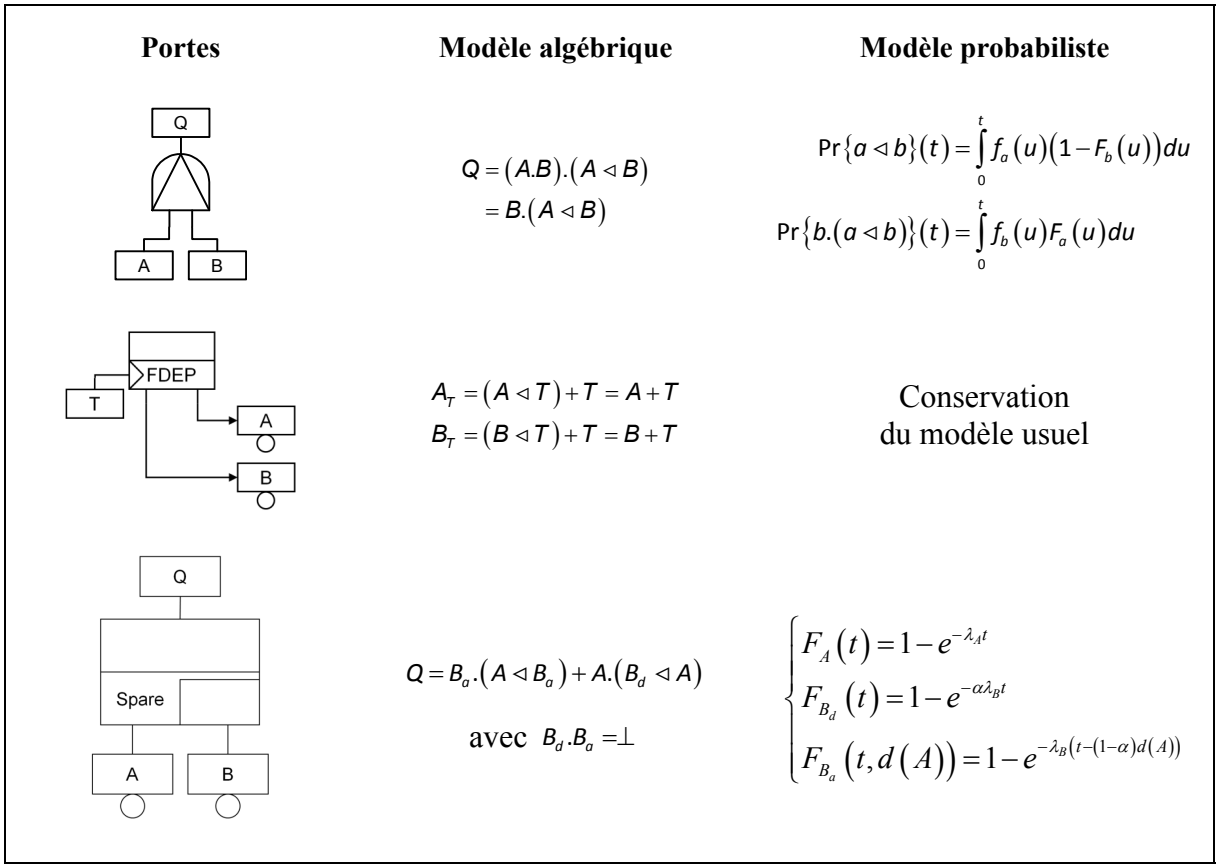


Figure 3.4. Modèles algébrique et probabiliste proposées pour les portes PAND, FDEP et SPARE

Le lecteur pourra se reporter à [A5] donnée en annexe pour obtenir plus de détails sur les modèles proposés.

Pour l'ADD de la figure 3.2, la représentation de cette fonction de structure avec les lois proposées est :

$$\begin{cases} Q = R + S \\ R = T \cdot (e_1 \trianglelefteq T) \\ S = e_3 \cdot e_4 \\ T = e_2 + e_3 \end{cases} \quad \text{soit } Q = ((e_2 + e_3) \cdot (e_1 \trianglelefteq (e_2 + e_3))) + (e_3 \cdot e_4)$$

En utilisant les théorèmes établis [T3], il est possible de développer et simplifier cette représentation algébrique pour l'exprimer ainsi :

$$Q = (e_3 \cdot e_4) + e_2 \cdot (e_1 \lhd e_2) \cdot (e_1 \lhd e_3) + e_3 \cdot (e_1 \lhd e_2) \cdot (e_1 \lhd e_3)$$

Cette forme décrit l'ensemble des conditions minimales pour que la défaillance de l'événement redouté ait lieu. Ces conditions font référence à des combinaisons des événements de base $(e_3 \cdot e_4)$ et si nécessaire, à des contraintes sur leur ordre d'occurrence $(e_1 \lhd e_2)$. Cette forme permet de représenter conjointement les coupes minimales et les séquences minimales. Elle correspond exactement aux besoins des fiabilistes lors d'une analyse qualitative.

Sur le plan mathématique, nous avons démontré que toute fonction de structure d'un ADD peut toujours être ramenée à une forme canonique de la forme :

$$Q = \sum((\prod e_i) \cdot (\prod(e_j \lhd e_k))), \text{ avec } j \notin \{i, k\}$$

Cette forme canonique peut être minimisée en éliminant les termes redondants lorsqu'ils existent. Un critère de minimisation a été proposé dans [A3] ainsi qu'un algorithme pour atteindre cette forme minimale.

Cette forme canonique peut être également exploitée pour mener des analyses quantitatives car elle permet de calculer la probabilité de défaillance de l'événement redouté. L'un des principaux intérêts de cette représentation est d'autoriser des études pour lesquelles les événements de base peuvent suivre des lois d'évolution non Markoviennes. Avec cette approche, il est possible de retenir des distributions de Weibull pour les composants et ainsi modéliser des comportements avec des défaillances juvéniles [A7].

Les données présentées figure 3.5 sont extraites de [C26] et sont relatives à une étude de cas tirée de [49]. Pour cet arbre dynamique, la forme canonique de la fonction de structure comporte 23 termes décrivant des coupes minimales (TE_2, TE_3 et TE_4) et des séquences de coupe minimales (TE_1).

3.2.3. Conclusions

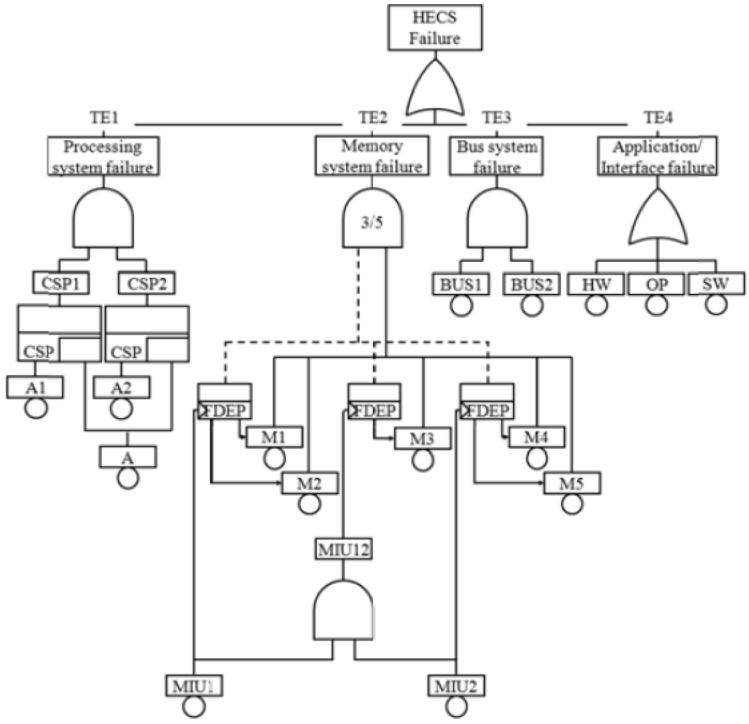
Dans le cadre de ces travaux, nous avons pu montrer qu'il était possible d'établir la fonction de structure d'un arbre de défaillance dynamiques sous l'hypothèse que les événements de base soient non-réparables. Le modèle mathématique proposé permet d'exprimer les conditions minimales que doivent respecter les événements de base pour que la défaillance de l'événement redouté ait lieu. Ces conditions font référence à l'occurrence des événements de base et si nécessaire, à des contraintes sur leur ordre d'occurrence. Les expériences conduites ont montré tout l'intérêt d'exploiter cette représentation pour mener des analyses qualitatives et quantitatives.

Avant d'aller scientifiquement plus loin dans ce travail, il est nécessaire d'évaluer finement le potentiel de cette approche par le traitement de différentes études de cas. Dans le cas d'analyses qualitatives, nos premiers essais comparatifs [T3] ont montrés que les résultats obtenus à l'aide de notre approche étaient de meilleur qualité que ceux obtenus avec certaines approches traditionnelles en raison des hypothèses simplificatrices qu'elles utilisent. D'autres expérimentations sont cependant nécessaires.

Pour mener à bien de telles expérimentations des travaux préalables doivent être réalisés :

- Notre approche repose entièrement sur la capacité d'établir la forme canonique de la fonction de structure d'un ADD. Actuellement, l'établissement de cette forme canonique s'effectue manuellement. Le premier développement à réaliser serait l'automatisation de ce calcul par le développement d'un module de calcul symbolique.
- L'établissement d'une bibliothèque d'études de cas validées par des experts de la discipline afin de disposer d'une base d'exemples reconnus et exploitables par différentes équipes de recherche. La constitution d'un arbre de défaillance dynamique est une activité de modélisation qui nécessite une certain recul. L'introduction de portes dynamiques dans un arbre de défaillance peut conduire rapidement à des arbres incohérents (arbres pour lesquels la défaillance d'un composant d'un système peut masquer la défaillance du système étudié). Nous avons découvert tardivement que certaines études exploitées par la communauté aca-

Arbre de défaillance dynamique [49]



Fonction de structure

$$TE = TE1 + TE2 + TE3 + TE4$$

$$\begin{cases} TE1 = A_a \cdot A2 \cdot (A1 \lhd A_a) \cdot (A1 \lhd A2) + A_a \cdot A1 \cdot (A2 \lhd A_a) \cdot (A2 \lhd A1) \\ TE2 = MIU1 \cdot MIU2 + MIU1 \cdot (M1 + M2 + M3) + MIU2 \cdot (M3 + M4 + M5) \\ \qquad + \sum_{\substack{i<j<k \\ i,j,k \in \{1,2,3,4,5\}}} Mi \cdot Mj \cdot Mk \\ TE3 = BUS1 \cdot BUS2 \\ TE4 = HW + OP + SW \end{cases}$$

Coupes minimales (21)

$$\{HW, OP, SW, BUS1 \cdot BUS2, MIU1 \cdot MIU2, MIU1 \cdot M1, \ldots, M3 \cdot M4 \cdot M5\}$$

Séquences minimales (4)

$$\{[A1, A2, A_a], [A1, A_a, A2], [A2, A1, A_a], [A2, A_a, A1]\}$$

Figure 3.5. ADD, fonction de structure et coupes et séquences minimales

démique et que nous avons utilisées avaient ce défaut.

La meilleure configuration possible pour mener à bien de tels travaux serait de les réaliser dans le cadre d'un outil logiciel exploité en sûreté de fonctionnement afin de bénéficier d'une dynamique de groupe. C'est naturellement vers l'outil HiP-HOPS (Hierarchically Performed Hazard Origin and Propagation Studies) développé par l'équipe de Yiannis Papadopoulos qu'il serait intéressant de se diriger.

3.3. Formalisation de la cohérence et calcul des séquences de coupe minimales pour les systèmes réparables

Ces travaux ont été réalisés dans le cadre de la thèse de Pierre-Yves Chaux [T5]. Cette thèse s'est déroulée dans le cadre d'une convention CIFRE avec la société EDF au sein du département Recherche & Développement MRI (Maitrise des Risques Industriels).

Un des rôles d'EDF R&D est de développer des méthodes et outils permettant une modélisation et une évaluation toujours plus pertinentes de la sûreté de ses installations. En raison de la durée de vie des installations de production d'énergie, EDF a du mettre en place des outils d'analyse intégrant la réparabilité des composants sur le temps de mission étudié, la capacité de reconfiguration de l'installation et les différents types de redondance entre les composants. C'est dans ce cadre qu'ont été proposés en 2003 par Marc Boussiou les Boolean logic Driven Markov Process (BDMP) [12].

Les BDMP sont une extension des arbres de défaillance dans laquelle les événements de base sont remplacés par des processus markoviens afin de prendre en compte la défaillance et la réparation des composants. A chaque élément terminal est associée un mode (« sollicité » ou non « non sollicité ») qui permet d'intégrer des dépendances simples entre les composants et matérialiser les différents types de redondances entre les composants. Le point fort des BDMP est de reposer sur la plateforme KB3 sur laquelle des algorithmes spécifiques ont été développés pour exploiter les modèles établis. Ces algorithmes exploitent des approximations maîtrisées pour les séquences amenant le système à la panne.

3.3.1. Positionnement scientifique

Il peut apparaître surprenant qu'une notion aussi utilisée que les séquences de coupe minimales ne dispose pas d'une définition unique admise par l'ensemble de la communauté. Les différentes définitions proposées ont été généralement construites à partir de la stratégie suivie par chaque équipe pour les obtenir.

Dans le cas des systèmes statiques, la notion de coupes minimales est basée sur une définition fonctionnelle : l'ensemble des coupes minimales correspond à la connaissance minimale que doit disposer l'ingénieur fiabiliste pour déterminer tous les sous-ensembles de composants dont la défaillance provoque la défaillance du système étudié. Cette capacité de reconstruction repose sur le fait que la défaillance d'un des composants du système ne peut pas masquer la défaillance du système (propriété dite de semi-cohérence [4]). Dans le cadre des arbres statiques cohérents (arbres ne comportant que des portes AND et OR), la recherche des coupes minimales se fait par le calcul des implicants premiers de la fonction combinatoire correspondant à la fonction de structure. Comme un arbre cohérent ne comporte pas de portes NOT, ces implicants sont mathématiquement les plus petits sous-ensembles de composants dont la défaillance provoque la défaillance du système étudié.

Pour les arbres de défaillance dynamiques à base de composants non réparables, l'équipe de J.B. Dugan a défini comme minimales les séquences de coupes obtenues à partir de l'algorithme proposé dans [51] pour les calculer. Cet algorithme comporte 4 étapes :

- la transformation de l'arbre dynamique en arbre statique,
- la recherche des coupes minimales de l'arbre statique,
- la définition des différentes séquences d'événements correspondant à chaque coupe minimale,

- l'élimination des séquences qui ne satisfont pas les contraintes d'occurrence données dans l'arbre initial.

Cependant, les travaux de Guillaume Merle ont montré que cette approche aboutissait à l'identification d'un sur-ensemble de séquences de défaillance pour certains arbres dynamiques. Les séquences identifiées à tort correspondaient à des séquences impossibles à atteindre vis-à-vis de l'arbre de défaillance considéré.

En 2006, Marc Boussiou a proposé pour les BDMP une définition des séquences de coupe minimales [13] basée sur une représentation à l'aide d'un automate à état. Il considère qu'une séquence de coupe est minimale si il s'agit d'une séquence conduisant à la défaillance du système dans laquelle toutes les sous-séquences contenues ne conduisent pas à la défaillance du système ou ne sont pas reconnues par l'automate fini. Les travaux de Pierre-Yves Chaux ont montré que cette définition constituait une base intéressante mais elle ne permettait pas de déterminer exhaustivement toutes les séquences de coupe [C31].

Pour l'étude des systèmes dynamiques à base de composants réparables, nous n'avons pas trouvé de définition fonctionnelle pour la notion de séquences de coupes minimales qui s'appuie sur ce quelles représentent physiquement pour le système étudié, comme ce qui est fait pour les systèmes statiques cohérents.

Pour rester en accord avec la stratégie suivie pour les arbre statiques, nous avons retenu de définir la notion de séquences de coupes minimales comme l'information nécessaire et suffisante que doit connaitre l'ingénieur fiabiliste pour reconstruire si nécessaire l'ensemble de toutes les séquences de coupe du système dynamique réparable en s'appuyant sur la propriété de cohérence du système étudié.

Pour un système dynamique à base de composants réparables, il s'agit de l'ensemble de taille minimale des séquences (de pannes ou de réparation) de longueur minimale permettant de générer l'ensemble de toutes les séquences de coupe.

3.3.2. Détails des principaux résultats scientifiques obtenus

Ne désirant pas limiter notre étude aux seuls modèles BDMP, le premier travail a été de choisir un formalisme pivot permettant de représenter le comportement dynamique de systèmes physiques à base de composants réparables.

Pour ce travail de formalisation, nous tenions à travailler sur un modèle de bas niveau pour que son interprétation soit exempt d'ambiguïté. Conscient que la difficulté de représentation porterait essentiellement sur l'ordre d'occurrence des événements de panne et de réparation, nous avons éliminé toute représentation à l'aide de structures de Kripke et privilégié une représentation à l'aide d'un automate à état fini. En retenant cette représentation, il nous a été possible de nous appuyer sur la théorie des langages et des outils associés pour cette formalisation.

Dans ce travail, nous considérons que le comportement dynamique d'un système physique à base de composants réparables peut s'exprimer à l'aide d'un automate à état fini (Figure 3.6) sous la forme du 5-uplet suivant :

$$AF = <\Sigma, Q, q_0, Q_M, \delta> \text{ où}$$

- Σ est l'alphabet formé des événements de défaillance et de réparation des différents composants,

- Q est l'ensemble des états du système,
- q_0 est l'état initial du système : dans la majorité des modèles de fiabilité, cet état correspond à l'état où tous les composants sont opérationnels,
- Q_M est l'ensemble des états marqués. Ce sous-ensemble de Q ($Q_M \subset Q$) représente tous les états où le système est défaillant.

δ est la fonction de transition ($\delta : (Q, \Sigma) \rightarrow Q$). Dans cet automate, chaque transition est étiqueté par un unique événement de défaillance ou de réparation. Soit σ une séquence admissible d'événements de Σ à partir de l'état $q \in Q$. L'état atteint par le système à l'issue de cette séquence σ sera noté $\delta(q, \sigma)$.

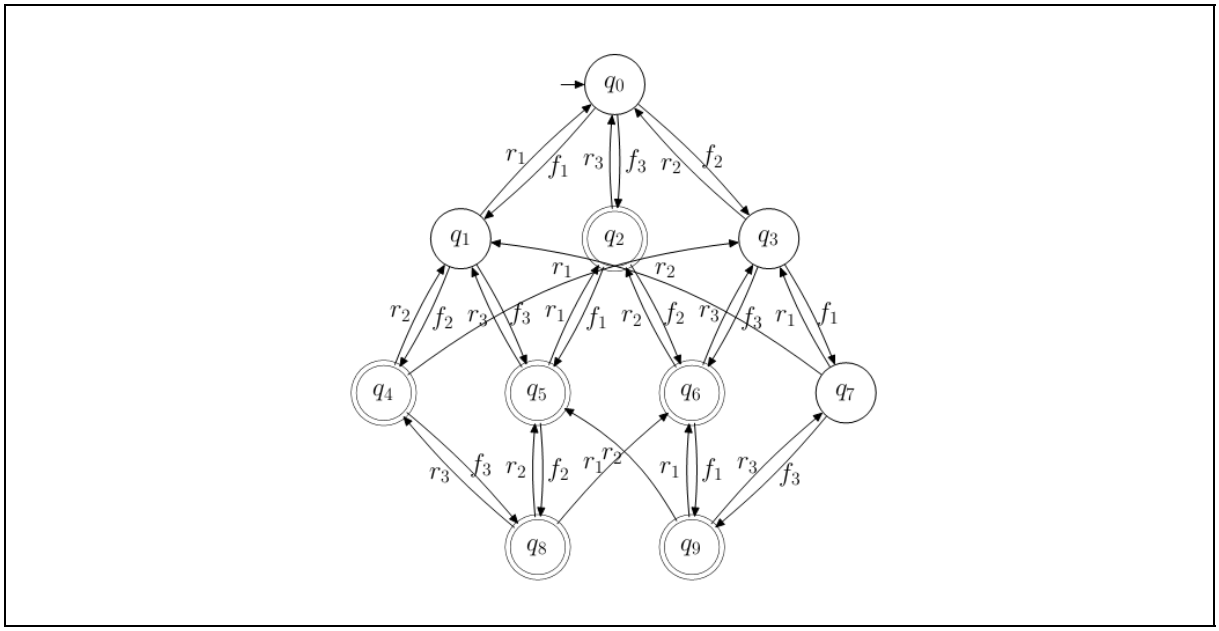


Figure 3.6. Modèle du comportement dynamique
d'un système physique à base de composants réparables

Dans cette modélisation, tout scénario représentant la vie du système correspond à une séquence σ, admissible par AF, d'événements de Σ à partir de l'état $q_0 \in Q$. Par construction, il s'agit du langage généré par AF que nous notons L_D (langage dysfonctionnel). Le langage marqué de AF, que nous notons L_F (langage défaillant), regroupe toutes les séquences conduisant à la défaillance du système. Le langage L_{CS} regroupant toutes les séquences de coupe, c'est-à-dire conduisant à la première défaillance du système est défini formellement par :

$$L_{CS} = \{\sigma \in L_F \mid (\forall \sigma_1 \in \text{Pref}(\sigma) - \{\sigma\}), \delta(q, \sigma_1) \notin Q_M\}$$

où $\text{Pref}(\sigma)$ est l'ensemble des préfixes de σ

L'ensemble recherché des séquences de coupe minimales SCM est par définition un sous-ensemble de L_{CS}.

Pour les modèles de fiabilité pour lesquels l'hypothèse de non occurrence simultanée d'événements de défaillance ou de réparation est retenue, l'obtention de cette représentation ne présente pas de difficultés théoriques comme nous avons pu le montrer pour les BDMP [C29].

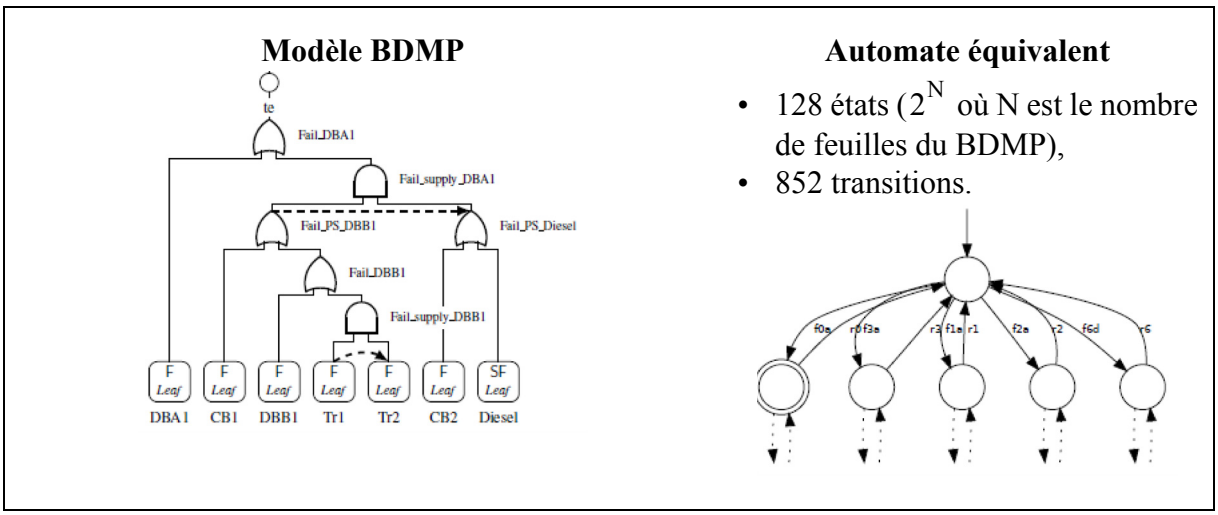


Figure 3.7. Expression du comportement d'un BDMP à l'aide d'un automate fini [C29]

Pour pourvoir établir une définition des séquences de coupe minimales basées sur la cohérence du système modélisé, nous avons tout d'abord montré que les règles de cohérence retenues pour les systèmes statiques pouvaient être étendues aux systèmes dynamiques après une reformulation en terme de séquences [T5].

La formalisation de ces règles à l'aide de la théorie des langages a permis d'établir que la connaissance d'un sous-ensemble de L_F était suffisante pour reconstruire tout le langage défaillant L_F grâce à la définition d'une fonction CR_4 matérialisant ces règles. Ce sous-ensemble de L_F, noté $Kern(L_F)$ constitue ainsi une représentation minimale du langage défaillant [C34].

La fonction CR_4 proposée admet en entrée un sous-ensemble S de séquences défaillantes ($S \subset L_F$) et permet de générer un sous-ensemble S' de séquences défaillantes ($S \subset S' \subset L_F$). Les séquences de S' sont obtenues en prolongeant une séquence σ de S par un événement de défaillance ou en distribuant les événements d'une séquence σ'' n'ayant pas d'influence sur la séquence σ, au sein de la séquence σ.

Le noyau $Kern(L_F)$ est défini comme suit :

$$\begin{cases} CR_4(Kern(L_F)) = L_F \\ \forall S \subset Kern(L_F), |S| < |Kern(L_F)| \rightarrow (CR_4(S) \neq L_F) \end{cases}$$

A partir de la définition de la fonction CR_4, différentes propriétés ont pu être démontrées pour $Kern(L_F)$. Il a été établi que [T5] :

- Aucune séquence de $Kern(L_F)$ ne peut générer une autre séquence de $Kern(L_F)$ à l'aide de CR_4,
- $Kern(L_F)$ est unique.
- $Kern(L_F)$ est un langage fini.

Grâce aux propriétés établies sur $\mathrm{Kern}(L_F)$, il a pu être démontré que l'ensemble recherché des séquences de coupe minimales correspondait à :

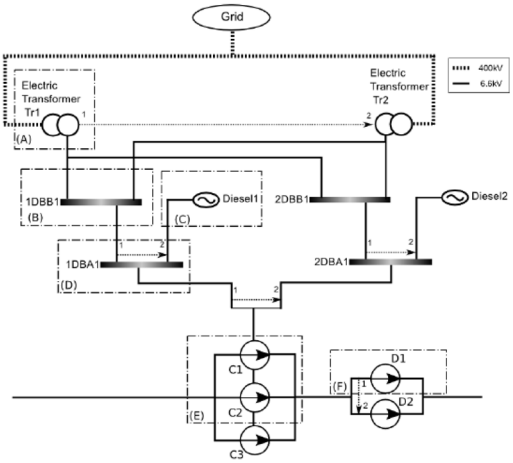
$$SCM = L_{CS} \cap \mathrm{Kern}(L_F)$$

Si la fonction CR_4 permet une définition formelle pour SCM, elle ne permet pas cependant de calculer directement SCM à partir du langage défaillant L_F car cette fonction n'est pas bijective. Il a donc été proposé de calculer $\mathrm{Kern}(L_F)$ à partir du couple (L_F, L_D) par identification et retrait des séquences du langage défaillant qui ne pouvait pas appartenir au noyau. Les algorithmes nécessaires à ce calcul sont détaillés dans [T5].

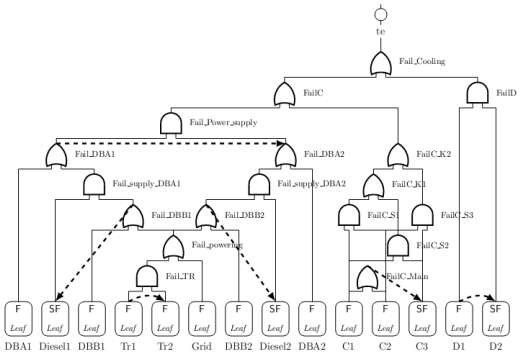
Toute cette approche a été expérimentée sur des études de cas proposées par EDF. L'implémentation des algorithmes proposés a permis le calcul de l'ensemble SCM recherché à partir de modèles BDMP. Ces expérimentations ont montrées la faisabilité de cette approche et permis la comparaison des résultats obtenus avec ceux fournis par les algorithmes présents dans la plateforme KB3.

La figure 3.8 présente l'une de ces études de cas. Il est donné l'installation électrique étudiée, le modèle BDMP proposé par EDF. A partir de l'automate fini représentant le BDMP, il a été possible de générer les séquences de coupe d'une longueur donnée et d'identifier les séquences de coupes minimales.

Installation électrique étudiée



Modèle BDMP de l'installation



Automate a état fini équivalent

- 16 384 états (2^{14}) dont 13 300 défaillants
- 229 376 transitions

- Séquences de défaillances :
  - Longueur 2 : 9
  - Longueur 3 : 255
  - Longueur 4 : 4 897
  - Longueur 5 : 79 594
  - Longueur 6 : 1 191 995

Séquence de coupes

- Longueur 2 : 9
- Longueur 3 : 172
- Longueur 4 : 2 226
- Longueur 5 : 23 458
- Longueur 6 : 215 451

Séquence de coupes minimales

- Longueur 2 : 9
- Longueur 3 : 19
- Longueur 4 : 39
- Longueur 5 : 17
- Longueur 6 : 45

Figure 3.8. Étude qualitative d'une installation électrique à l'aide de BDMP

Pour ce modèle, les travaux de Pierre-Chaux ont montré que les approximations retenues dans KB3 éliminaient à tort certaines séquences. L'implémentation de la méthode proposée a ainsi permis d'identifier des séquences de coupes minimales d'ordre 5 comportant des événements de réparation. De telles séquences étaient systématiquement éliminées par les algorithmes présents dans la plateforme KB3 qui exploitaient l'hypothèse simplificatrice retenant qu'une séquence de coupe minimale ne pouvait pas contenir d'événements de réparation.

3.3.3. Conclusions

En nous appuyant sur la théorie des langages, il nous a été possible de donner une définition formelle aux séquences de coupe minimales (SCM) basée sur la notion de cohérence des systèmes dynamiques réparables. Cette définition repose sur l'existence d'un noyau $Kern(L_F)$ pour le langage défaillant constituant la représentation minimale du langage défaillant. Les algorithmes permettant le calcul de $Kern(L_F)$ et de SCM reposant sur cette définition ont pu être établis et expérimentés sur des études de cas proposées par EDF.

Pour permettre à EDF d'exploiter efficacement ces premiers résultats, un travail d'optimisation des algorithmes reste cependant à faire afin d'intégrer au plus tôt les spécificités des modèles BDMP. En effet, les algorithmes établis dans ce travail de thèse ont été établis en respectant strictement le cadre mathématique choisi pour ce travail de formalisation. Pour une utilisation dédiée au BDMP, et au vue des résultats obtenus, il apparait qu'une modélisation à base d'une structure de Kripke étiquetée pourrait s'avérer plus efficace.

Lors de ce travail de formalisation, différentes propriétés portant sur la cohérence des modèles dynamiques réparables ont été établies. Dans notre approche, nous avons supposé que ces propriétés étaient toujours satisfaites. Cependant, comme un ingénieur fiabiliste est très peu assisté lors de l'établissement de ces modèles, les possibilités d'erreur sont nombreuses. Tout un travail sur la vérification de la cohérence d'un modèle dynamique est à réaliser. Il serait intéressant d'évaluer les possibilités offertes par le model-checking pour l'analyse de ce type de modèles.

Le travail de formalisation réalisé repose sur l'hypothèse de non simultanéité d'événements de panne et de réparation. Pour satisfaire cette hypothèse, nous avons du restreindre nos expérimentations aux BDMP ne faisant pas référence à des défaillances à la sollicitation (refus de démarrage d'un générateur entraîné par un moteur thermique). Pour prendre en compte de ce type défaillance, il sera certainement nécessaire de s'appuyer sur un cadre mathématique différent.

Ce partenariat avec EDF autour des BDMP peut donc se poursuivre dans plusieurs directions.

3.4. Analyse prévisionnelle des risques à l'aide de chaînes de Markov

Ces travaux sont menés dans le cadre d'une collaboration avec Antoine Rauzy aujourd'hui professeur à Centrale-Supelec et titulaire de la Chaire Blériot-Fabre en partenariat avec Safran. Cette collaboration a débutée lors du master de Pierre-Antoine Brameret [S10] et se poursuit dans le cadre de la co-direction de la thèse de Pierre-Antoine [T6] débutée en septembre 2012.

Ces travaux s'inscrivent aujourd'hui dans le cadre du projet AltaRica dont l'objectif est de développer un atelier logiciel permettant de mener des études de fiabilité sur des modèles de grandes tailles (Figure 3.9). Cet atelier logiciel sera mis gratuitement à la disposition de l'ensemble de la communauté dès que les principales briques seront développées. Cet outil est basé sur les « Guarded Transition Systems » [44].

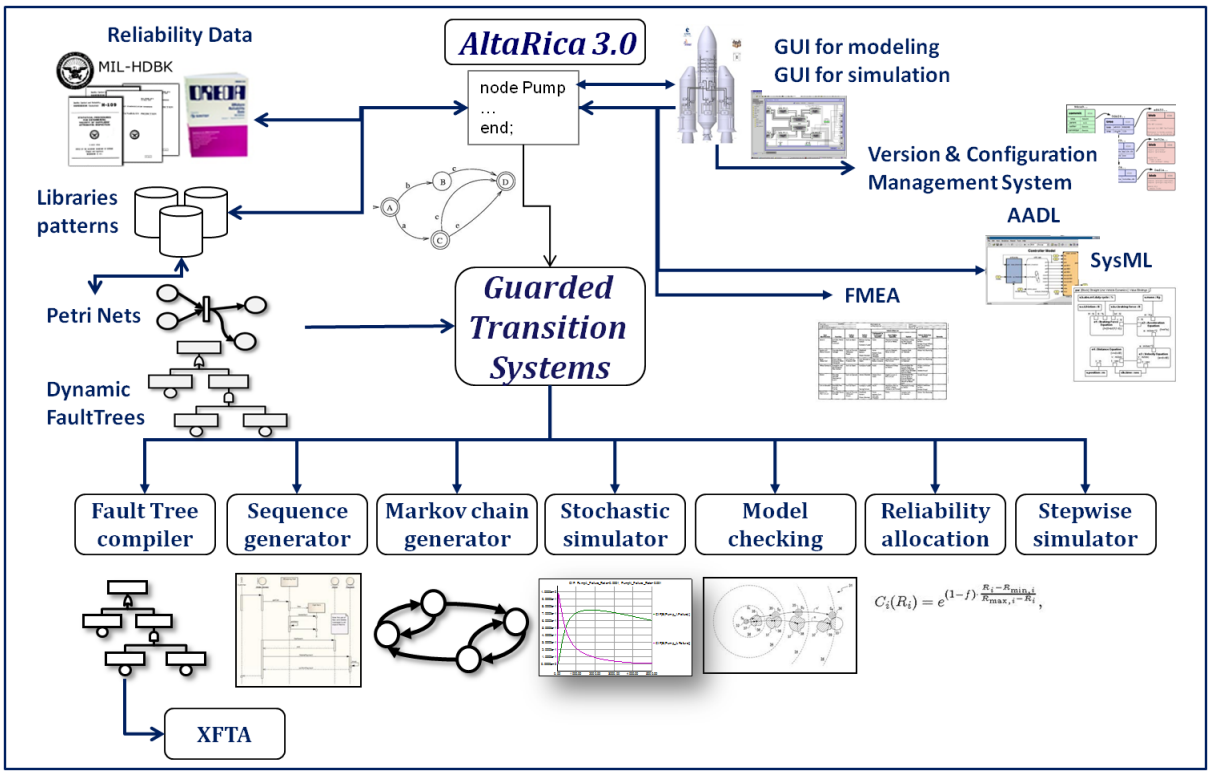


Figure 3.9. Projet AltaRica basé sur les « Guarded Transition Systems »

3.4.1. Positionnement scientifique

Si les arbres de défaillance restent largement présents dans les analyses de sûreté, il convient de souligner que leur génération se fait aujourd'hui de plus en plus de manière automatique à partir de modèles de fiabilité de plus haut niveau car il s'avère très difficile de les établir et les maintenir lorsque la taille et la complexité des systèmes étudiés augmentent [1].

D'un point de vue industriel, la constitution des arbres de défaillance d'une installation demeure une nécessité :

- Ils sont exigés par les autorités de régulation en tant que support du processus de certification.
- Ils constituent le seul modèle exploitable pour des analyses quantitatives lorsque la taille des modèles augmente.

Pour mener leurs analyses, les industriels utilisent donc des ateliers logiciels leur permettant de construire des modèles de haut niveau et d'obtenir les arbres de défaillance statiques corres-

pondants. Différents outils commerciaux proposent ce type de fonctionnement. Lors de la génération automatique d'un arbre de défaillance statique à partir d'un modèle de plus haut niveau, de nombreuses approximations conservatives doivent être faite afin que la perte d'information induite par cette transformation ne conduisent pas à une sous-approximation des risques. Ces approximations conservatives sont bien acceptées par la communauté. Cependant, leur emploi réduit les possibilités d'exploitation des données quantitatives. Par exemple, il est difficile de comparer plusieurs stratégies de redondance entre composants pour une même installation car les résultats fournis en utilisant les arbres de défaillance statiques sont sensiblement équivalents.

Pour des systèmes complexes, il est évident que l'analyse prévisionnelle des risques à l'aide de chaînes de Markov permet d'obtenir des résultats plus précis qu'avec un arbre de défaillance statique car le comportement décrit représente plus finement le comportement étudié. Cependant, lorsque cette chaîne de Markov est générée automatiquement à partir de modèles de fiabilité de haut niveau, sa taille rend très difficile voire impossible son évaluation. Le comportement complet d'un système physique comportant seulement 20 composants réparables peut donner lieu à une chaîne de Markov de plus 10^6 états.

L'évaluation d'une chaîne de Markov homogène à temps continu est généralement faite par traduction du processus markovien en un ensemble d'équations différentielles [50] décrivant les évolutions du temps de séjour dans les différents états. La résolution de ces équations peut être analytique [6] ou numérique [38] [48]. L'évaluation de chaînes de Markov de grandes tailles est un challenge important faisant l'objet de nombreux travaux visant à réduire la taille [23] [33] [43] [57] ou à l'évaluer de manière approchée [38]. Pour toutes ces méthodes, les données d'entrées sont les chaînes de Markov sans aucune autre information sur le système étudié. Sans informations complémentaires, ces méthodes doivent être générales et ne permettront pas de traiter des chaînes de Markov de la taille de celles obtenues à partir d'un modèle de fiabilité.

L'approche proposée est basée sur le fait que les chaînes de Markov sont obtenues à partir d'un modèle de plus haut niveau. Cette technique d'obtention donne à ces chaînes de Markov certaines spécificités qui méritent d'être exploitées :

- Le temps de séjour dans les différents états de la chaîne de Markov n'est pas réparti de manière équilibrée en raison des valeurs retenues pour les taux de défaillance ou de réparation associés aux composants ($\lambda \ll \mu$).
- Pour un système à base de composants réparables, la valeur du temps de séjour de l'état correspondant au fonctionnement nominal du système étudié est de loin la plus importante. Pour les autres états, cette valeur décroit rapidement lorsqu'on s'écarte de l'état correspondant au fonctionnement nominal.

Pour pouvoir étudier des systèmes de grande taille, nous proposons de ne construire que partiellement la chaîne de Markov correspondant au système étudié en privilégiant les états dans lequel le temps de séjour est à priori le plus important. Nous sommes en mesure de garantir l'erreur de calcul induite par cette construction partielle.

3.4.2. Détails des principaux résultats scientifiques obtenus

D'un point de vue théorique, il n'est pas possible de déterminer à priori avec exactitude la valeur du temps de séjour dans un état d'une chaîne de Markov. L'approche que nous suivons consiste seulement à estimer, à l'aide d'une heuristique, l'influence de chacun des états au moment de sa construction. Pour ce faire, nous associons à chacun des états une grandeur que nous avons nommée « Relevance factor ». En privilégiant la construction de la chaîne de Markov à

partir des états dont le « Relevance factor » est le plus grand, nous sommes en mesure de faire une construction partielle en ne retenant que les états estimés comme les plus influents.

La grandeur « Relevance factor » associée à chaque état de la chaîne de Markov est comprise entre 0 et 1. Cette grandeur est à 1 pour l'état correspondant au fonctionnement nominal du système étudié. Pour les autres états s_j, cette grandeur $R(s_j)$ est définie de la manière suivante :

$$R(s_j) = \mathrm{Max}_{s_i \in \text{ Etats Amonts}} \left(R(s_i) \times \frac{q_{i \to j}}{\sum_k q_{i \to k}} \right)$$

où $q_{i \to j}$ est le taux de transition entre l'état s_i et l'état s_j.

Sur le plan algorithmique, il est possible de déterminer la valeur de $R(s_i)$ sans avoir à établir entièrement la chaîne de Markov à la condition que cette valeur soit calculée de proche en proche à partir de l'état correspondant au fonctionnement nominal du système étudié ($R(s_{Init}) = 1$) et en privilégiant systématiquement l'analyse depuis l'état où cette valeur est maximale. Cela est du aux deux aspects suivants :

- La valeur $R(s_j)$ pour un état s_j est la valeur maximale des contributions apportées par ses transitions situées en amont. Il n'est pas nécessaire de connaître chacune des contributions.
- La contribution de la transition $s_i \to s_j$ est égale au produit de $R(s_i)$ par une grandeur inférieure à 1. L'apport des transitions est de plus en plus faible.

Un algorithme basé sur l'algorithme de Dijkstra [18] a été proposé pour permettre la construction partielle de la chaîne de Markov [C35] suivant notre heuristique. Il a été ensuite étudié à partir de quelle valeur seuil $R(s_j)$ le résultat calculé à l'aide de la chaîne réduite permet de donner une bonne approximation du résultat calculé avec la chaîne complète. Il est apparu, à travers le traitement de différents exemples, qu'il suffisait de conserver les états s_j pour lesquels la valeur de $R(s_j)$ était supérieure à 10^{-4} pour obtenir l'indisponibilité d'une installation avec une erreur relative inférieure à 1%.

Pour illustrer ce propos, considérons le système présenté sur la figure 3.10 tiré de la littérature [36] et qui a servi de benchmark dans [40] pour les logiciels Galileo [20], DBNet [39] et DRPFTproc [40]. L'étude porte sur le taux d'indisponibilité d'un module informatique comportant des composants redondants non réparables. Il est comparé les résultats quantitatifs obtenus par les trois logiciels pour cinq dates données (de 1 000 h à 5 000 h).

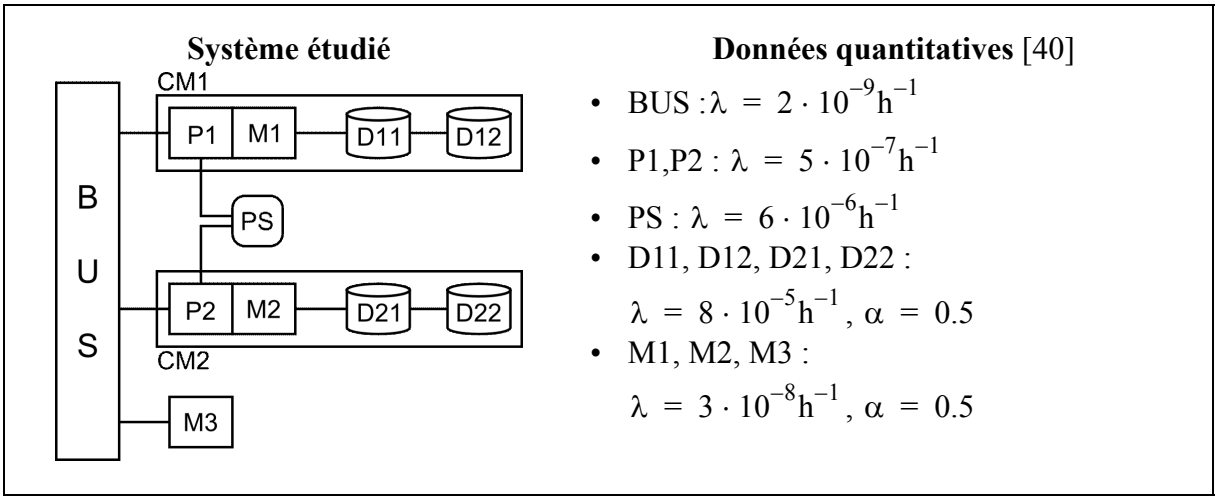


Figure 3.10. Description du benchmark proposé dans [40]

Pour cette étude, le système a été modélisé à l'aide du langage AltaRica. La chaîne de Markov complète correspondante comporte 17 152 transitions[1]. Il a été tout d'abord comparé les résultats obtenus avec cette chaîne de Markov en calculant l'indisponibilité pour les cinq dates considérées de l'étude. Les résultats obtenus sont similaires à ceux fournis par les autres outils.

Disposant d'un modèle de référence pour lequel l'indisponibilité de l'installation pouvait être calculée à notre convenance, nous avons étudié l'évolution au cours du temps de l'erreur introduite par notre heuristique. La chaîne de Markov réduite en ne conservant que les états s_j pour lesquels la valeur de $R(s_j)$ était supérieure à 10^{-4} comporte seulement 470 transitions. La courbe présentée figure 3.11 correspond à l'évolution de l'erreur relative faite pour la valeur de l'indisponibilité du système sur une plage de temps de 0 à 50 000h en effectuant 1 000 mesures équi-réparties. Pour ce système, l'indisponibilité peut être calculée avec une erreur relative inférieure à 0, 25 % à l'aide d'une chaîne de Markov de 470 transitions au lieu d'une chaîne de Markov de 17 152 transitions.

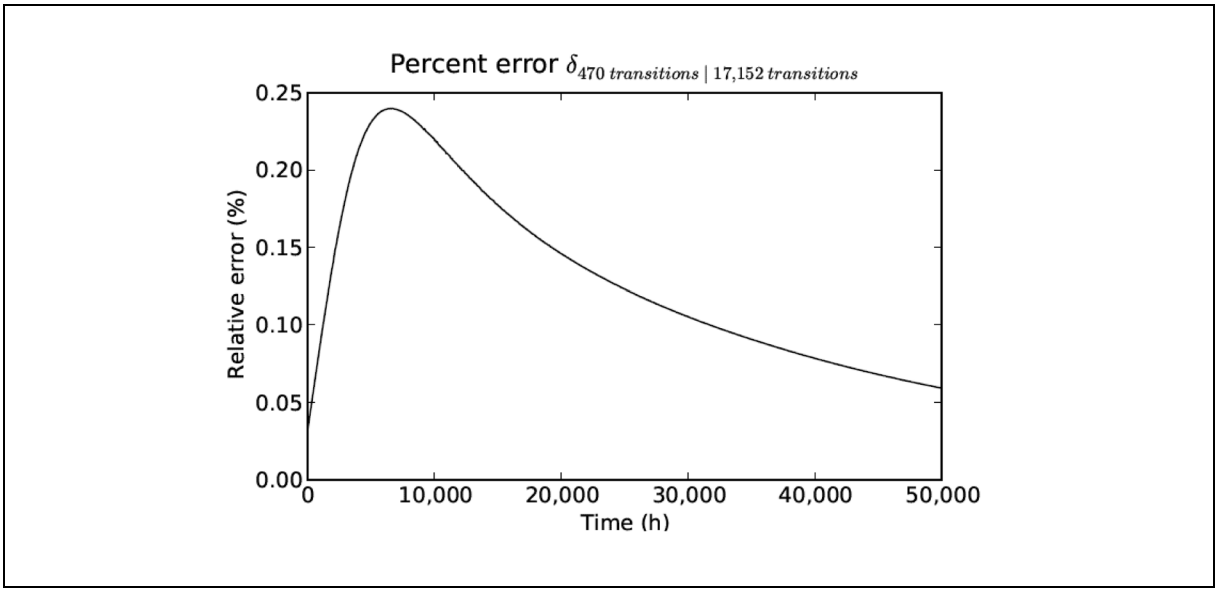


Figure 3.11. Évolution de l'erreur relative entre les 2 chaînes de Markov

Des expérimentations sur un système comportant 21 composants réparables ont montré que son indisponibilité pouvait être calculée à l'aide d'une chaîne de Markov de 5 713 transitions au lieu d'une chaîne de Markov de 10 768 622 transitions avec une erreur relative inférieure à 1%. Le temps de calcul de l'indisponibilité est de 4,5 s au lieu de 3,3 h.

Si ces essais montrent toute la pertinence de l'heuristique proposée, la précision de la valeur de l'indisponibilité calculée à l'aide de la chaîne de Markov approximée ne peut être garantie que si la chaîne de Markov complète peut être elle aussi calculée. Pour que cette nouvelle méthode de calcul soit acceptée par un ingénieur-fiabiliste, il est nécessaire que l'erreur induite par cette approximation soit quantifiable, même grossièrement sans avoir à la comparer avec la valeur calculée pour la chaîne de Markov complète.

Pour être capable de borner l'erreur introduite par l'usage d'un tel heuristique, il a été nécessaire de recourir à une nouvelle chaine de Markov réduite. Les états non retenus par l'algorithme précédent sont maintenant fusionnés en un seul état puits (Figure 3.12).

1. Pour ces travaux, la taille d'une chaîne de Markov est caractérisée par son nombre de transitions car le temps nécessaire à son évaluation à l'aide de l'outil utilisé dépend linéairement de ce paramètre [48].

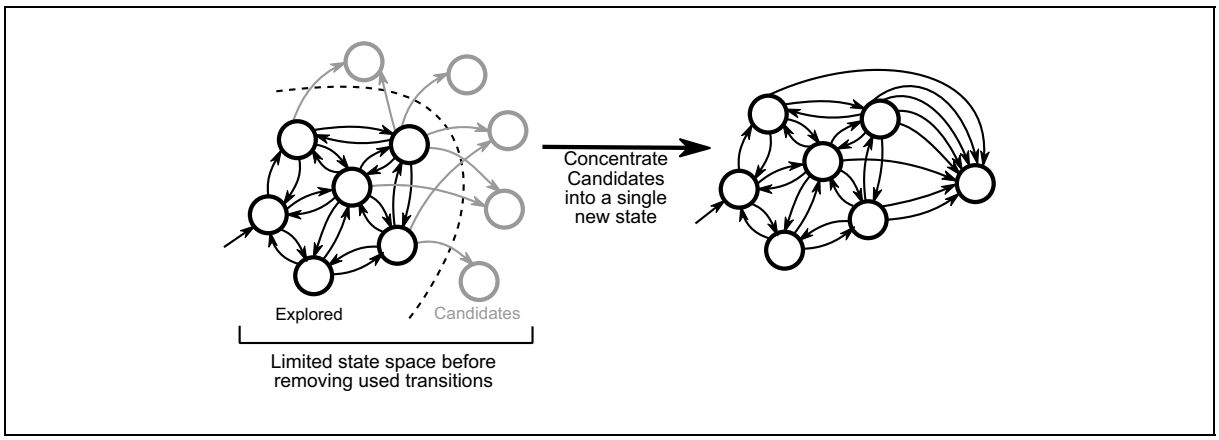


Figure 3.12. Stratégie suivie pour l'obtention de la chaîne de Markov réduite à erreur garantie

En considérant cet état comme défaillant, nous sommes certains de faire une sur-approximation de l'indisponibilité du système étudié. En considérant cet état comme fonctionnel, nous sommes certains de faire une sous-approximation de cette indisponibilité.

Le temps de séjour dans cet état puits est donc un majorant de l'erreur induite par l'utilisation de la chaîne de Markov réduite. Comme il s'agit d'un état puits, le temps de séjour dans cet état est strictement croissant. L'erreur induite par cette approximation doit être calculée pour la date considérée de l'étude. Il convient de signaler qu'il s'agit cependant d'un majorant très grossier de cette erreur. Pour garantir que l'erreur induite est inférieure à un seuil donné, il est nécessaire de retenir plus d'états que dans le cas précédent.

Nous sommes aujourd'hui en mesure de majorer l'erreur de calcul induite par l'utilisation de cette heuristique pour des systèmes dont la taille ne permet plus de générer la chaine de Markov complète. Pour de tels systèmes, nous proposons au concepteur de fixer un nombre de transitions autorisées pour la chaîne de Markov partielle. Celle-ci est alors construite et évaluée pour la durée considérée de l'étude. Il est également calculé, à l'aide du temps de séjour dans l'état puits, un majorant de l'erreur relative induite par cette génération partielle. Si cette valeur est trop grande, le concepteur peut procéder à une nouvelle génération pour la chaîne de Markov partielle en retenant plus de transitions.

3.4.3. Conclusions

Lorsque la taille des systèmes étudiés augmente, l'évaluation de l'indisponibilité d'une installation demande l'introduction d'approximations pour que le traitement des modèles établis soit numériquement possible. Ces approximations sont généralement faites en repassant par des arbres de défaillance statiques. Pour conserver la possibilité d'utiliser une chaîne de Markov pour l'évaluation de l'indisponibilité d'une installation, nous avons établi une heuristique permettant de donner une sur-approximation garantie de cette indisponibilité calculée à l'aide d'une chaîne de Markov réduite. Tout l'intérêt de cette approche est d'éviter de construire l'espace d'état complet pour le système étudié. Cette approche peut être utilisée pour d'autres modèles de fiabilité.

La pertinence de cette approche a été validée au travers du traitement de différentes études de cas de taille et de complexité croissantes. Cette stratégie peut être aujourd'hui directement expérimentée sur des modèles exprimés en AltaRica grâce aux derniers développements informatiques réalisés par Pierre-Antoine Brameret. Ils permettent de montrer par l'exemple tout le

potentiel de cette approche basée entièrement sur une heuristique permettant d'estimer a priori l'influence de chacun des états de la chaîne de Markov.

Nous travaillons actuellement à des techniques d'optimisation du choix des états de la chaine de Markov réduite dans le cas ou le majorant proposé pour l'erreur ne satisferait pas l'ingénieur-fiabiliste. Nous envisageons pour cela de mettre en place un processus itératif qui intégrera cette heuristique et les résultats obtenus lors de l'évaluation des chaînes de Markov réduites.

Chapitre 4

Conclusions et Perspectives

4.1. Conclusions

Les travaux et résultats présentés dans ce mémoire ont été conduit avec la conviction de mener une recherche académique de qualité tout en répondant à des problèmes issus du monde industriel.

Pour la conception sûre des systèmes de contrôle-commande, les questions auxquelles il a pu être apporté des réponses scientifiques ont été :

- Le programme de la commande satisfait-il les attentes pour lesquelles il a été fait ?
- Quels sont les modèles de commande d'un système logique qui satisfont un ensemble donné d'exigences ?
- Le comportement réel du contrôleur de cette commande correspond-il au comportement spécifié ?

Pour la formalisation des outils pour les analyses de sûreté, ces questions ont été :

- A quelle condition se produit la défaillance d'un système constitué de composants non réparables ? Peut-on exprimer cette condition algébriquement dans le cas d'un système dynamique ?
- Quelles sont les caractéristiques des séquences de défaillance qui conduisent à la défaillance d'un système dynamique constitué de composants non réparables ?
- Comment continuer à quantifier finement la défaillance d'une installation lorsque le nombre de composants augmente ?

Pour répondre à l'ensemble de ces questions, nous nous sommes appuyés sur les paradigmes et les concepts des SED. Pour certaines questions, les cadres théoriques existants ont permis de formaliser le problème et d'en rechercher une solution. Nos efforts se sont alors concentrés sur la recherche des moyens les plus pertinents pour obtenir des résultats fiables. Pour d'autres questions, il nous a fallu construire de nouveaux cadres théoriques adaptés à la classe de problèmes traités. Ces cadres ont été essentiellement des structures algébriques dédiées dans lesquelles une partie des réponses ont été obtenues à l'aide de calcul symbolique.

L'un des points communs à tous ces travaux est la sensibilité du problème traité à l'explosion combinatoire liée à la taille des modèles manipulés. Conscient de ce risque, il a toujours été recherché, lorsque cela était possible, à intégrer cette difficulté lors des études théoriques afin de ne pas se retrouver avec une réponse théorique dont la mise en œuvre n'était techniquement pas envisageable.

La majorité des recherches présentées dans ce mémoire ont donné lieu au développement de maquettes informatiques permettant de traiter des exemples significatifs en complexité et en taille. Il est évident que le développement de telles maquettes est chronophage et se fait toujours au détriment d'autres activités plus théoriques. Cette activité est cependant très bénéfique au travail de recherche car elle permet de montrer tout le potentiel des résultats théoriques, d'obtenir un premier retour sur l'applicabilité des travaux tout en affinant si nécessaire les stratégies mises en œuvre. Pour ces problématiques issues du monde industriel, ses maquettes informatiques sont également forte utiles pour intéresser de potentiels partenaires industriels dans la poursuite des travaux.

4.2. Perspectives

Durant ces années de recherche au LURPA, j'ai pu m'intéresser à la conception sûre des systèmes de contrôle-commande et à la formalisation des outils pour les analyses de sûreté. J'ai eu la possibilité d'encadrer des travaux de thèses en nombre équilibré dans chaque thématique.

Pour chacune des recherches présentées, des travaux complémentaires sont nécessaires afin de repousser les limites théoriques ou opérationnelles identifiées. Cependant ces recherches ne peuvent pas toutes se dérouler dans le cadre d'une thèse car bien que scientifiquement intéressantes, certaines d'entre-elles ne permettront pas à un doctorant de s'épanouir scientifiquement tout en garantissant une valorisation suffisante de son travail. De plus, pour certaines recherches, les possibilités de financement pour le doctorant et son environnement vont s'avérer difficiles à trouver. C'est avec ces contraintes à l'esprit que ce projet de recherche a été construit.

Le premier projet dans lequel je souhaite m'investir est la vérification formelle de modèles dynamiques utilisés en sûreté de fonctionnement. Lors des travaux précédents, je me suis aperçu combien il était difficile d'établir de tels modèles que cela soit à l'aide d'arbres de défaillance

dynamiques, de BDMP ou de modèles AltaRica. Le développement d'un outil de vérification formelle par model-checking fait d'ailleurs partie des actions envisagées dans le cadre du projet AltaRica (Figure 3.9). J'ai participé l'an dernier à une première recherche de financement au travers du dépôt du dossier DesiRe dans le cadre d'un appel ANR. La reconnaissance, au premier trimestre 2014, par l'IRT System X rattachée à l'université Paris-Saclay de ce projet devrait permettre la mise en place du cadre nécessaire à son aboutissement.

Au vue des résultats obtenus lors de la vérification formelle de logiciels de contrôle-commande, la mise en place d'un outil de model-checking adapté au langage AltaRica présente plusieurs verrous scientifiques :

- La première étude à mener doit porter sur le choix de la structure mathématique à retenir. Les « Guarded Transistions Systems » qui sont le modèle théorique sur lequel repose le projet AltaRica correspondent à une modélisation à base d'une structure de Kripke étiquetée. Cependant, les algorithmes de model-checking sont basés sur des structures de Kripke non étiquetées et leur performance repose sur cet aspect. Doit-on adapter les algorithmes de model-checking au traitement des structures de Kripke étiquetées ou exprimer les « Guarded Transistions Systems » à l'aide de structures de Kripke non étiquetées ? Ne faut-il pas envisager de travail avec autant de structures de Kripke qu'il y a d'étiquettes à intégrer et ainsi concilier les deux aspects ?
- La seconde étude à mener doit porter sur la stratégie à retenir pour prendre en compte l'interprétation retenue pour l'évaluation des « Guarded Transistions Systems » et plus particulièrement des assertions. En retenant une évaluation des assertions par recherche d'un point fixe pour les variables de type flow, il a été choisi une interprétation qui simplifie grandement l'élaboration des modèles mais qui en complexifie leur interprétation. Peut-on simplifier cette recherche de point fixe en intégrant une phase de réécriture des conditions d'affectation de ces variables afin de simplifier le travail du model-checker ?
- Il sera également intéressant d'étudier l'intégration du processus de validation de modèles au sein du processus même de conception des modèles afin de maîtriser la taille de la structure de Kripke à manipuler. Pour être efficace, il est nécessaire de proposer aux ingénieurs de valider des portions de modèles au plus tôt. Le langage AltaRica propose la conception de modèles par instanciation d'objets dont le comportement générique est décrit à l'aide d'une classe d'objets. Il serait judicieux d'exploiter ce mécanisme pour accroître les performances du processus de validation.
- En complément de ces premières études, il sera nécessaire de se pencher sur l'expression des propriétés à vérifier. Par expérience, je suis convaincu qu'un des freins à l'utilisation des outils de Model-checking repose sur la nécessité d'avoir à recourir aux logiques temporelles pour l'expression des propriétés. Pour permettre à des ingénieurs de sûreté de valider leurs propres modèles, il faut leur faciliter l'expression des propriétés. Pour quelles familles de propriétés cela est-il possible ? Peut-on reproduire le principe des automates observateurs en utilisant des « Guarded Transistions Systems » pour les exprimer ?

J'aimerai également continuer à m'investir dans le domaine de la conception sûre des systèmes de contrôle-commande. Il serait pertinent de profiter des compétences acquises en poursuivant certains des travaux dans le cadre d'une thèse CIFRE. Le partenaire industriel idéal est un industriel de l'offre comme Schneider ou Siemens ou de grands groupes comme Alstom, Areva ou la SNCF, qui ont les moyens de développer leur propre produit.

La conception sûre d'un système de contrôle-commande nécessite l'utilisation de modèles parfaitement définis, d'outils d'analyse de ces modèles mais également de méthodes rigoureuses pour les élaborer. Aujourd'hui, les modèles utilisés sont de mieux en mieux définis. Les outils théoriques d'analyse existent mais restent encore hors de la portée des utilisateurs. C'est en agissant sur le processus de conception qu'il sera possible de donner à un concepteur les moyens d'analyser ses propres modèles.

Au vue des résultats obtenus par la communauté dans ce domaine, je considère qu'il est maintenant nécessaire de personnaliser les prochains travaux à un secteur d'activité donné, voire une pratique industrielle afin de tenir compte du processus de conception mis en place. Pour que l'usage des méthodes formelles se développe dans l'industrie, il faut faire évoluer progressivement le processus de conception propre à chaque société en apportant aux utilisateurs les outils dont ils ont besoin pour chacune des étapes du processus. Le second projet dans lequel je souhaite m'investir est l'amélioration du processus de conception d'un système de contrôle-commande grâce à l'utilisation des méthodes formelles.

Dans le cadre d'un partenariat avec un grand groupe comme Alstom, Areva ou la SNCF, le travail préparatoire à la définition d'un sujet de thèse sera l'analyse du processus de conception retenue par cette société et de sa volonté à le faire évoluer en tenant compte des attentes des utilisateurs. L'usage des méthodes formelles pour le développement de la commande d'un système nécessite un fort investissement de la part des concepteurs. Pour que les évolutions nécessaires soient acceptées, il faut que le concepteur puisse bénéficier au plus tôt des possibilités offertes par les outils logiciels supports de ces méthodes formelles, en étant assisté lors de l'élaboration des modèles par exemple.

La mise en œuvre des méthodes formelles ne peut se faire sans l'emploi d'un atelier logiciel permettant à l'utilisateur d'éditer, de simuler, d'analyser le modèle établi. Dans le cas d'un partenariat avec une société qui développe son propre atelier logiciel, une réflexion basée sur la définition de modules d'assistance aux concepteurs devrait être accueillie favorablement.

Sur le plan scientifique, le premier travail à réaliser sera l'analyse du processus de conception retenu dans la société et des différents outils de modélisation utilisés afin d'identifier les étapes où les méthodes formelles peuvent apporter le plus d'assistance aux utilisateurs. Il sera également nécessaire d'analyser le travail du concepteur lors de l'élaboration des modèles pour distinguer les phases de création du modèle des phases d'enrichissement de celui-ci. C'est lors des dernières phases d'enrichissement du modèle que les outils support des méthodes formelles peuvent apporter le plus d'assistance aux concepteurs. Par exemple, la connaissance de l'espace d'état atteignable à partir d'un état donné d'un modèle pour une profondeur spécifiée doit permettre au concepteur de disposer des éléments pour critiquer son modèle. Il peut être également envisagé que le modèle proposé soit partiellement complété automatiquement. Cela doit être aujourd'hui possible en exploitant de manière simultanée les résultats obtenus par la communauté pour la synthèse de réseaux de Petri avec ceux obtenus en synthèse algébrique.

En bénéficiant concrètement des possibilités offertes par les outils support des méthodes formelles, le concepteur fera naturellement évoluer son processus de conception pour tirer profit des méthodes formelles.

Pour mener à bien de tels travaux dans le domaine de la conception sûre des systèmes de contrôle-commande, la première étape sera de convaincre un partenaire industriel de la nécessité d'investir dans son processus de conception et des bénéfices qu'il pourra en tirer.

Références bibliographiques

[1] R. Adeline, Méthodes pour la validation de modèles formels pour la sûreté de fonctionnement et extension aux problèmes multi-physiques, Thèse de l'Institut Supérieur de l'Aéronautique et de l'Espace, 131 pages, 14 mars 2011

[2] S. Balemi, Control of Discrete Event Systems: Theory and Application, Ph.D. dissertation, PhD thesis, Swiss Federal Institute of Technology Zurich, 1992.

[3] B. Bérard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, P. Schnoebelen, Systems and Software Verification: Model-Checking Techniques and Tools, 1st Edition, Springer Publishing Company, 1999

[4] Z. Birnbaum, J. Esary, S. Saunders (1961). Multi-component systems and structures and their reliability, Technometrics, vol. 3, no. 1, pp. 55-77, 1961

[5] A. Bobbio, D. Codetta Raiteri, Parametric fault trees with dynamic gates and repair boxes, Proceedings of the annual reliability and maintainability symposium (RAMS2004), LosAngeles, CA, USA, p.459–65, 2004

[6] G. Bolch, S. Greiner, H. de Meer, K. S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley-Interscience, 2006

[7] H. Boudali, J.B. Dugan, A discrete-time bayesian network reliability modeling and analysis framework, Reliability Engineering and System Safety, vol. 87, no. 3, pp. 337-349, 2005

[8] H. Boudali, J.B. Dugan, A continuous-time bayesian network reliability modeling and analysis framework, IEEE Transactions on Reliability, vol. 55, no. 1, pp. 86-97, 2005

[9] H. Boudali, P. Crouzen, M. Stoelinga, Dynamic fault tree analysis through input/output interactive Markov chains, Proceedings of the international conference on dependable systems and networks (DSN2007), IEEE Computer Society, pp.25–38, 2007

[10] H. Boudali, P. Crouzen, M. Stoelinga, A compositional semantics for dynamic fault tree in terms of interactive Markov chains, Lecture notes in computer science, vol. 4762, pp. 441–456, 2007

[11] H. Boudali,P. Crouzen, M. Stoelinga, A rigorous compositional and extensible framework for dynamic fault tree analysis, IEEE Transactions on Dependable and Secure Computing, vol. 7, no. 2, pp. 128-143, 2010

[12] M. Bouissou, J. Bon, J. (2003). A new formalism that combines advantages of fault-trees and Markov models: Boolean logic Driven Markov Processes, Reliability Engineering and System Safety, vol. 82, no. 2, pp. 149-163, 2003

[13] M. Bouissou, Détermination efficace de scenarii minimaux de défaillance pour des systèmes séquentiels, In 15ième colloque de fiabilité et maintenabilité, Lille, 2006

[14] F. M. Brown, Boolean Reasoning: The Logic of Boolean Equations, Dover Publications, 2003

[15] D. Codetta Raiteri, The conversion of dynamic fault trees to stochastic Petri nets, as a case of graph transformation, Electronic Notes on Theoretical Computer Science, vol. 127, no. 2, pp. 45–60, 2005

[16] D. Darvas, B. Fernandez, E. Blanco, Transforming PLC programs into formal models for verification purposes. Internal note, CERN. CERN-ACC-NOTE-2013-0040, 2013

[17] G. Dantzig, R. Fulkerson, and S. Johnson, Solution of a large-scale traveling-salesman problem," Journal of the Operations Research Society of America, vol. 2, no. 4, pp. 393–410, 1954

[18] E. Dijkstra, A note on two problems in connexion with graphs, Numerische mathematik, vol. 1, no. 1, pp. 269-271, 1959

[19] J. B. Dugan, S.J. Bavuso, M.A. Boyd, Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability, vol. 41, no. 3, pp. 363–377, 2000

[20] J. B. Dugan, K. J. Sullivan, D. Coppit, Developing a low-cost high-quality software tool for dynamic fault-tree analysis, IEEE Transactions on Reliability, vol. 49, no. 1, pp. 49–59, 2000

[21] M Fabian, A Hellgren, PLC-based implementation of supervisory control for discrete event systems, 37th IEEE Conference on Decision and Control, vol. 3, pp 3305-3310, 1998.

[22] A. Fernandes Pires, Apports des méthodes formelles pour les cycles de développement logiciel embarqué basés sur le modèle en V, Conférence en IngénieriE du Logiciel (CIEL 2012), pp. 1-6, 2012

[23] J. M. Fourneau, N. Pekergin, S. Youns, Censoring markov chains and stochastic bounds. Lecture Notes in Computer Science, vol. 4748, pp. 213-227, 2007

[24] V. Gourcuff, Représentations formelles efficaces pour l'aide à la certification de contrôleurs logiques industriels, Thèse de l'ENS Cachan, 131 pages, 17 décembre 2007

[25] D. Gouyon, J. Petin, A. Gouin, Pragmatic approach for modular control synthesis and implementation, International Journal of Production Research, vol. 42, no. 14, pp. 2839–2858, 2004

[26] D. Gouyon, Contrôle par le produit des systèmes d'exécution de la production : apport des techniques de synthèse, Thèse de l'Université Henri Poincaré, 164 pages, 6 décembre 2004

[27] R. P. Grimaldi, Discrete and Combinatorial Mathematics: An Applied Introduction, Fifth Edition, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2004

[28] D. A. Huffman, The synthesis of sequential switching circuits., J. of the Franklin Institute 257 (3-4) (1954) 161–190 and 275–303

[29] IEC 60848, IEC 60848 Standard: GRAFCET specification language for sequential function charts, 2nd ed. International Electrotechnical Commission, 2002

[30] IEC 61131-3, IEC 61131-3 Standard: Programmable controllers - Part 3: Programming languages, 2nd Edition, International Electrotechnical Commission, 2003

[31] IEC 61508, IEC 61508 Standard: Functional safety of electrical/electronic/ programmable electronic safety-related systems, 2nd Edition, International Electrotechnical Commission, 2010

[32] T. Jéron, Contribution à la génération automatique de tests pour les systèmes réactifs, Habilitation à diriger des recherches, Université de Rennes 1, France, 2004

[33] R. Lal, U. Bhat, Reduced system algorithms for markov chains. Management Science, vol. 34, no. 10, pp. 1202-1220, 1988

[34] D. Lee, M. Yannakakis. Principles and methods of testing finite state machines - a survey. Proceedings of the IEEE, vol. 84, pp. 1090–1123, 1996

[35] M. Lucas and D. Tilbury, A study of current logic design practices in the automotive manufacturing industry, International Journal of Human-Computer Studies, vol. 59, no. 5, pp. 725–753, 2003

[36] M. Malhotra, K. S. Trivedi, Dependability modeling using Petri-nets. IEEE Transactions on Reliability, vol. 44, no. 3, pp. 428-440, 1995

[37] K. Mei-Ko, Graphic programming using odd or even points, Chinese Mathematics, 1, 273–277, 1962

[38] S. Mercier, Bounds and approximations for continuous-time markovian transition probabilities and large systems. European Journal of Operational Research, vol. 185, no. 1, pp. 216-234, 2008

[39] S. Montani, L. Portinale, A. Bobbio ,M. Varesio, D. Codetta-Raiteri, DBNet, a tool to convert dynamic fault trees into dynamic Bayesian networks, Università del Piemonte Orientale, Technical Report TR-INF-2005-08-02-UNIPMN, 2005

[40] S. Montani, L. Portinale, A. Bobbio, M. Varesio, D. Codetta-Raiteri, A tool for automatically translating dynamic fault trees into dynamic bayesian networks, Reliability and Maintainability Symposium (RAMS 2006), pp. 434-441, 2006

[41] G. H. Mealy, A method for synthesizing sequential circuits, Bell System Technical Journal 34 (5) (1955) 1045–1079

[42] E. F. Moore, Gedanken Experiments on Sequential Machines, in: Automata Studies, Princeton U., 1956, pp. 129–153

[43] Y. Pribadi, J. Voeten, B. Theelen, Reducing Markov chains for performance evaluation. Proceedings of PROGRESS'01, Utrecht, The Netherlands pp. 173-179, 2001

[44] T. Prosvirnova, A. Rauzy, AltaRica 3.0 project: compile Guarded Transition Systems into Fault Trees, 22th European Safety & Reliability Conference (ESREL'13), Amsterdam (The Netherlands), pp. 1121-1128, September 2013

[45] P. Ramadge, W. Wonham, Supervisory control of a class of discrete event processes, SIAM Journal on Control and Optimization, vol. 25, no. 1, pp. 206–230, 1987

[46] K. A. Reay, J.D. Andrews, A fault tree analysis strategy using binary decision, diagrams, Reliability Engineering and System Safety, vol. 78, no. 1, pp. 45–56, 2002

[47] A. Rauzy, Mathematical foundations of minimal cut sets, IEEE Transactions on Reliability, vol. 50, no. 4, pp. 389–396, 2001

[48] A. Rauzy, An experimental study on iterative methods to compute transient solutions of large markov models. Reliability Engineering and System Safety, vol. 86, no. 1, pp. 105-115, 2004

[49] M. Stamatelatos, W. Vesely, Fault tree hand book with aerospace applications, NASA Office of Safety and Mission Assurance, 205 pages, 2002

[50] W. Stewart, Introduction to the numerical solution of Markov chains, volume 1. Princeton University Press, 1994

[51] Z. Tang, J.B. Dugan, Minimal cut set/Sequence generation for dynamic fault trees, Proceedings of the annual reliability and maintain ability symposium (RAMS2004), Los Angeles, USA, pp. 207–213, 2004

[52] J. Tretmans, Model based testing with labelled transition systems, Lecture Notes in Computer Science, vol. 4949, pp. 1–38, 2008

[53] W. Vesely, F.F. Goldberg,N.H. Roberts,D.F. Haas, Fault tree hand book, US Nuclear Regulatory Commission, 1981

[54] G. von Bochmann et G.-V. Jourdan. Testing k-safe Petri nets, Lecture Notes in Computer Science, vol. 5826, pp. 33–48, 2008

[55] M. Walker, Y. Papadopoulos, Qualitative temporal analysis: towards a full implementation of the fault tree handbook, Control Engineering Practice, vol. 17, no. 10, pp. 1115–1125, 2009

[56] W. Yi, F. Xing-hua, W. Dai-qiang, An implementation of random single input change technique for low-power test, Proceeding of the 2nd International Conference on Anti-counterfeiting, Security and Identification, pp. 352-355, 2008

[57] Y. Q. Zhao, D. Liu, The censored markov chain and the best augmentation, Journal of Applied Probability, vol. 33, no. 3, pp. 623-629, 1996

Liste des figures

Annexes

Publication relative à la
Vérification formelle de programmes de contrôle-commande établis pour des APIs

- Verification of a timed multitask system with Uppaal
  H. Bel Mokadem, B. Bérard, V. Gourcuff, O. De Smet, J.-M. Roussel
  IEEE Transactions on Automation Science and Engineering, 7(4), pp. 921-932, 2010

Publication relative à la
Synthèse algébrique de programmes de contrôle-commande à partir de spécifications informelles

- Design of logic controllers thanks to symbolic computation of simultaneously-asserted Boolean equations
  J.-M. Roussel, J.-J. Lesage
  Mathematical Problems in Engineering, 2014 (Article ID 726246) 15 pages, accepté le 7 février 2014

Publication relative au
Test de conformité d'un contrôleur logique vis-à-vis de sa spécification

- Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers
  J. Provost, J.-M. Roussel, J.-M. Faure
  IEEE Transactions on Industrial Informatics, 9 pages, accepté le 28 mars 2014

Publication relative à la
Modélisation algébrique des arbres de défaillance dynamiques

- Algebraic determination of the structure function of dynamic fault trees
  G. Merle, J.-M. Roussel, J.-J. Lesage
  Reliability Engineering and System Safety, 96(2), pp. 267-277, 2011

Verification of a Timed Multitask System With UPPAAL

Houda Bel Mokadem, Béatrice Bérard, Vincent Gourcuff, Olivier De Smet, and Jean-Marc Roussel

***Abstract*—System and program verification has been a large area of research since the introduction of computers in industrial systems. It is an especially important issue for critical systems, where errors can cause human and financial damages. Programmable Logic Controllers (PLCs) are now widely used in many industrial systems and verification of the corresponding programs has already been studied in various contexts for a few years, for the benefit of users and system designers. First restricted to an untimed setting, verification was recently extended to systems where quantitative constraints are needed, possibly related to time elapsing. For instance, timed features like TON (Timers ON delay), used in PLC programs, were modeled with timed automata, thus increasing the size of the verification problems addressed.**

In this framework, we propose the modeling and verification of a particular timed multitask PLC program, which is part of the so-called MSS (Mecatronic Standard System) platform from Bosch Group. In this case study, time aspects are combined with multitask programming, which raises questions related to the reaction time between the detection of a signal and the resulting event. Our model for station 2 of the MSS platform is a network of timed automata, including automata for the operative part and for the control program, which is first described in SFC then translated in *Ladder Diagram*.

This model is constrained with atomicity hypotheses concerning program execution, and model checking of a reaction time property is performed with the tool UPPAAL.

***Note to Practitioners*—In this work, we are interested in the combination of timed aspects with multitask PLC programming. It can be used to reduce the reaction time of the control program written in Ladder Diagram language to an external signal. To asses safety properties, we use a model checking tool, UPPAAL. Basically, the model-checking technique consists in: i) modeling the program and its environment as a transition system; ii) modeling some property as a logical formula; and iii) using an algorithm to test if the model satisfies the formula. We propose some techniques and abstractions to permit the verification of timed properties with the tool UPPAAL for a part of the control of a real system.**

***Index Terms*—Model checking, programmable logic controllers, timed automata.**

Manuscript received March 23, 2009; accepted July 22, 2009. Date of publication July 01, 2010; date of current version October 06, 2010. This paper was recommended for publication by Associate Editor B. Turchiano and Editor Y. Narahari upon evaluation of the reviewers' comments. This work was supported in part by the Pluri Formation Project VSMT of Ecole Normale Supérieure de Cachan (ENS Cachan), Cachan, France.

H. Bel mokadem is with Laboratoire Spécification et Vérification, Ecole normale supérieure de Cachan (ENS Cachan), Cachan 94235, France (e-mail: mokadem@lsv.ens-cachan.fr).

B. Bérard is with LIP6, Université Pierre et Marie Curie, Paris 75005, France (e-mail: beatrice.berard@lip6.fr).

V. Gourcuff, O. De Smet, and J.-M. Roussel are with LURPA, Ecole Normale Supérieure de Cachan (ENS Cachan), Cachan 94235, France (e-mail: vincent.gourcuff@lurpa.ens-cachan.fr; olivier.de_smet@lurpa.ens-cachan.fr; jean-marc.roussel@lurpa.ens-cachan.fr).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2010.2050199

I. INTRODUCTION

General Context:

VERIFICATION of safety properties for programs is a very important issue for users and system designers, particularly when those programs are to control critical applications for reactive systems. Indeed, many accidents have been discovered to be the result of programming errors, which lead to the need of formal methods for system verification. Among these methods, model-checking is a rather successful one. Basically, the model-checking technique consists in: (i) modeling the program and its environment as a transition system; (ii) modeling some property as a logical formula; and (iii) using an algorithm to test if the model satisfies the formula. This algorithm is based on a symbolic exploration of the system state space, thus providing an exhaustive search for counterexamples for the property to be tested. It is implemented in a tool called *model-checker*, many of them having been developed for untimed systems.

Since the 1990s, the attention focused on systems where quantitative properties related to time are involved. For such systems, additional components must be introduced, for instance clocks, thus increasing the size of the model and making the verification step harder. Nevertheless, the model of timed automata, introduced in 1990 by Alur and Dill [2], [3], has proved very fruitful: some positive decidability results were obtained for this model, as well as for some extensions, and analysis algorithms were implemented in efficient tools called *timed model-checkers*, like HYTECH [10], KRONOS [6] or UPPAAL [12], which were then applied to industrial case studies.

Comparison with other techniques [9], [22] such as discret event simultation (POOSL, *http://www.es.ele.tue.nl/poosl/*, SHESIM), symbolic timing analysis (SymTA/S, *http://www.symtavision.com*), modular performance analysis (MPA, *http://www.mpa.ethz.ch*) showed similar cost in time and resources for less accurate results. This comforts us in the choice of the timed verification approach.

The particular framework of PLC programs also attracted increasing interest in the past few years. In this area, work was mostly devoted to the untimed setting [4], [8], [18], even when function blocks for timers were included [19], although the model of timed automata has already been used for the modeling of timed features in PLC programming [7], [15], [16].

Contribution: In this work, we are interested in the combination of these time aspects with multitask PLC programming. Our case study concerns a part (called station 2) of the Mecatronic Standard System (MSS) platform from Bosch Group, in

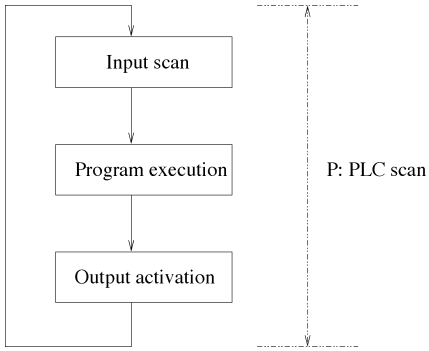


Fig. 1. The cyclic execution of a PLC program.

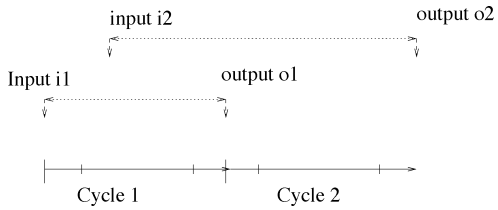


Fig. 2. Reaction time with mono-task programming.

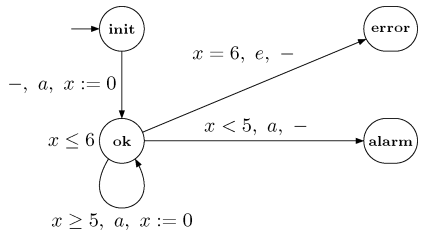


Fig. 3. A timed automaton.

which multitask programming can be used to reduce the reaction time of the control program to an external signal. The program is written in *Ladder Diagram*, one of the languages most commonly used in this area, which is part of the IEC-61131-3 standard [11]. We give semantics for a subclass of Ladder Diagram programs including timer function blocks, in terms of timed automata, and we also provide a timed automata based model for the operative part of the system. These timed automata are described in UPPAAL syntax. While a similar approach was introduced in [15], we propose here additional restrictions which significantly reduce the size of the complete model, obtained from its components by a synchronized product. These restrictions consist of atomicity hypotheses, compacting sequences of actions from the control program into a single one, and lead to reasonable verification times for the response property to be checked. We also give a simpler model for timers, using particular features of UPPAAL.

Outline: Section II of the paper explains the context of this study: the problem of reaction times in PLC programs, and includes a description of timed automata and a short presentation of UPPAAL. In Section III, we give more details on Bosch MSS platform and in Section IV, we give the semantics of the control program. Section V presents the UPPAAL timed automata which form the components of the network, while Section VI gives the results of the verification step.

II. PROGRAMMABLE LOGIC CONTROLLERS AND TIMED AUTOMATA

In this section, we describe the general context of our study and recall the main features of timed automata.

A. Programmable Logic Controllers With Multitask Programming

Programmable Logic Controllers (PLCs) execute programs for the control of an operative part, to which they are connected via an input/output system. The control programs can be written in several languages described in the IEC-61131-3 standard. The execution of such a program consists in iterating a cycle with three main steps (Fig. 1): first, input variables are read and their values are stored in memory. Then, a computation step is performed using these values, producing output values which are also stored. The last step is an activation using the output values. The cycle duration P is called the PLC scan.

The programming design may be either mono-task or multitask. In the first case, a single program executes sequentially, while in the second case, the main task can be interrupted by additional parts of code, either with a fixed period or triggered by some events. These two execution models result in different reaction times to changes of values. In the mono-task case, if the change of value occurs at the input scan, the corresponding output is emitted at the end of the PLC cycle. If the change occurs later, this output may be emitted at as far as the end of the next cycle. This results in a reaction time in the interval $[P, 2P]$ (Fig. 2). This reaction time can be reduced with multitask programming, which is available in most PLCs (with the use of so-called Organization Blocks [20]): consider an event-driven task interrupting the main task when some event occurs. In turn, the interrupting task reads its input and computes its new output values. Depending on the configuration and type of the PLC, these values can be emitted either at the end of the event-driven task or at the end of the current main task. In this work, we investigate the second case where output values of the event-driven task are emitted by the main program, which yields a reaction time of at most P.

B. Timed Automata

The model of timed automata was introduced by Alur and Dill [2], [3]. A timed automaton is built from two main parts:

- a finite automaton in the usual sense, which describes the states also called locations and the discrete transitions of the system;
- a finite set of real variables called clocks, used for the specification of quantitative time constraints which may be associated with transitions. These variables evolve synchronously with time.

Example: The timed automaton on Fig. 3 describes a system which observes a sequence of events a. In location **ok**, the delay between two consecutive occurrences of a must belong to $[5, 6[$. If it is less than 5, an alarm is triggered (the automaton enters location **alarm**), and if the time reaches 6 before the next a occurs, the system enters the **error** location. To measure these delays, a clock x is reset at each occurrence of a inside the interval $[5, 6[$.

It then increases as time elapses and its value can be compared to the constants 5 and 6, to see if a change of mode is required.

Formally, a timed automaton is a tuple $A = (\Sigma, X, Q, q_0, I, E)$, where:

- Σ is a finite set of actions;
- X is the finite set of clocks;
- Q is a finite set of locations, with $q_0 \in Q$ the initial location;
- I is a mapping associating with each location q an *invariant* $I(q)$;
- E is the set of transitions.

In order to describe more precisely the components I and E, we denote by $P(X)$ the powerset of the set X (of clocks) and by $C(X)$ the set of boolean conjunctions of atomic formulas of the form $x \bowtie c$ for a clock x, a constant c in $\mathbb{N}$ (the set of natural numbers) and $\bowtie$ in $\{<, \leq, =, \geq, >\}$. Elements of $C(X)$ are called *clock constraints*.

- An invariant $I(q)$ for location q is a clock constraint, which contains only atomic formulas of the form $x \leq c$ or $x < c$. This constraint must hold as long as time elapses in this location. In the example of Fig. 3, the condition $x \leq 6$ is an invariant for location **ok**.
- The set E of transitions is a subset of $Q \times C(X) \times \Sigma \times P(X) \times Q$.
  A *transition* (q, g, a, r, q') of the automaton is written $q \overset{g,a,r}{\rightarrow} q' \in E$. Its label contains three parts (each one is optional).
  1) The first one g is a constraint in $C(X)$, called the *guard*, which must be satisfied for the transition to be fired. For instance, $x < 5$ is the guard for the transition from location **ok** to location **alarm**.
  2) The second one, called *action name*, is an element of Σ, like a in the example.
  3) The third one, $r \in P(X)$, called *clock reset*, contains the subset of X of clocks which must be reset to zero when the transition is fired. For $r = \{x\}$, a reset of x when reaching location **ok** is written $x := 0$ in Fig. 3.

In order to define the semantics of a timed automaton, we use the notion of *clock valuation*. Let $\mathbb{R}_{\geq 0}$ denote the set of non-negative real numbers. A valuation is a mapping from X to $\mathbb{R}_{\geq 0}$, and the set of valuations is written $\mathbb{R}_{\geq 0}^X$. For each $v \in \mathbb{R}_{\geq 0}^X$ and $d \in \mathbb{R}_{\geq 0}$, we use $v + d$ to denote the valuation which maps each clock $x \in X$ to the value $v(x) + d$.

For a subset r of X, we write $v[r \leftarrow 0]$ for the valuation which maps each clock in r to the value 0 and agrees with v over $X \setminus r$. Constraints of $C(X)$ are interpreted over clock valuations.

The semantics of a timed automaton is given in terms of transition systems. A *configuration* of the system is a pair (q, v), where q is a location of the automaton and v is a valuation of the variables, i.e., a mapping associating a real value with each clock. The initial configuration is (q_0, v_0) where all clock values are equal to 0 in v_0.

The system may change its configuration in two ways.

- Either by letting time elapse: as long as no invariant is violated in the current location, time may progress and the clock values increase by the amount of time elapsed. Such a move is written $(q, v) \overset{d}{\rightarrow} (q, v + d)$ for a delay of d time units, and it is possible only if $v + d$ satisfies the invariant $I(q)$ of location q.
- Or by carrying out a discrete transition of the automaton: the clocks to be reset take the value zero, while the values of the other clocks remain unchanged. This is written $(q, v) \overset{a}{\rightarrow} (q', v')$, when there is a transition $q \overset{g,a,r}{\rightarrow} q'$ in E such that v satisfies the constraint g, $v' = v[r \leftarrow 0]$ and v' satisfies the invariant of q'.

In the example above, the initial configuration of the system is $(\mathbf{init}, 0)$. A particular execution of the system can be described by the following sequence of configurations: $(\mathbf{init}, 0) \overset{7.2}{\rightarrow} (\mathbf{init}, 7.2) \overset{a}{\rightarrow} (\mathbf{ok}, 0) \overset{5.3}{\rightarrow} (\mathbf{ok}, 5.3) \overset{a}{\rightarrow} (\mathbf{ok}, 0) \overset{6}{\rightarrow} (\mathbf{ok}, 6) \overset{e}{\rightarrow} (\mathbf{error}, 6)$.

C. The Tool UPPAAL

The tool UPPAAL (see [5] for the most recent developments) offers a compact description language, a simulation module and a model-checker. In this paragraph we present a subset of UPPAAL functionalities which are sufficient for our purpose. A system is represented in UPPAAL by a collection of timed automata, which communicate through binary synchronization: a channel c can be defined for two automata. Sending a message is denoted by the discrete action $c!$ while receiving the message is denoted by $c?$. These automata also handle a set of (discrete) integer variables, a feature which makes easier the modeling of complex systems. Then, a guard is a conjunction of atomic clock conditions and similar conditions on the integer variables. Moreover, a clock reset may be augmented by an update of the integer variables, of the form $z := c$ for some constant c.

A global *configuration* is a triple (ℓ, v, w), where ℓ is a location vector (indicating the current state in each component of the timed automata network), v is a valuation of the clocks and w is a valuation for discrete variables: as before, they assign to each clock a real value, but they also assign to each discrete variable an integer value. An execution in the network starts in initial locations of the different components with all the clocks and variables set to zero. The semantics of this model is expressed by moves between configurations. Three types of moves can occur in the system: delay moves, internal moves, and synchronized moves. Delay moves and internal moves have already been described above for a single automaton, so we simply describe now the global evolution.

1) Delaying: Given a current location vector, time elapses for all automata synchronously, as long as no invariant is violated. All clock values increase by the amount of time elapsed. No changes occur for the locations or the integer variables. The move for a set of n automata can be described as above by $(\ell, v, w) \overset{d}{\rightarrow} (\ell, v + d, w)$, where $\ell = (q_1, \ldots, q_n)$ is the tuple of locations, v is the clock valuation and w is the valuation of discrete variables. It is possible only if $v + d$ satisfies the conjunction of invariants $I(q_i)$, for all $1 \leq i \leq n$.

2) Performing an Internal Action: An internal action is an action which corresponds to neither $c!$ (sending a message), nor $c?$ (receiving a message). If such an action is enabled (the variable values satisfy the guard condition), the component can perform this action alone, while the others do nothing. Only the location of this component is changed, as well as its variables, according to the transition. If $q_i \overset{g_i,a_i,r_i}{\rightarrow} q_i'$ is the transition possible from component i, the global move can be written $((q_1, \ldots, q_i, \ldots, q_n), v, w) \overset{a_i}{\rightarrow} ((q_1, \ldots, q_i', \ldots, q_n), v', w')$

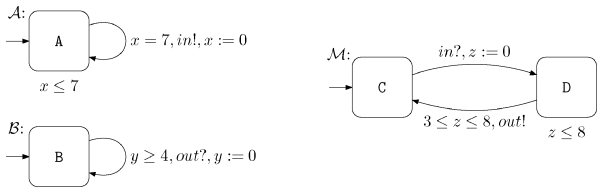


Fig. 4. Three components with channels in and out.

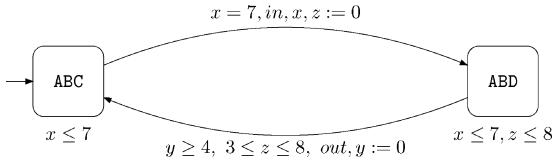


Fig. 5. Composition $\mathcal{A}|\mathcal{B}|\mathcal{M}$.

with $v' = v[r_i \leftarrow 0]$ and w' is obtained by the update of discrete variables from component i.

3) Synchronizing: If, in the network, some complementary actions $c!$ and $c?$ are enabled in two components (in particular, guards must be satisfied by the current valuation), then these components can synchronize. Note that if a component can execute only an action $c!$ while no other component can perform a complementary action $c?$, then this component is blocked. A carefree programing can lead to a deadlock, for instance if the system consists of two components, the first one executing $c!$ followed by $d?$ when the second executes $d!$ followed by $c?$. In the synchronizing transition, the location vector is changed for both components and the clock and variable values are changed according to the clock reset and updates of variables for the two transitions. If components i and j synchronize on some channel c, with transitions $q_i \overset{g_i, c!, r_i}{\rightarrow} q_i'$ and $q_j \overset{g_j, c?, r_j}{\rightarrow} q_j'$, the move is described by $((q_1, \ldots, q_i, \ldots, q_j, \ldots, q_n), v, w) \overset{c}{\rightarrow} ((q_1, \ldots, q_i', \ldots, q_j', \ldots, q_n), v', w')$, with $v' = v[(r_i \cup r_j) \leftarrow 0]$ and w' is obtained by the update of discrete variables from components i and j. Multiple assignment of the same variable can occur, but it generally indicates bad programming in the control program.

To illustrate this composition operation, consider the example from [21], where three components $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{M}$ communicate with two channels *in* and *out* (Fig. 4). The first one produces an *in!* message every 7 time units, the second one, $\mathcal{B}$, emits *out?* messages, with at least 4 time units between them, and the third one, $\mathcal{M}$, performs the communication by transmitting to $\mathcal{B}$ *out!* the messages received from $\mathcal{A}$ *in?*, with a delay between 3 and 8 time units.

Fig. 5 shows how to obtain the composition $\mathcal{A}|\mathcal{B}|\mathcal{M}$, by synchronizing on the channels in and out.

Finally, we introduce two additional features of UPPAAL which will be very useful in our modeling.

- A *committed* location (decorated by the special label C) in an automaton, corresponds to a location in which no progress of time is possible and no transition from a non-committed location is allowed.
- A *broadcast* channel is a channel where more than two automata may communicate: emission of a message $c!$ can be synchronized with several (possibly zero) receptions $c?$ in other components. Note that this is a nonblocking synchronization, since the sender is never blocked, although the receiver must synchronize if it can. Guards on clocks are not allowed on the receiving edge.

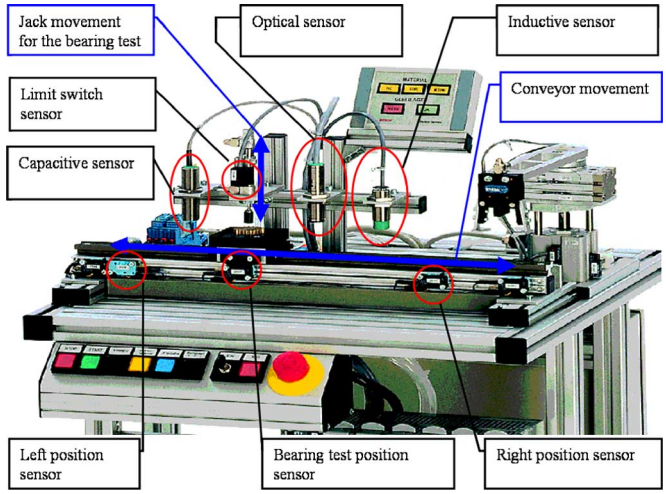


Fig. 6. Presentation of station 2 of the MSS platform.

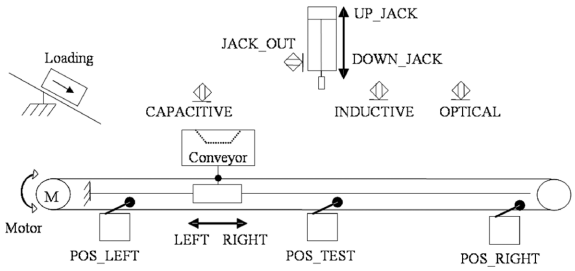


Fig. 7. Scheme of station 2.

III. Description of the Mecatronic Standard System (MSS) Platform

A. Presentation

Platform MSS (from Bosch Group) provides a function for sorting a stock of pinions of different materials and for adding or withdrawing a press-fit bushing to a given pinion [17]. The platform contains four stations.

Our study is centered on station 2. Fig. 6 shows the various components of this station, which are schematized in Fig. 7. The work-pieces are transported by a linear conveyor put into motion by a belt, and driven by an engine with two direction moves. They first reach a scanning position, where the presence or absence of a press-fit bushing is detected. They are then tested by three sensors (respectively inductive, capacitive and optical sensor) to determine their material (steel, copper or black PVC). The detected information is forwarded to the next stations. A rotary/lift gripper performs the transfer to a follow-on station if applicable.

The function performed by the controller for the linear conveyor and the detection of the press-fit and material is presented in Fig. 8 by the SFC [11] specification, where no interruptive task is used. In step 1 the motor starts to move the conveyor to the

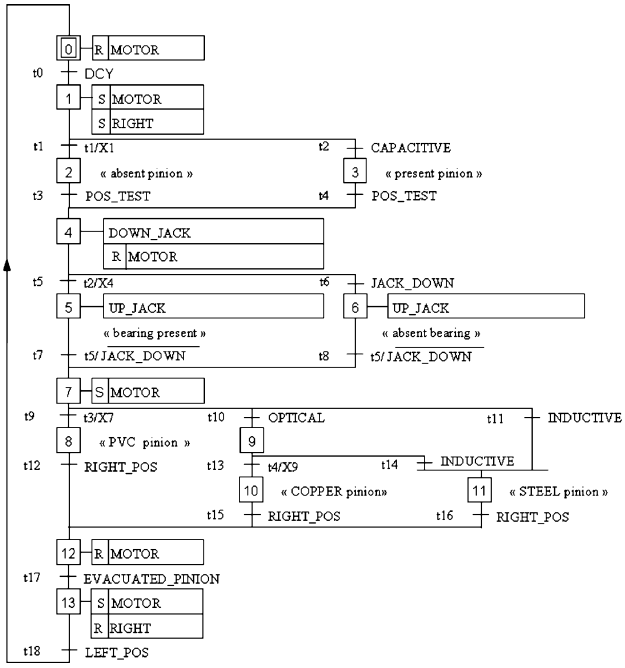


Fig. 8. SFC specification of station 2.

right. Steps 2 and 3 are used to detect the presence of a pinion and arrival of the conveyor at the test position. Step 4 stops the motor to halt the conveyor, steps 5 and 6 correspond to the test by the jack. At the end of the test, Step 7 restarts the motor to move the conveyor further to the right. Steps 8 to 11 test the material of the pinion as it passes under the various sensors. Step 12 is activated when the conveyor is at the rightmost position (motor stops), the end of the transfer by the rotary/lift gripper is indicated by EVACUATED_PINION. Step 13 returns the conveyor to its initial position. It can be noted that the controller specification uses six timed functions as TON blocks (for instance `t1/X1`), in order to measure delays. The semantics of timers is explained in Section IV-C.

With this specification, a problem arises when the conveyor arrives at the bearing test position (POS_TEST sensor). At this time the conveyor moves at high speed (200 mm/s) and the variation of the reaction time of the control system, above 10 ms, is not negligible. Indeed the conveyor position should have a precision of 1 mm for the tester (or jack) to be able to penetrate inside the pinion, in case the bearing is absent. So, we can deduce that the variation of the reaction time of the control system must be less than 5 ms. In the rest of the paper, we study the case of a multitask controller, with an event-driven task, launched on the rising edge of the test position (POS_TEST) sensor, which stops the conveyor if it comes from the loading station.

B. Properties to Check

The multitask control program of this station must satisfy the following properties.

P1: to ensure safety, the conveyor must stop on its way out but not when it comes back from unloading.

P2: the time performance is accurate: the conveyor stops in less than 5 ms at the press-fit bushing test point.

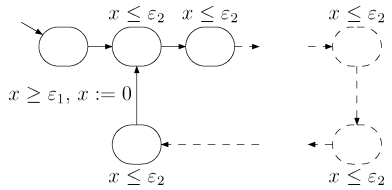


Fig. 9. Mader–Wupper model.

In this work, we focus on the timed property **P2**, and show that the multitask solution reduces the reaction time (Fig. 2). For the first property, the model of the system will be built so that an erroneous control program can invalidate it. In our case, the proposed control program ensures **P1**.

IV. MODELING STATION 2

In this section, we briefly recall the timed automata based semantics proposed by [15] for a control program. Then we explain the structure of our model for (station 2 of) the MSS platform, with a particular attention to the question of timers. Finally, we give the complete description of UPPAAL timed automata for the system.

A. Modeling Principles for the Control Program

Various models have already been proposed for the analysis of PLC programs. Our approach is based on the model introduced by [15], which disregards the exact execution times of elementary instructions.

As depicted in Fig. 9, the model has a clock x to measure the cycle scan, which is thus reset after each cycle of the program. The duration of the two steps `input scan` and `output activation` (see Fig. 1) is at least ε_1 while the whole PLC cycle is at most ε_2. Therefore, the invariant $x \le \varepsilon_2$ is associated with each location. The guard $x \ge \varepsilon_1$ appears on the last edge of the cycle which models the `output activation` and `input scan` steps. A dotted edge in the model describes a step of the control program.

Mader-Wupper also models each timer block as a timed automaton that runs in parallel with the control program. Synchronization is performed through operations on the timer variables and on the timer calls, which requires one extra clock and three synchronization channels for each timer.

B. An Overview of the Model

Our model is built in a compositional way from a collection of nondeterministic processes with finite control structure and real-valued clocks, communicating through channels or shared variables (see Fig. 10). The two main parts are the environment and the control program, which communicate through shared variables and synchronization messages. The modeling of the operative part (environment) is necessary for the verification of the safety and performance properties stated previously.

The operative part initializes the input variables of the PLC, used to compute the output variables. According to these new values, messages are sent to the operative part, acting as orders given to start or stop the conveyor, or to get the jack down. When the conveyor is at the testing position, the operative part sends a

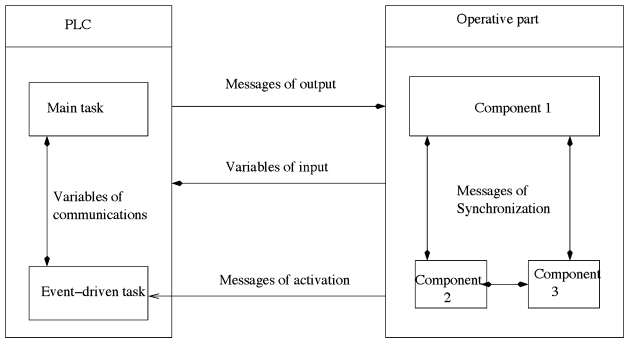


Fig. 10. An overview of the model.

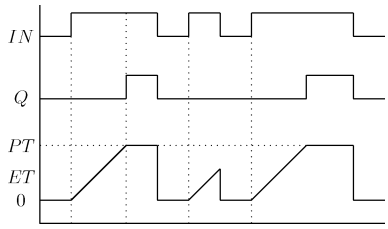


Fig. 11. Time diagram of a TON block.

message to activate the event-driven task. Details are explained in Section V.

C. Modeling Timers

The control program contains timing operations, described by functional blocks called timer on-delay (TON). This mechanism has:

- two input variables: a boolean variable IN, to start or stop counting the time and a time parameter PT (Preset Time) which indicates the timing delay,
- two output variables: a Boolean variable Q, with value 1 if the delay has expired and a time variable ET (Elapsed Time) which gives the time elapsed from the last rising edge of IN.

The IEC 61131-3 standard explains the behavior of a TON block by the diagram in Fig. 11. The time count starts when the variable IN changes from 0 to 1. Then, variable ET increases until reaching the value PT. At that point, variable Q is set to 1. When IN changes to 0, then ET and Q are immediately set to 0. If IN changes from 1 to 0 before ET reaches the delay PT, then ET is immediately set to 0.

Note that only the boolean variables of the timers IN and Q appear in the control program (Fig. 8). For the IEC 61131-3, it is not mandatory to connect all input and output variables of a functional block. Since the variable ET is not used in the program it will not appear explicitly in the model of the timer (see Fig. 12), but is instead modeled by a clock. The parameters PT associated with various instances of TON blocks are defined in a declaration part which is not represented here.

Six independent timers are used in station 2 control program. We now explain how our model of a TON function block differs from that of Mader–Wupper [15] and how we use broadcast channels in UPPAAL to avoid deadlocks. Each TON block

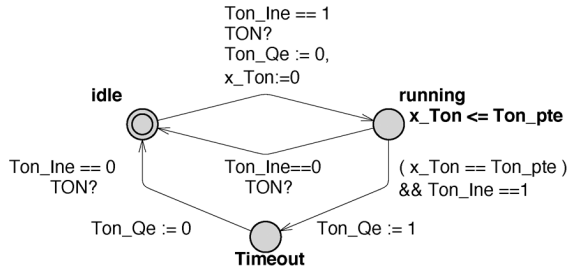


Fig. 12. UPPAAL model of a TON block.

is modeled by an automaton, described in Fig. 12, with three locations, one clock *x-Ton* and two discrete variables *Ton-Ine* (input) and *Ton-Qe* (output).

Initially **idle**, the location becomes **running** when the timer has been switched on and **Timeout**, when some fixed preset delay (constant *Ton_pte*) has been reached. At each cycle of the main task, a synchronization message `TON!` } is sent with a broadcast channel (as seen in Fig. 18). This message is synchronized with `TON?` for all timers. We choose this modeling technique because it allows us to use a single broadcast channel for all TON blocks instead of three ordinary channels per TON in Mader–Wupper's model. Note that in our case, no deadlock can occur. This choice is validated in practice since all TONs are used to measures delay of larger scale than the PLC cycle time. Another assumption in PLC languages is that the state of a TON block cannot change as the control program is processed.

D. Modeling the Environment

In order to validate not only the PLC program but also its integration in the system it has to control, we also need to model the operative part. This implies a thorough knowledge of the system to control, particularly the behavior of each element and its reaction time. Modeling the environment makes it possible to speed up the verification time, in particular by reducing the combinatorial aspects related to nondeterministic definition of all possible input values, including sometimes non relevant ones. Indeed, when the input values of the PLC program are emitted by a model instead of a nondeterministic process, the space of reachable states is reduced. Since our timed properties are significant only in nominal mode, we choose to restrict the model of the plant to the nominal mode without failure.

Each physical device is represented by a timed automaton. In such an automaton, a given location represents a particular configuration of the device. However, while some components of the operative part, like a sensor for example, behave as discrete event systems, this is not the case for all of them. Some devices like closed-loop control have pure continuous behaviors, and cylinders have hybrid behaviors, discrete in the end of their course but continuous during the move. In the models proposed here, clocks are the only continuous components, while physical continuous moves are discretized (for instance, for the conveyor). Modeling with hybrid tools like HYTECH (instead of UPPAAL) would be a solution to this problem, but at the price of much larger verification times, and sometimes even no guarantee of termination.

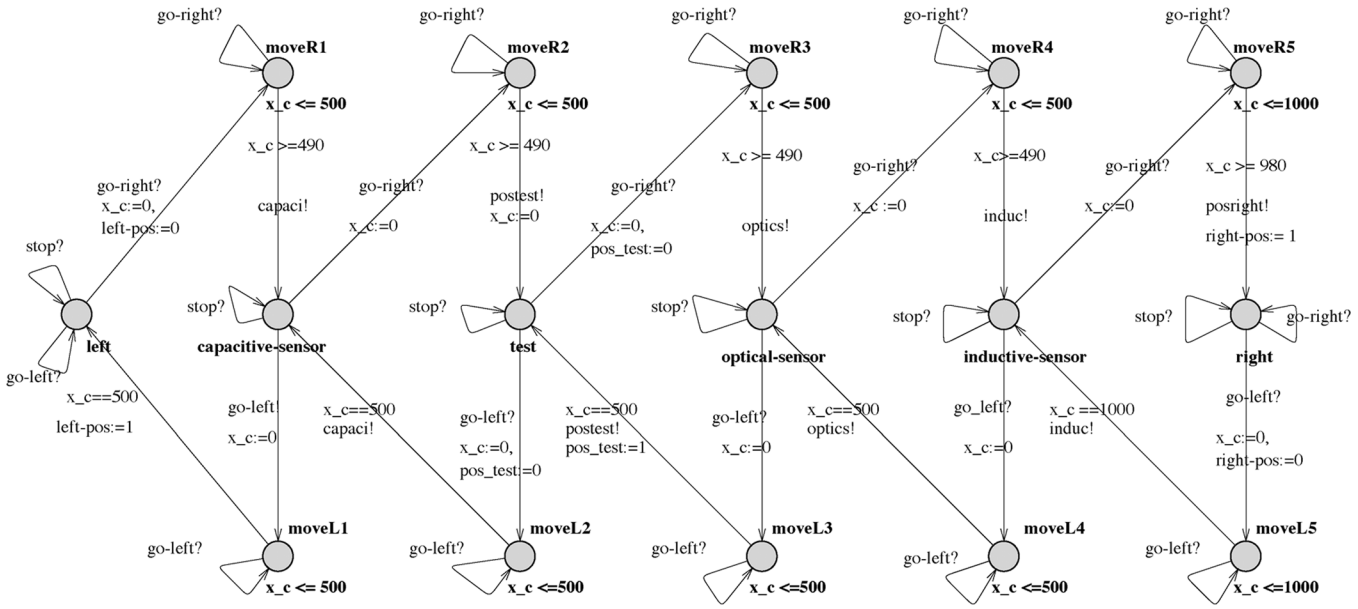


Fig. 13. Timed automaton for the conveyor.

1) The Linear Conveyor: The conveyor is the main element of the operative part, because several triggerings of sensors depend on its position. It is also the most delicate to model because of its continuous behavior along the belt, while our model can only provide a discrete abstraction of this behavior, leaving out the details which do not influence the properties to be checked. In order to obtain reasonable performances in terms of memory and automatic verification time, we model only the almost stable positions, i.e., the positions where the conveyor can stop, or trigger a sensor. These positions correspond to the six states: **inductive-sensor**, **capacitive-sensor**, **optical-sensor**, **test**, **left**, and **right**. They are simply associated with a particular point on the conveyor trajectory, useful for detection, but the conveyor does not stop in these positions, except if it receives a *stop!* message. Between two given positions, the behavior of the conveyor is assumed to be a movement with constant speed, and is thus modeled by a clock. This clock evolves in only one state and is controlled by an invariant, the corresponding constant representing the time needed by the conveyor to cross the distance between these two positions. For example, the conveyor goes from the left position to the capacitive-sensor position in 490 to 500 ms. There is another abstraction imposed by the fact that no stopwatch exists in UPPAAL: between two almost stable positions, the conveyor cannot change direction. However, on a stable position, the direction is permitted to change, which allows property **P1** to be violated. In fact this property is satisfied with our control program as in [18]. The conveyor sends synchronization messages to the various sensors (like $optics!$) and the event-driven task ($postest!$) at the time of its arrival to the test position. It also modifies the input variables of the control program. The corresponding automaton is represented in Fig. 13.

Note that we did not model time non-determinism for the movement of the conveyor from right to left (lower part of the automaton). This is not relevant for the properties to check as this corresponds to Step 13 of the SFC specification Fig. 8.

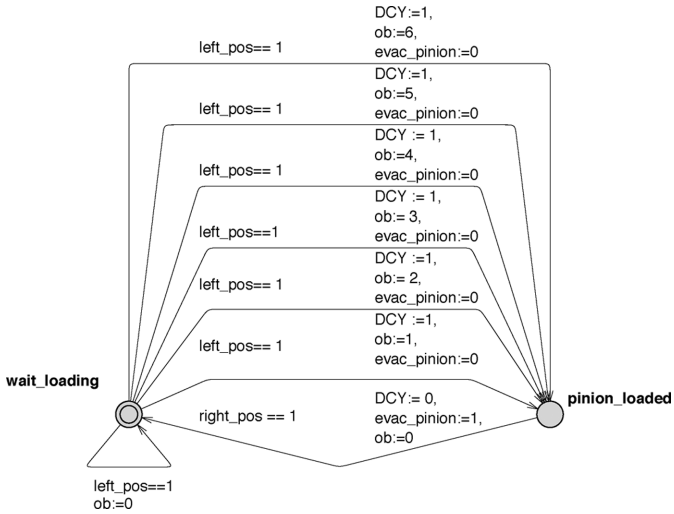


Fig. 14. Model of the environment external to station 2.

2) The External Environment: In station 2, the leftmost position corresponds to the loading of pinions, while the rightmost position is used for unloading. However, the control of loading and unloading operations is not part of this station, which just waits for them to be done. Information about termination of one of these operations is obtained through changes of input values. Upon loading, the conveyor is provided an unspecified pinion. This is modeled by an automaton, presented in Fig. 14, which selects in a nondeterministic way the nature of the pinion (variable ob) when the conveyor is at the leftmost position.

3) The Jack: The jack detects the presence or absence of a press-fit bushing in a work-piece. This test is made by a vertical movement of the jack until a limiting position. The jack must go down until the limiting position is reached, in a given time, to conclude to the absence of the press-fit bushing. The time allotted for this movement is given in the specification (see Fig. 8) by the TON `t2/X4`. As it is larger than the PLC cycle duration,

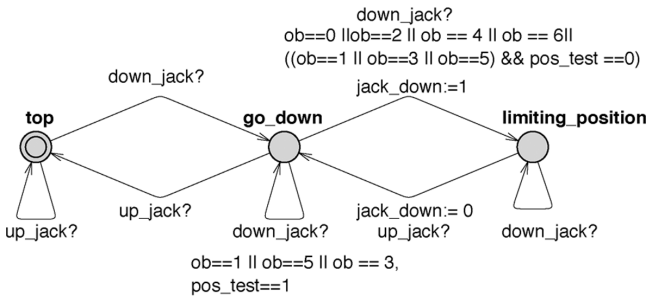


Fig. 15. Timed automaton for the jack.

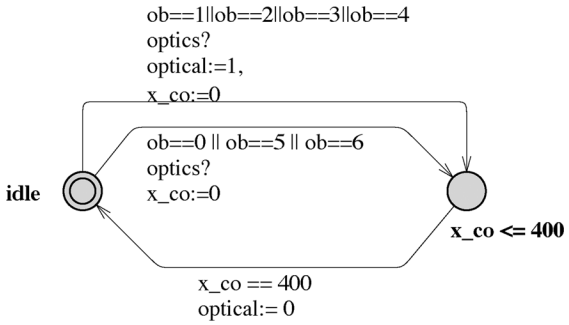


Fig. 16. Timed automaton for the optical sensor.

the *down_jack* order will be held during several PLC cycles. The model of this sensor (Fig. 15) depends on the characteristics of the work-pieces which are represented by the values of the variable *ob*. The automaton starts from location **top**. It moves to state **go-down** when it receives a message *down-jack?* from the PLC program. From this point on, there are two cases: if there is a press-fit bushing in the work-piece (represented in the model by the guard $ob == 1||ob == 3||ob == 5$) then the automaton waits in the state **go-down**, else it moves to state **limiting position**. This model of the jack needs at least two PLC cycles to reach a stable position, which is ensured by the `t2` duration.

4) The Sensors: The optical, capacitive and inductive sensors are modeled by timed automata synchronized with the automaton of the conveyor. We only show here the model for the optical sensor, the others are very similar, only location names and signal names should be adapted. The conveyor sends the activation messages (for example, *optics?* in Fig. 16) when it is under the corresponding sensor. According to the nature of the material, the sensor modifies the value of the corresponding variable (here *optical*) which is then used by the PLC program. No clock is used in this model as its value is used only during the input scan of the PLC cycle. The state of the sensor is used as a constant within a given PLC scan. The filtering delay for the sensors is assumed to be smaller than the input scan duration (Fig. 1).

E. Modeling the Control Program

1) The Main Program: The functional specification of the global system is modified from Fig. 8 to use an interruptive task in Ladder language (see Fig. 17).

In this specification, the interruptive task is triggered by the `POS_TEST` sensor. To stop only from left to right, the action is

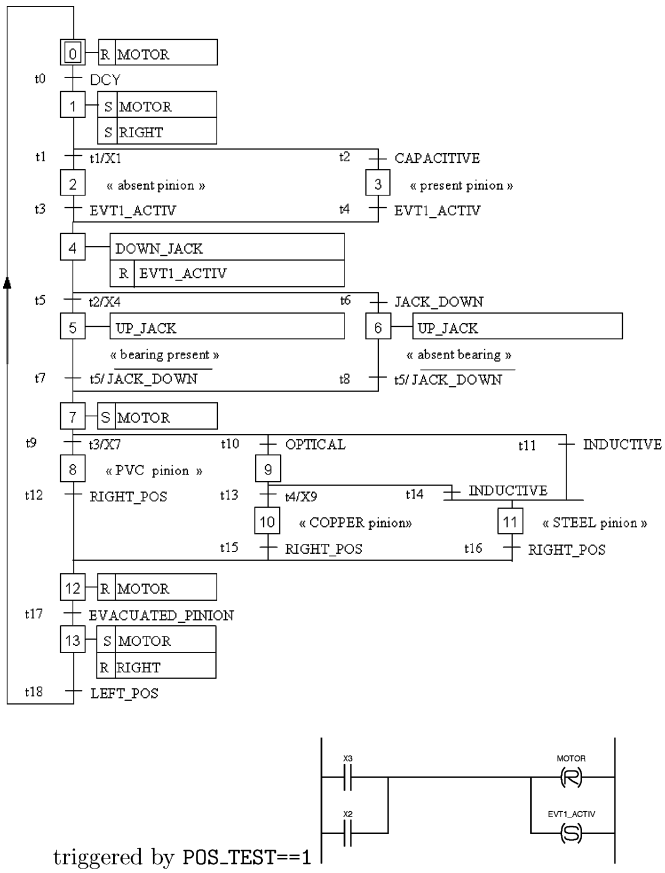


Fig. 17. SFC specification of station 2 with LD interruptive task.

conditioned by being in SFC step 2 or 3 (denoted by `x2` and `x3` in the interruptive task in Fig. 17.

This SFC specification needs to be translated into a PLC programming language. Recall that the execution of a PLC program is a cycle with three phases: input reading, program execution and output writing as in Fig. 1. This translation is done using the (classical) algebraic representation of SFC [13] with three sequential modules inside the program execution part.

- The first module is the computation of the clearing conditions of the transitions as boolean variables called `CFT0...CFT18` in Fig. 18. For instance, the `t0` transition in Fig. 17 is associated with `CFT0` variable defined by `CFT0 = x0&&DCY` which means that state 0 must be active and the transition condition `DCY` must be true to allow activation of Step 1 and deactivation of Step 0 in the next module.
- The second module is the computation of the step variables called `x0...x13` in the same figure. For Step 1, activation is computed by $\texttt{x1} = \texttt{CFT0}||(\texttt{x1\&\&!CFT1\&\&!CFT2})$.
- The last module is the computation of actions using the step variables. As seen in Fig. 17, action `motor` is reset when Step 0 or Step 12 is activated and set when Step 1 or Step 7 or Step 13 is activated, this is written as $\texttt{R\_motor} = \texttt{x0}||\texttt{x12},\texttt{S\_motor} = \texttt{x1}||\texttt{x7}||\texttt{x13},\texttt{motor} = \texttt{S\_motor\&\&!R\_motor}$.

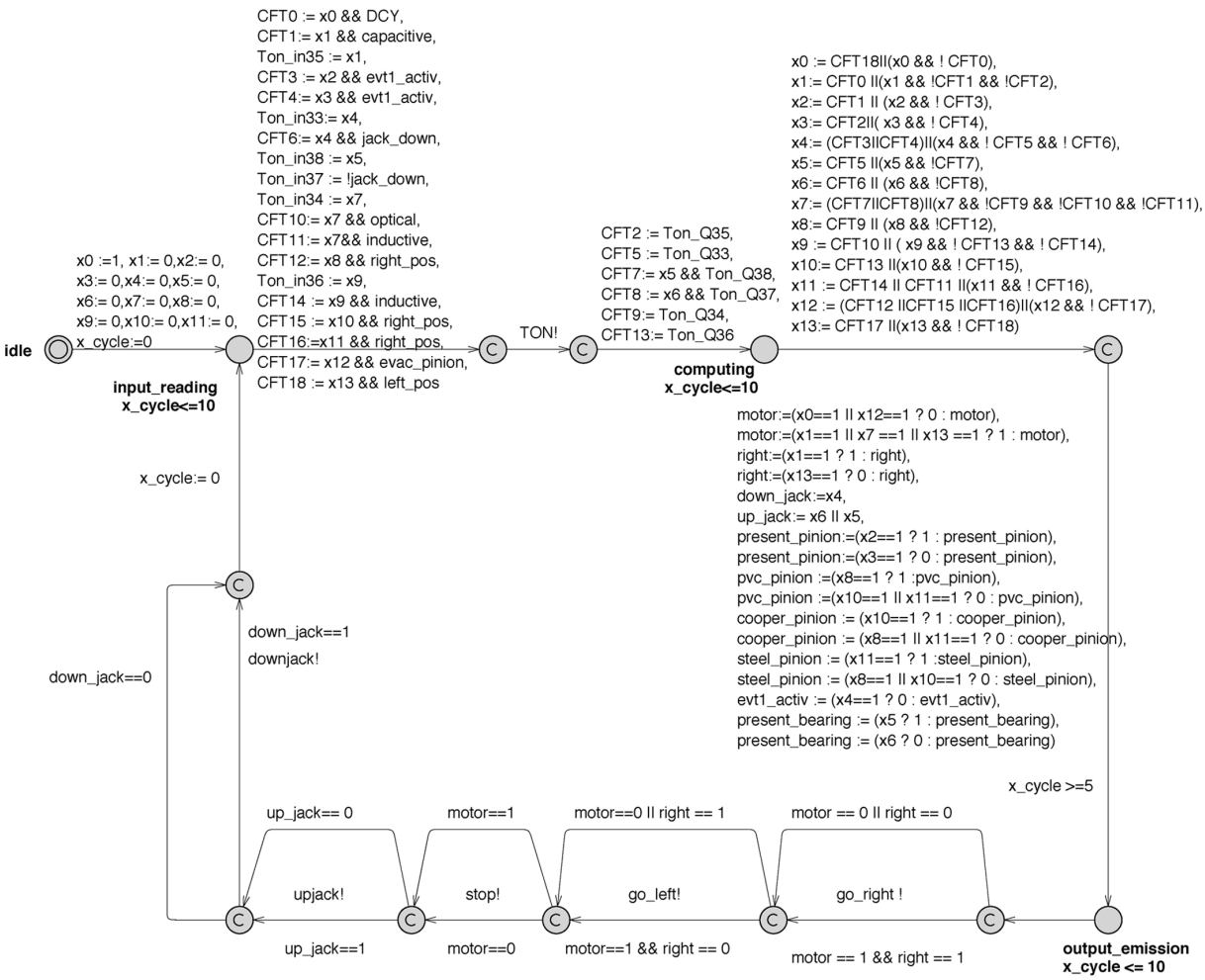


Fig. 18. UPPAAL model of main program.

Those algebraic equations are directly implemented in the PLC with Ladder diagram language, they are also used in the UPPAAL model with small syntactic variations as in Fig. 18.

The complete PLC cycle is modeled in UPPAAL by an automaton structured as a loop, which includes a clock to measure the cycle time (equal to ten time units here). This loop consist of four steps:

1) input reading and computation of new values for the evolution conditions (`CFT`) of the SFC;
2) computation of other new values for SFC variables: step activation (`x`) and output computation (`motor...`);
3) output writing, performed by a sequence of messages for synchronization with the operative part;
4) reset of the clock modeling the cycle time.

The atomicity hypothesis is the following: time can elapse only in the three states between these steps (**input_reading**, **computing**, **output_emission** in Fig. 18), to represent the duration of their execution. Note that the outputs are usually emitted simultaneously by the PLC, so that the duration of the output emission is negligible with respect to the cycle time. We ensure this atomicity property of step 3 with committed locations.

2) The Event-Driven Program: Since it is run upon activation of the bushing-test position, the event-driven task is strongly dependent on the environment. This aspect is modeled by the emission of a message from the conveyor, received by the automaton of the event-driven task (see Fig. 19).

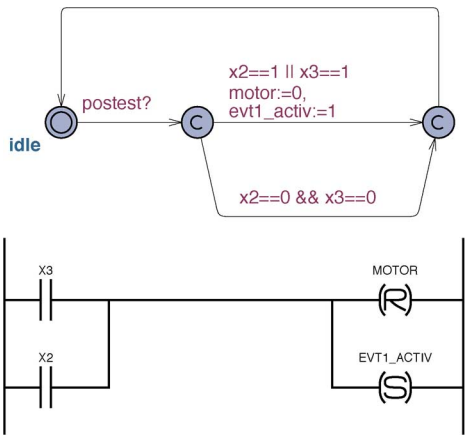


Fig. 19. UPPAAL model for the event-driven task.

When the message *postest?* is received, the automaton executes the algebraic equations which represent the Ladder program and modifies the internal variable `motor` if the condition holds. Note that the execution time of the event-driven task is taken as null due to its negligible duration. In practice, for a

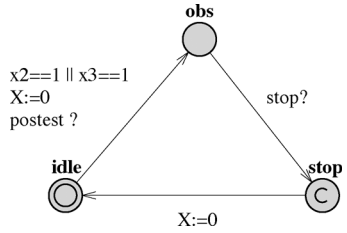


Fig. 20. Observer automaton.

basic PLC, this execution time is about 1 microsecond, while the cycle time of the main program is about 3 milliseconds.

The first *committed* location is used to model the priority of the event-driven task and the other ones to express the null duration. The two programming designs are considered in order to determine the conditions under which the requirements are satisfied and to compare both models:

- the event-driven task only modifies the internal memory of the output as defined above (see Fig. 17);
- the event-driven task is not activated and the main task is designed to control the whole process (see Fig. 8).

V. Verification With Uppaal

A. The Observer Automaton

In order to verify the timed property **P2**, we need to introduce an additional automaton (see Fig. 20) which is composed with the rest of the system. This automaton plays the role of an external observer and does not interfere with the model previously described.

This automaton contains a location **stop**, reached when the conveyor stops in testing position. It also contains a clock X to measure the reaction time. The observer automaton starts from state **idle** with X set to 0. When the message *postest?* is received from the conveyor, the automaton moves to state **obs** and resets the clock X. From this point on, the clock value again increases with time. When the message *stop?* is received from the main program, the automaton switches to state **stop** which must be committed to avoid time elapsing. Thus, the value of X in this last state corresponds to the time elapsed between the triggering of the event-driven task and the physical stop of the conveyor. To check the timed property **P2**, we express its negation (**C1** in Table I): the observer automaton will eventually reach the state **stop** with the value of the clock X greater than five time units. Property **C1** is written as

$$\mathsf{E}\langle\rangle(observer.stop \text{ and } X > 5)$$

in Uppaal syntax, which is a fragment of the logic TCTL [1]. In this formula, the combination $\mathsf{E}\langle\rangle$ means "for some path in the future" and *observer.stop* denotes location **stop** of the observer automaton. The formula above can thus be read as "for some paths in the future, the location **stop** of the observer is reached with the value of clock X greater than 5." Note that if property **C1** is true in our abstracted model, it is also true in the real program. Indeed, if time can elapse in the intermediate locations of the main program, this will result in a possibly greater value for the clock X. On the contrary, if we obtained a negative answer, we would have to extend the model so that time could elapse in the location just before emission of the message *stop!*.

TABLE I
Results of the Experiments

property	result	time	memory
with the event driven task			
C1:E<> observer.stop and $X > 5$	yes	15 s	30 Mb
C2:E<> observer.stop and $X \leq 5$	yes	15 s	30 Mb
C3:E<> observer.stop and $X > 10$	no	22 s	61 Mb
without the event driven task			
C5:E<> observer.stop and $X \geq 10$	yes	16 s	30 Mb
C6:E<> observer.stop and $X > 20$	no	22 s	70 Mb
C7:E<> observer.stop and $X < 10$	no	22 s	69 Mb
with Mader-Wupper model			
C8:E<> observer.stop and $X > 5$	-	-	-

B. Experiments

First note that the global model has about 30.10^6 configurations, which are explored in an on the fly computation of the set of reachable states.

Table I gives the time and memory used for verification (on a linux machine with a pentium4 at 2.4 GHz with 3 Gb RAM). The results provide a comparison of the reaction times between mono-task and multitask programming. Indeed, on the one hand, properties **C5**, **C6**, and **C7** show that the conveyor stops between 10 and 20 time units after it reaches the test position. This is far from being a surprise because these values correspond respectively to one and two PLC cycle times. On the other hand, property **C3** shows that the conveyor stops in less than one PLC cycle time (which would also be the case if time could elapse in intermediate locations because the length of the cycle is preserved). Thus, multitask programming reduces the reaction time. However, property **C1** proves that it is not sufficient to satisfy the requirement **P2**.

Note that, after 29 hours of computation, we stopped the verification process in the case of Mader–Wupper model.

These performances are due to two main reasons: the atomicity hypothesis for executions between some states of the main program and the enhanced model of the TON block.

- The atomicity hypothesis: we assume that each one of the four steps of the main program (Section IV-E1) executes instantaneously. Recall that time can elapse only in three states (**input_reading**, **computing**, **output_emission**).
- The enhanced model of the TON block: we use one broadcast channel to synchronize all the TON blocks and the main program instead of three ordinary channels for each TON block as in Mader–Wupper model.

VI. Discussion and Conclusion

In this work, we use a generic algebraic translation of SFC specification with TON blocks which leads to formal semantics for a subset of Ladder diagram language, with timed automata. The loop structure of the automaton is generic, and updates on the transitions result directly from the algebraic equations.

We also describe the operative part of station 2 of MSS platform with timed automata. This part is not generic in this paper, but some work has been devoted to this problem [14]. On this network of timed automata represented in Uppaal syntax, we

formally prove by model-checking that multitask programming reduces the reaction time of the conveyor, upon emission of an output order to stop. While this does not really come as a surprise, we obtain reasonable verification times (less than 30 s) on a global model with about 30.10^6 states, by adding an atomicity hypothesis to Mader–Wupper model and modifying the automata for timer blocks. In comparison, model-checking the same formula with the original model had to be stopped after several hours.

This experiment shows that timed model-checking is a useful technique for the verification of PLC programs. Attention must be focused on the following points.

- The semantics of the language must be formally defined. Here, due to the algebraic translation of SFC specifications and the loop structure of the PLC cycle, the construction of a timed automaton and a control program in Ladder diagram language could be generated in a systematic way. This construction process could be done automatically in the future.
- Timed aspects can be integrated with respect to some constraints (logical time for the control program, discrete time for plant model).
- The technique requires hypotheses on the behavior of the operative part, on the real-time kernel of the PLC, to build a compact enough model of the system.

Taking these conditions into account may lead to efficient formal verification of timed properties. We believe that design-based approaches (e.g., synchronous languages) would not fit this objective. While design-based approaches can also be used for formal verification purposes, we think that they are not so convenient to deal with the asynchronous context of PLC programming.

Scaling remains the main challenge for verification techniques due to the combinatorial explosion: combining asynchronous processes will result in a very high number of configurations even with untimed models. Here we choose to present a timed approach combined with multitask processes in one PLC. Adding more processing stations controlled by the same PLC would not be a real problem. However, adding a processing station controlled by another PLC leads to combinatorial explosion due to a fine description of the PLC behavior.

Although this study is not yet at industrial scale, our approach aims at techniques that can be applied in an industrial context, with commonly used languages as starting point. This can provide tools to enhance the quality of control programs with the usual industrial practices.

REFERENCES

[1] R. Alur, C. Courcoubetis, and D. L. Dill, "Model-checking in dense real-time," *Inf. Comput.*, vol. 104, no. 1, pp. 2–34, 1993.

[2] R. Alur and D. L. Dill, "Automata for modeling real-time systems," in *Proc. 17th Int. Coll. Automata, Languages, and Programming (ICALP'90)*, Jul. 1990, vol. 443, Lecture Notes in Computer Science, pp. 322–35.

[3] R. Alur and D. L. Dill, "A theory of timed automata," *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, 1994.

[4] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and Ph. Schnoebelen, "Towards the automatic verification of PLC programs written in instruction list," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC'2000)*, Nashville, TN, Oct. 2000, pp. 2449–2454, 2000.

[5] A. David, G. Behrmann, K. G. Larsen, and W. Yi, A tool architecture for the next generation of UPPAAL Dept. Inf. Technol., Uppsala Univ., Uppsala, Sweden, Tech. Rep. 2003-011, Feb. 2003, p. 20.

[6] C. Daws, A. Olivero, S. Tripakis, and S. Yovine, "The tool KRONOS," in *Proc. Workshop Hybrid Systems III: Verification and Control*, New Brunswick, NJ, Oct. 1995, vol. 1066, Lecture Notes in Computer Science, pp. 208–219.

[7] H. Dierks, "PLC-automata: A new class of implementable real-time automata," *Theor. Comput. Sci.*, vol. 253, no. 1, pp. 61–93, 2000.

[8] G. Frey and L. Litz, "Formal methods in PLC-programming," in *Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC'2000)*, Nashville, TN, Oct. 2000, pp. 2431–2436.

[9] M. Hendriks and M. Verhoef, "Timed automata based analysis of embedded system architectures," in *Proc. Parallel and Distrib. Processing Symp.*, 2006, DOI: 10.1109/IPDPS.2006.1639422.

[10] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "A user guide to HyTech," in *Proc. 1st Int. Workshop Tools and Algorithms for the Construction and Analysis of Systems (TACAS'95)*, Aarhus, DK, May 1995, vol. 1019, Lecture Notes in Computer Science, pp. 41–71.

[11] *IEC Standard 61131-3: Programmable Controllers—Part 3*, Standard 61131-3, IEC (International Electrotechnical Commission), 1993.

[12] K. G. Larsen, P. Pettersson, and W. Yi, "UPPAAL in a nutshell," *J. Softw. Tools for Technol. Transfer*, vol. 1, no. 1–2, pp. 134–152, 1997.

[13] J. Machado, B. Denis, J.-J. Lesage, J.-M. Faure, and J. Ferreira Da Silva, "Logic controllers dependability verification using a plant model," in *Proc. 3rd IFAC Workshop on Discrete-Event System Design (DESDes'06)*, Rydzyna, Poland, Sep. 2006, pp. 37–42.

[14] J. Machado, B. Denis, and J.-J. Lesage, "A generic approach to build plant models for DES verification purposes," in *Proc. 8th Int. Workshop On Discrete Event Systems (WODES'06)*, Ann Arbor, MI, Jul. 2006, pp. 407–412.

[15] A. Mader and H. Wupper, "Timed automaton models for simple programmable logic controllers," in *Proc. 11th Euromicro Conf. Real-Time Syst. (ECRTS'99)*, York, U.K., Jun. 1999, pp. 114–122, IEEE Comp. Soc. Press, 1999.

[16] E. Olderog, "Correct Real-Time Software for Programmable Logic Controllers," in *Correct System Design. Recent Insights and Advances*. New York: Springer, 1999, vol. 1710, Lecture Notes in Computer Science, pp. 342–362.

[17] *Mechatronik Standard System*, [Online]. Available: http://www.boschrexroth.com/country_units/europe/germany/sub_websites/brs_germany/de/didactic/lehrsysteme/mechatronik/mechatronik_standard_system_mss/index.jsp, Rexroth Bosch Group.

[18] O. Rossi, O. de Smet, S. Lampérière-Couffin, J.-J. Lesage, H. Papini, and H. Guennec, "Formal verification: A tool to improve the safety of control systems," in *Proc. 4th Symp. Fault Detection, Supervision and Safety for Technical Processes (IFAC Safeprocess 2000)*, Budapest, Hungary, 2000, pp. 885–890.

[19] O. Rossi and Ph. Schnoebelen, "Formal modeling of timed function blocks for the automatic verification of ladder diagram programs," in *Proc. 4th Int. Conf. Autom. Mixed Processes: Hybrid Dynamic Systems (ADPM'2000)*, Dortmund, Germany, Sep. 2000, pp. 177–182.

[20] "Siemens," Programming With STEP 7 v5.4 Ref: A5E00706944-01. [Online]. Available: http://www.automation.siemens.com

[21] J. Sifakis and S. Yovine, "Compositional specification of timed systems," in *Proc. 13th Annu. Symp. Theor. Comput.Sci. (STACS'96)*, 1996, vol. 1046, Lecture Notes in Computer Science, pp. 347–359.

[22] F. W. Vaandrager and A. L. de Groot, "Analysis of a biphase mark protocol with Uppaal and PVS," in *Formal Aspects of Computing*. New York: Springer, 2006, vol. 18, pp. 433–458, number 4.

[23] L. Waszniowski and Z. Hanzlek, "Formal verification of multitasking applications based on timed automata model," *Real-Time Syst.*, vol. 38, pp. 39–65, 2008, DOI 10.1007/s11241-007-9036.

Houda Bel Mokadem received the Ph.D. degree from École Normale Supérieure de Cachan, Cachan, France.

She is an Associate Professor at ENSA de Tanger (Morocco). Her research area is the verification of temporal properties.

Béatrice Bérard is a Full Professor at the University Pierre et Marie Curie, Paris, France. Her research area is the modeling and verification of concurrent systems with quantitative constraints, in particular, real-time and hybrid systems.

Olivier De Smet is currently an Associate Professor of Manufacturing Engineering at the Conservatoire National des Arts et Métier , Paris, France. He carries out research at the LURPA on the control of discrete-event systems with verification approaches.

Vincent Gourcuff received the Ph.D. degree from École Normale Supérieure de Cachan, Cachan, France.

He is a teacher at the Institute of Technology of Cachan, France. He carries out research on the verification of safety properties on PLC programs.

Jean-Marc Roussel received the Ph.D. degree from the École Normale Supérieure de Cachan, Cachan, France, in 1994.

He is currently an Associate Professor of Automatic Control at the École Normale Supérieure de Cachan and carries out research at the LURPA on the control of discrete-event systems with algebraic approaches.

Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 726246, 15 pages
http://dx.doi.org/10.1155/2014/726246

Research Article

Design of Logic Controllers Thanks to Symbolic Computation of Simultaneously Asserted Boolean Equations

Jean-Marc Roussel and Jean-Jacques Lesage

LURPA, ENS Cachan, 61 avenue du Président Wilson, 94230 Cachan, France

Correspondence should be addressed to Jean-Marc Roussel; jean-marc.roussel@lurpa.ens-cachan.fr

Received 11 December 2013; Revised 6 February 2014; Accepted 7 February 2014

Academic Editor: Hamid R. Karimi

Formal methods can strongly contribute to improve dependability of controllers during design, by providing means to avoid flaws due to designers' omissions or specifications misinterpretations. This paper presents a synthesis method dedicated to logic controllers. Its goal is to obtain the control laws from specifications given in natural language by symbolic computation. The formal framework that underlies this method is the Boolean algebra of n-variable switching functions. In this algebra, thanks to relations and theorems presented in this paper, it is possible to formally express logical controllers specifications, to automatically detect inconsistencies in specifications, and to obtain automatically the set of solutions or to choose an optimal solution according to given optimization criteria. The application of this synthesis method to an example allows illustrating its main advantages.

1. Introduction

Programmable logic controllers (PLCs) are industrial automation components that receive input signals coming from sensors and send output signals to actuators, in accordance with control laws implemented into a user program (Figure 1). The control algorithms that allow the real time calculation of new output values, according to the current state of the PLC and the observation of new values of inputs, are written in standardized languages, such as ladder diagram (LD), structured text (ST) or instruction list (IL) [1]. A PLC cyclically performs three tasks: inputs reading, program execution, and outputs updating. The period of this task may be constant (periodic scan) or may vary (cyclic scan).

Because of their reliability, even in very severe conditions in terms of temperature, vibrations, electromagnetic perturbations, and so forth, PLCs are frequently used for the control of safety-critical systems (energy production, transport, chemical industry, etc.). In this context, improving the reliability of the user program has been one of the main challenges of the past two decades in the field of automation. Among the different techniques that can be used in this aim [2], formal verification and validation and formal synthesis are the most efficient. Verification is the proof that the internal semantics of a model is correct, independently from the modeled system. The searched properties of the models are stability, deadlock existence, and so on. The validation determines if the model agrees with the designer's purpose [3]. Efficient validation/verification techniques of PLC programs [4], most often based on model-checking technique, have been proposed by researchers and are now widely used in industry [5], despite problems of state-space explosion that arise when treating large scale systems.

Contrary to verification techniques that aim at proving, after a PLC program has been more or less correctly designed by an expert, that control laws are safe, automatic synthesis methods aim at systematically generating control laws which guarantee by construction the respect of expected safety properties. The avoidance of human errors during the design of controllers is one of the main reasons for which synthesis is a very important subject of research in the field of discrete event systems (DES) since the end of 80's.

Most part of recent works in this area are still based onto the Supervisory Control Theory (SCT) [6] and are aiming for the synthesis of a *supervisor*, and not directly to the *controller* of an automated system. Furthermore, the use of state models (Finite Automata, Petri Nets, etc.) and their composition

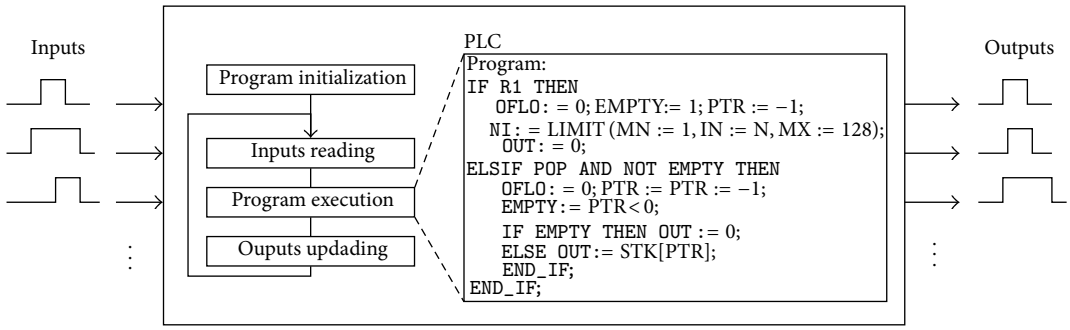


Figure 1: PLC basic principle.

for the construction of the models of the plant and of the specifications generates a complexity which remains problematic for the synthesis of a supervisor for complex systems [7]. It is therefore interesting to explore other ways for performing synthesis, such as algebraic approaches. In previous works, we proposed a method specifically developed to get the control laws that can be directly implemented into the controller [8]. We have chosen to synthesize these control laws under the form of recurrent Boolean equations because of the wide possibilities they offer for the formalization of safety requirements and for implementation.

Nevertheless, whatever is the used synthesis method, one of the weak links of the automatic generation of the control laws is the step of formal transcription by the designer (within state models or algebraic expressions) of the informal requirements and safety properties the controller has to satisfy. In the case of SCT, some authors have proposed more or less generic approaches for the construction of the models of the plant [9] or of the specifications [10]. But in any case, the hypothesis that requirements can be inconsistent has never been taken into account. Unfortunately in the framework of industrial collaborations we have been able to verify that it is always the case. In this paper we show how, in consideration of specific hypotheses, it is possible to install a correction loop for helping the designer to formalize these requirements and so improving the synthesis method robustness to the lack of precision of the specifications.

This paper is organized as follows. Some basics of algebraic synthesis given in Sections 2 and 3 recall the main steps of our method. Section 4 presents the mathematical framework of our approach and new results that allow us to accept inconsistencies in specifications. The strategy we developed for making the synthesis more robust to the lack of consistency of the specifications is described in Section 5, thanks to a case study.

2. Problem Statement

Figure 2 proposes a generic representation of a DES whose controller has p Boolean inputs (u_i), q Boolean outputs (y_j), and r Boolean state variables (x_l). Plant and controller are connected through a closed loop exchanging inputs and outputs signals. The state variables, needed for expressing

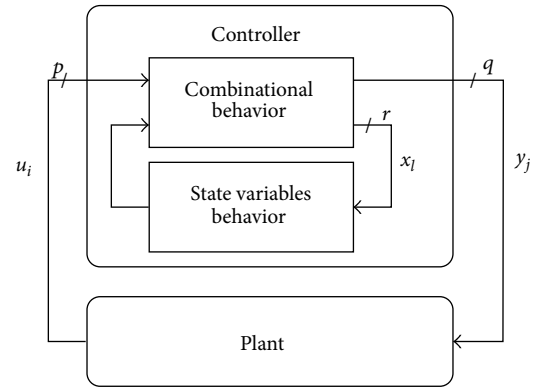


Figure 2: A sequential DES.

sequential behaviors of the controller, are represented by internal variables.

The algebraic modeling of the control laws of the controller necessitates the definition of $(q+r)$ switching functions of $(p+r)$ variables. Even if this representation is very compact (the r Boolean state variables allow the representation of 2^r different states), the construction by hands of these switching functions is a very tedious and error-prone task [11]; the controller of Figure 2 admits 2^p inputs combinations can send 2^q outputs combinations and can express $(2^{2^{(p+r)}})^{(q+r)}$ sequential behaviors. That is the reason why algebraic modeling approaches have been replaced by methods based on state models since the middle of 50's [12, 13]. Nevertheless, thanks to recent mathematical results obtained onto Boolean algebras [14, 15], the automatic algebraic synthesis of switching functions is now possible.

In [16] an interesting approach for the systematic construction of a reactive program from its formal specification is proposed. In this work, the program synthesis is considered as a theorem proving activity. A program with input x and output y, specified by the formula $\varphi(x, y)$, is constructed as a byproduct of proving the theorem $(\forall x)(\exists y)\varphi(x, y)$. The specification $\varphi(x, y)$ characterizes the expected relation

between the input x and the output y computed by the program. This approach is based on the observation that the formula $(\forall x)(\exists y)\varphi(x, y)$ is equivalent to the second-order formula $(\exists f)(\forall x)\varphi(x, f(x))$, stating the existence of a function f, such that $\varphi(x, f(x))$ holds for every x.

This approach provides a conceptual framework for the rigorous derivation of a program from its formal specification. It has also been used to synthesize specifications under the form of finite automata from their linear temporal logic (LTL) description [17].

The core of our approach is based on this strategy: we aim at deducing the $(q+r)$ switching functions of $(p+r)$ variables which define the behavior of the controller from a formula $\varphi(u_i[k], x_l[k-1], y_j[k], x_l[k])$ that holds for every k, every $u_i[k]$, and every $x_l[k-1]$.

To cope with combinatorial explosion, switching functions will be handled through a symbolic representation (and not their truth-tables which contain $2^{(p+r)}$ Boolean values). Each input u_i (resp., output y_j) of the controller will be represented by a switching function U_i (resp., Y_j). To take into account the recursive aspect of state variables, each state variable x_l will be represented by two switching functions: X_l (for time $[k]$) and ${}_pX_l$ (for time $[k-1]$).

According to this representation, the synthesis of control laws of a logical system from its specification can now be transformed into the search of the solutions to the mathematical problem as follows:

$$(\forall U_i)\left(\forall {}_pX_l\right)\left(\exists Y_j\right)\left(\exists X_l\right)\varphi\left(U_i, {}_pX_l, Y_j, X_l\right), \quad (1)$$

where $(U_i, {}_pX_l, Y_j, X_l)$ are $(p+q+2r)$ switching functions of $(p+r)$ variables.

3. Overview of Our Method

The input data of the proposed method (Figure 3) are unformal functional and safety requirements given by the designer. In practice, these requirements are most often given in a textual form and/or by using technical Taylor-made languages (Gantt diagrams, function blocks diagrams, Grafcet, etc.) or imposed standards.

All the steps of our synthesis method are implemented into a prototype software tool developed in Python (Case studies are available online: http://www.lurpa.ens-cachan.fr/-226050.kjsp). The first step is the formalization of requirements within an algebraic description; examples are given in Section 5.2. Requirements expressed with a state model can directly be translated into recurrent Boolean equations, thanks to the algorithm proposed by Machado et al. [18]. In case where the knowhow of the designer enables him to build a priori the global form of the solution (or of a part of the whole solution) it is also possible to give fragments of solution as requirements [19].

The second step consists in checking the consistency of the set of requirements by symbolic calculation. The sufficient condition for checking this consistency has been given in [20] but no strategy has been proposed for coping with potential inconsistencies. In this paper we show that thanks to new theorems the causes of these inconsistencies can be pointed out. It is then possible for the designer to fix priority rules between the concerned requirements that will allow finding, if exist, solutions despite inconsistencies.

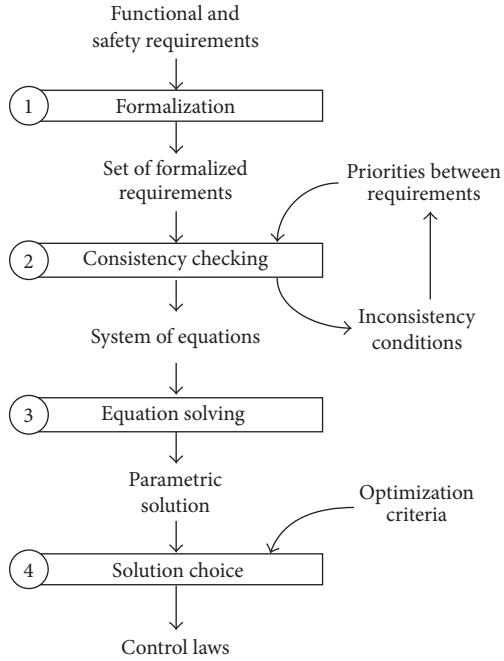


Figure 3: The algebraic synthesis method step by step.

The core of the method is the third step, which consists in the synthesis of the control laws. This step is performed by solving the system of equations which represents the set of consistent requirements. The mathematical results we have obtained (Theorem 12 given in Section 4.3), allow finding a parametric expression of the set of solutions.

In the fourth step of the method, a particular solution has to be chosen among the set of solutions. For that, a specific value of each parameter of the general solution has to be fixed. In a previous work [19], we showed how well chosen heuristics can be used for fixing these parameters. In this paper, we show that the choice of a particular solution among the set of solutions can be expressed as an optimization problem. We propose new theorems that allow calculating the maximum and the minimum of a Boolean formula, and we show how optimal solutions can be automatically found. For ergonomic reasons, the synthesized control laws can finally be displayed under the form of a finite automaton [21].

After the mathematical background of the method has been recalled, we are going to show how, in consideration of specific hypotheses, the second step of the method can be improved by a correction loop helping the designer to formalize the requirements and so improving the robustness of our synthesis method to the lack of precision of the specifications. The strategy to find an optimal solution according to given criteria will be also presented.

4. Mathematical Foundations

This section is composed of five subsections. Sections 4.1 and 4.2 recall some classical results about Boolean algebras and the Boolean algebra of n-variable switching functions. Section 4.3 presents how to solve Boolean equations. Sections 4.4 and 4.5 present specific results obtained for the algebraic synthesis of control laws.

4.1. Boolean Algebra: Typical Feature

Definition 1 (Boolean algebra). (Definition 15.5 of [22]) Let $\mathscr{B}$ be a nonempty set that contains two special elements 0 (the zero element) and 1 (the unity, or one, element) and on which we define two closed binary operations $+$, $\cdot$, and an unary operation $\overline{\ }$. Then $(\mathscr{B}, +, \cdot, \overline{\ }, 0, 1)$ is called a Boolean algebra if the following conditions are satisfied for all $x, y, z \in \mathscr{B}$:

$$\begin{aligned}
&\text{Commutative Laws:}\\
&\quad x + y = y + x\\
&\quad x \cdot y = y \cdot x\\
&\text{Distributive Laws:}\\
&\quad x \cdot (y + z) = (x \cdot y) + (x \cdot z)\\
&\quad x + (y \cdot z) = (x + y) \cdot (x + z)\\
&\text{Identity Laws:}\\
&\quad x + 0 = x\\
&\quad x \cdot 1 = x\\
&\text{Inverse Laws:}\\
&\quad x + \overline{x} = 1\\
&\quad x \cdot \overline{x} = 0\\
&\quad 0 \neq 1.
\end{aligned} \tag{2}$$

Many Boolean algebras could be defined. The most known are the two-element Boolean algebra: $(\{0, 1\}, \vee, \wedge, \neg, 0, 1)$ and the algebra of classes (set of subsets of a set S): $(2^S, \cup, \cap, \overline{\ }, \emptyset, S)$.

Definition 2 (Boolean formula). (From Section 3.6 of [15]) A *Boolean formula* (or a Boolean expression) on $\mathscr{B}$ is any formula which represents a combination of members of $\mathscr{B}$ by the operations $+$, $\cdot$, or $\overline{\ }$.

By construction, any Boolean formula on $\mathscr{B}$ represents one and only one member of $\mathscr{B}$. Two Boolean formulae are equivalent if and only if they represent the same member of $\mathscr{B}$. Later on, a Boolean formula $\mathscr{F}$ built with the members $(\alpha_1, \ldots, \alpha_n)$ of $\mathscr{B}$ is denoted $\mathscr{F}(\alpha_1, \ldots, \alpha_n)$.

Theorem 3 (Boole's expansion of a Boolean formula). *Let $(\alpha_1, \ldots, \alpha_n)$ be n members of $\mathscr{B} \setminus \{0, 1\}$. Any Boolean Formula $\mathscr{F}(\alpha_1, \ldots, \alpha_n)$ can be expanded as*

$$\mathscr{F}(\alpha_1, \ldots, \alpha_n) = \mathscr{F}_0(\alpha_2, \ldots, \alpha_n) \cdot \overline{\alpha_1} + \mathscr{F}_1(\alpha_2, \ldots, \alpha_n) \cdot \alpha_1, \tag{3}$$

where $\mathscr{F}_0(\alpha_2, \ldots, \alpha_n)$ and $\mathscr{F}_1(\alpha_2, \ldots, \alpha_n)$ are Boolean formulae of only $\alpha_2, \ldots, \alpha_n$. These two formulae can be directly obtained from $\mathscr{F}(\alpha_1, \ldots, \alpha_n)$ as follows:

$$\begin{aligned}
\mathscr{F}_0(\alpha_2, \ldots, \alpha_n) &= \mathscr{F}(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 0} = \mathscr{F}(0, \alpha_2, \ldots, \alpha_n)\\
\mathscr{F}_1(\alpha_2, \ldots, \alpha_n) &= \mathscr{F}(\alpha_1, \ldots, \alpha_n)\big|_{\alpha_1 \leftarrow 1} = \mathscr{F}(1, \alpha_2, \ldots, \alpha_n).
\end{aligned} \tag{4}$$

The relation *equality* is not the only defined relation on a Boolean algebra. It is also possible to define a partial order relation between members of $\mathscr{B}$. This relation is called *Inclusion-Relation* in [15].

Definition 4 (Inclusion-Relation). (Definition 15.6 of [22].) If $x, y \in \mathscr{B}$, define $x \leq y$ if and only if $x \cdot y = x$.

As Relation Inclusion is reflexive ($x \leq x$), antisymmetric (if $x \leq y$ and $y \leq x$, then $x = y$), and transitive (if $x \leq y$ and $y \leq z$, then $x \leq z$), this relation defines a partial order between members of $\mathscr{B}$ (Theorem 15.4 of [22]).

Since in any Boolean algebra, $x \cdot y = x \Leftrightarrow x \cdot \overline{y} = 0$, we also have $x \leq y \Leftrightarrow x \cdot \overline{y} = 0$.

Remark 5. For the algebra of classes $(2^S, \cup, \cap, \overline{\ }, \emptyset, S)$, the Inclusion-Relation is the well-known relation $\subseteq$ and we have: $x \subseteq y \Leftrightarrow x \cap y = x$.

Theorem 6 (Reduction of a set of relations). *(Theorem 5.3.1 of [15].) Any set of simultaneously asserted relations built with the members $(\alpha_1, \ldots, \alpha_n)$ of $\mathscr{B}$ can be reduced to a single equivalent relation such as: $\mathscr{F}(\alpha_1, \ldots, \alpha_n) = 0$.*

To obtain this equivalent relation, it is necessary

(i) to rewrite each equality according to

$$\begin{aligned}
&\mathscr{F}_1(\alpha_1, \ldots, \alpha_n) = \mathscr{F}_2(\alpha_1, \ldots, \alpha_n)\\
&\Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \cdot \overline{\mathscr{F}_2(\alpha_1, \ldots, \alpha_n)}\\
&\qquad + \overline{\mathscr{F}_1(\alpha_1, \ldots, \alpha_n)} \cdot \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0,
\end{aligned} \tag{5}$$

(ii) to rewrite each inclusion according to

$$\begin{aligned}
&\mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \leq \mathscr{F}_2(\alpha_1, \ldots, \alpha_n)\\
&\Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) \cdot \overline{\mathscr{F}_2(\alpha_1, \ldots, \alpha_n)} = 0,
\end{aligned} \tag{6}$$

(iii) to group together rewritten equalities as follows:

$$\begin{aligned}
&\begin{cases} \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) = 0\\ \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0 \end{cases}\\
&\Longleftrightarrow \mathscr{F}_1(\alpha_1, \ldots, \alpha_n) + \mathscr{F}_2(\alpha_1, \ldots, \alpha_n) = 0.
\end{aligned} \tag{7}$$

4.2. The Boolean Algebra of n-Variable Switching Functions. To avoid confusion between Boolean variables and Boolean functions of Boolean variables, each Boolean variable b_i is denoted by ${}_b b_i$. The set of the two Boolean values ${}_b 0$ and ${}_b 1$ is denoted by $B = \{{}_b 0, {}_b 1\}$.

Definition 7 (N-variable switching functions). (From Section 3.11 of [15].) An n-variable *switching function* is a mapping of the form

$$f: B^n \longrightarrow B$$
$$({}_b b_1, \ldots, {}_b b_n) \longmapsto f({}_b b_1, \ldots, {}_b b_n) \quad (8)$$
$$\text{where } B = \{{}_b 0, {}_b 1\}.$$

The domain of a n-variable switching function has 2^n elements and the codomain has 2 elements; hence, there are 2^{2^n} n-variable switching functions. Let $F_n(B)$ be the set of the 2^{2^n} n-variable switching functions.

$F_n(B)$ contains $(n + 2)$ specific n-variable switching functions: the 2 *constant functions* $(0, 1)$ and the *n projection-functions* (f^i_{Proj}). These functions are defined as follows:

$$\begin{aligned} 0&: B^n \longrightarrow B \\ &({}_b b_1, \ldots, {}_b b_n) \longmapsto {}_b 0 \\ 1&: B^n \longrightarrow B \\ &({}_b b_1, \ldots, {}_b b_n) \longmapsto {}_b 1 \\ f^{\text{i}}_{\text{Proj}}&: B^n \longrightarrow B \\ &({}_b b_1, \ldots, {}_b b_n) \longmapsto {}_b b_i, \end{aligned} \quad (9)$$

$F_n(B)$ can be equipped with three closed operations (two binary and one unary operations)

$$\begin{aligned} \text{Op. } +&: F_n(B)^2 \longrightarrow F_n(B) \\ &(f, g) \longmapsto f + g \\ \text{Op. } \cdot&: F_n(B)^2 \longrightarrow F_n(B) \\ &(f, g) \longmapsto f \cdot g \\ \text{Op. } {}^-&: F_n(B) \longrightarrow F_n(B) \\ &f \longmapsto \overline{f}, \end{aligned} \quad (10)$$

where $\forall({}_b b_1, \ldots, {}_b b_n) \in B^n$,

$$\begin{aligned} &(f + g)({}_b b_1, \ldots, {}_b b_n) \\ &\quad = f({}_b b_1, \ldots, {}_b b_n) \vee g({}_b b_1, \ldots, {}_b b_n), \\ &(f \cdot g)({}_b b_1, \ldots, {}_b b_n) \\ &\quad = f({}_b b_1, \ldots, {}_b b_n) \wedge g({}_b b_1, \ldots, {}_b b_n), \\ &\overline{f}({}_b b_1, \ldots, {}_b b_n) = \neg f({}_b b_1, \ldots, {}_b b_n). \end{aligned} \quad (11)$$

$(F_n(B), +, \cdot, {}^-, 0, 1)$ is a Boolean algebra [22]. Then, it is possible to write a Boolean formula of n-variable switching functions and relations between Boolean formula of n-variable switching functions. In the case of n-variable switching functions, relations Equality and Inclusion can also be presented as follows:

(i) f and g are equal ($f = g$) if and only if the columns of the truth-tables of f, g are exactly the same,

that is, $\forall({}_b b_1, \ldots, {}_b b_n) \in B^n$, $f({}_b b_1, \ldots, {}_b b_n) = g({}_b b_1, \ldots, {}_b b_n)$.

(ii) f is included into g ($f \leq g$) if and only if the value of g is always ${}_b 1$ when the value of f is ${}_b 1$,

that is, $\forall({}_b b_1, \ldots, {}_b b_n) \in B^n$, $[f({}_b b_1, \ldots, {}_b b_n) = {}_b 0]$, or $[g({}_b b_1, \ldots, {}_b b_n) = {}_b 1]$.

Remark 8. Each n-variable switching function can be expressed as a composition of $(f^1_{\text{Proj}}, \ldots, f^n_{\text{Proj}}, 0, 1)$ by operations $+, \cdot$ and ${}^-$.

Therefore, the Boolean algebra $(F_n(B), +, \cdot, {}^-, 0, 1)$ is a mathematical framework which allows composing and to comparing switching functions. Thanks to the results presented in the next subsection, this framework allows also solving Boolean equations systems of switching functions.

4.3. Solutions of Boolean Equations over Boolean Algebra $F_n(B)$. In [15], Brown explains that many problems in the application of Boolean algebra may be reduced to solving an equation of the form

$$f(X) = 0, \quad (12)$$

over a Boolean algebra $\mathscr{B}$. Formal procedures for producing solution of this equation were developed by Boole himself as a way to treat problems of logical inference. Boolean equations have been studied extensively since Boole's initial work (a bibliography of nearly 400 sources is presented in [14]). These works concern essentially the two-element Boolean algebra $(\{{}_b 0, {}_b 1\}, \vee, \wedge, \neg, {}_b 0, {}_b 1)$.

In our case, we focus on the Boolean algebra of n-variable switching functions $F_n(B)$. We consider a Boolean system composed of m relations among members of $F_n(B)$ for which k of them are considered as unknowns. Theorems presented in this section permit to solve any system of Boolean equations as it exists in a canonic form of a Boolean system of k unknowns and we are able to calculate solutions for this form.

4.3.1. Canonic Form of a Boolean System of k Unknowns over Boolean Algebra $F_n(B)$. Consider the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, {}^-, 0, 1)$.

(i) Let $(f^1_{\text{Proj}}, \ldots, f^n_{\text{Proj}})$ be the n projection-functions of $F_n(B)$.

(ii) Let $(x_1, \ldots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*.

For notational convenience, we note "X_k" as the vector $(x_1, \ldots, x_k)$ of the k unknowns and "Proj" as the vector $(f^1_{\text{Proj}}, \ldots, f^n_{\text{Proj}})$ of the n projection-functions of $F_n(B)$.

Table 1: Futures concerning a same case study.

	Formal requirements	Synthesized controller	PLC program (structured text)
Supervisory Control Theory	Plant behavior: 11 finite automata (481 states and 1330 transitions) Specifications: 11 finite automata	Finite automaton of 45 states and 70 transitions	130 lines
Algebraic synthesis	8 equations and 2 priority rules	2 6-variables switching functions	4 lines

Theorem 9 (Reduction of a set of relations between n-variable switching functions). *Any set of simultaneously asserted relations of switching functions can be reduced to a single equivalent relation such as*

$$\mathcal{F}\left(X_k, Proj\right) = 0. \tag{13}$$

This theorem comes from Theorem 6.

In order to be able to write a canonic form for a Boolean system of k unknowns over Boolean algebra $F_n(B)$, we introduce the following notation: for $x \in F_n(B)$ and $a \in \{0,1\}$, x^a is defined by

$$x^0 = \overline{x}, \qquad x^1 = x. \tag{14}$$

This notation is extended to vectors as follows: for $X_k = (x_1, \ldots, x_k) \in F_n(B)^k$ and $A_k = (a_1, \ldots, a_k) \in \{0,1\}^k$, $X_k^{A_k}$ is defined by

$$X_k^{A_k} = \prod_{i=1}^{i=k} x_i^{a_i} = x_i^{a_1} \cdot \cdots \cdot x_k^{a_k}. \tag{15}$$

Theorem 10 (Canonic form of a Boolean equation). *Any Boolean equation $Eq(X_k, Proj) = 0$ can be expressed within the canonic form*

$$\sum_{A_k \in \{0,1\}^k} Eq\left(A_k, Proj\right) \cdot X_k^{A_k} = 0, \tag{16}$$

where $Eq(A_k, Proj)$ (with $A_k \in \{0,1\}^k$) are the 2^k discriminants of $Eq(X_k, Proj) = 0$ according to X_k (the term of "discriminant" comes from [15]).

This canonic form is obtained by expanding $\mathrm{Eq}(X_k, \mathrm{Proj})$ according to the k unknowns $(x_1, \ldots, x_k)$. For example, we have

$$\begin{aligned}
\mathrm{Eq}\left(x, \mathrm{Proj}\right) &= \mathrm{Eq}\left(0, \mathrm{Proj}\right) \cdot \overline{x} + \mathrm{Eq}\left(1, \mathrm{Proj}\right) \cdot x, \\
\mathrm{Eq}\left(x_1, x_2, \mathrm{Proj}\right) &= \mathrm{Eq}\left(0, 0, \mathrm{Proj}\right) \cdot \overline{x_1} \cdot \overline{x_2} \\
&\quad + \mathrm{Eq}\left(0, 1, \mathrm{Proj}\right) \cdot \overline{x_1} \cdot x_2 \\
&\quad + \mathrm{Eq}\left(1, 0, \mathrm{Proj}\right) \cdot x_1 \cdot \overline{x_2} \\
&\quad + \mathrm{Eq}\left(1, 1, \mathrm{Proj}\right) \cdot x_1 \cdot x_2.
\end{aligned} \tag{17}$$

4.3.2. Solution of a Single-Unknown Equation over $F_n(B)$. The following theorem has initially been demonstrated for the two-element Boolean algebra [14]. A generalization for all Boolean algebras can be found in [15], but no detailed demonstration is given. A new formalization of this theorem and its full demonstration are given below.

Theorem 11 (Solution of a single-unknown equation). *The Boolean equation over $F_n(B)$*

$$Eq\left(x, Proj\right) = 0, \tag{18}$$

for which the canonic form is

$$Eq\left(0, Proj\right) \cdot \overline{x} + Eq\left(1, Proj\right) \cdot x = 0, \tag{19}$$

is consistent (i.e., has at least one solution) if and only if the following condition is satisfied:

$$Eq\left(0, Proj\right) \cdot Eq\left(1, Proj\right) = 0. \tag{20}$$

In this case, a general form of the solutions is

$$x = Eq\left(0, Proj\right) + p \cdot \overline{Eq\left(1, Proj\right)}, \tag{21}$$

where p is an arbitrary parameter, that is, a freely-chosen member of $F_n(B)$.

This solution can also be expressed as

$$\begin{aligned}
x &= \overline{Eq\left(1, Proj\right)} \cdot \left(Eq\left(0, Proj\right) + p\right) \\
&= \overline{p} \cdot Eq\left(0, Proj\right) + p \cdot \overline{Eq\left(1, Proj\right)}.
\end{aligned} \tag{22}$$

Proof. This theorem can be proved in four steps as follows:

(a) Equation (18) is consistent if and only if (20) is satisfied;

(b) Equation (21) is a solution of (18) if (20) is satisfied;

(c) each solution of (18) can be expressed as (21);

(d) if (20) is satisfied, the three parametric forms proposed are equivalent.

Step (a) can be proved as follows: Equation (20) is a sufficient condition for (18) to admit solutions since $x = \mathrm{Eq}(0, \mathrm{Proj})$ is an obvious solution of (18). Equation (20) is also a necessary condition as if (18) admits a solution, then (18) can be also expressed thanks to the consensus theorem as $\mathrm{Eq}(0, \mathrm{Proj}) \cdot \overline{x} + \mathrm{Eq}(1, \mathrm{Proj}) \cdot x + \mathrm{Eq}(0, \mathrm{Proj}) \cdot \mathrm{Eq}(1, \mathrm{Proj}) = 0$ and we have necessarily $\mathrm{Eq}(0, \mathrm{Proj}) \cdot \mathrm{Eq}(1, \mathrm{Proj}) = 0$.

To prove Step (b), it is sufficient to substitute the expression for x from (21) into (18) and to use (20) as follows:

$$\begin{aligned}
&\mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{x}+\mathrm{Eq}(1,\mathrm{Proj})\cdot x\\
&\quad=\mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{\left(\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)}\\
&\qquad+\mathrm{Eq}(1,\mathrm{Proj})\cdot\left(\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\\
&\quad=\mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\left(p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)}\\
&\qquad+\mathrm{Eq}(0,\mathrm{Proj})\cdot\mathrm{Eq}(1,\mathrm{Proj})\\
&\qquad+p\cdot\mathrm{Eq}(1,\mathrm{Proj})\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&\qquad=0+0+0=0.
\end{aligned}\tag{23}$$

To prove Step (c), it is sufficient to find one element p of $F_n(B)$ for each solution for x of (18). Let us consider p defined by "$p=\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot x$" where x is a solution to (18). Then we have

$$\begin{cases}\mathrm{Eq}(0,\mathrm{Proj})\cdot\mathrm{Eq}(1,\mathrm{Proj})=0\\ \mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{x}+\mathrm{Eq}(1,\mathrm{Proj})\cdot x=0\\ p=\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot x\end{cases}\tag{24}$$
$$\Longrightarrow x=\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}$$

as

$$\begin{aligned}
x&=1\cdot x=\Big(\mathrm{Eq}(0,\mathrm{Proj})+\mathrm{Eq}(1,\mathrm{Proj})\\
&\qquad+\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\Big)\cdot x\\
&=\mathrm{Eq}(0,\mathrm{Proj})\cdot x+\mathrm{Eq}(1,\mathrm{Proj})\cdot x\\
&\quad+\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot x\\
&=\mathrm{Eq}(0,\mathrm{Proj})\cdot x+0+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&\quad\cdot\left(\overline{\mathrm{Eq}(0,\mathrm{Proj})}\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot x\right)\\
&\qquad\text{as }\mathrm{Eq}(1,\mathrm{Proj})\cdot x=0\\
&=\mathrm{Eq}(0,\mathrm{Proj})\cdot x+\mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{x}\\
&\quad+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot p\quad\text{as }\mathrm{Eq}(0,\mathrm{Proj})\cdot\overline{x}=0\\
&=\mathrm{Eq}(0,\mathrm{Proj})\cdot(x+\overline{x})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}.
\end{aligned}\tag{25}$$

To prove Step (d), it is sufficient to rewrite (21) in the two other forms by using (20) as follows:

$$\begin{aligned}
x&=1\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=\left(\mathrm{Eq}(1,\mathrm{Proj})+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\cdot\mathrm{Eq}(0,\mathrm{Proj})\\
&\quad+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=\mathrm{Eq}(0,\mathrm{Proj})\cdot\mathrm{Eq}(1,\mathrm{Proj})+(\mathrm{Eq}(0,\mathrm{Proj})+p)\\
&\quad\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=0+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot(\mathrm{Eq}(0,\mathrm{Proj})+p)\\
&=\overline{\mathrm{Eq}(1,\mathrm{Proj})}\cdot(\mathrm{Eq}(0,\mathrm{Proj})+p),\\
x&=1\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=(p+\overline{p})\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}\\
&=\overline{p}\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\left(\mathrm{Eq}(0,\mathrm{Proj})+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\\
&=\overline{p}\cdot\mathrm{Eq}(0,\mathrm{Proj})\\
&\quad+p\cdot\left(\mathrm{Eq}(0,\mathrm{Proj})\cdot\mathrm{Eq}(1,\mathrm{Proj})+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\\
&=\overline{p}\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\left(0+\overline{\mathrm{Eq}(1,\mathrm{Proj})}\right)\\
&=\overline{p}\cdot\mathrm{Eq}(0,\mathrm{Proj})+p\cdot\overline{\mathrm{Eq}(1,\mathrm{Proj})}.
\end{aligned}\tag{26}$$

□

4.3.3. Solution of k-Unknown Equations over $F_n(B)$. The global result presented in the following theorem can be found in [14] or [15]. However, in these works, the solution is not expressed with a parametric form, but with intervals only. The formulation presented in this paper is more adapted to symbolic computation and is mandatory for practice optimization.

A k-unknown equation can be solved by solving successively k single-unknown equations. If we consider the k-unknown equation as a single-unknown equation of x_k, its consistence condition corresponds to a $(k-1)$-unknown equation. The process can be iterated until x_1. After substituting $S(x_1)$ for x_1 in the last equation, it is possible to find the solution for x_2. Then, it is sufficient to apply this procedure again $(k-2)$ times to obtain successively the solutions $S(x_3)$ to $S(x_k)$.

Theorem 12 (Solution of a k-unknown equation). *The Boolean equation over $F_n(B)$*

$$Eq_0(X_k,Proj)=0.\tag{27}$$

is consistent (i.e., has at least one solution) if and only if the following condition is satisfied:

$$\prod_{A_k\in\{0,1\}^k}Eq_0(A_k,Proj)=0.\tag{28}$$

If (28) *is satisfied,* (27) *admits one or more k-tuple solutions* $(S(x_1), \ldots, S(x_k))$ *such each component* $S(x_i)$ *is defined by*

$$S(x_i) = \prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(0, A_{k-i}, Proj) + p_i \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} Eq_{i-1}(1, A_{k-i}, Proj)}, \tag{29}$$

with

(i) $Eq_i(x_{i+1}, \ldots, x_k, Proj) = Eq_{i-1}(x_i, x_{i+1}, \ldots, x_k, Proj)|_{x_i \leftarrow S(x_i)}$

(ii) p_i *is an arbitrary parameter, that is, a freely-chosen member of* $F_n(B)$.

The full demonstration of this theorem cannot be given in this paper because of lack of space (a full demonstration by mathematical induction can be found in [8]). A description of the different steps of the proof and the detail of the principal steps are given below.

Proof (Elements of Proof). Equation (27) can be solved by applying Theorems 3 and 11 k times according to the unknowns x_k to x_1 as follows.

According to Theorem 3, (27) is equivalent to

$$\mathrm{Eq}_0(X_{k-1}, 0, \mathrm{Proj}) \cdot \overline{x_k} + \mathrm{Eq}_0(X_{k-1}, 1, \mathrm{Proj}) \cdot x_k = 0. \tag{30}$$

According to Theorem 11, (30) admits solutions in x_k if and only if

$$\mathrm{Eq}_0(X_{k-1}, 0, \mathrm{Proj}) \cdot \mathrm{Eq}_0(X_{k-1}, 1, \mathrm{Proj}) = 0. \tag{31}$$

Equation (31) is an equation with $(k-1)$ unknowns. Each term of (31) can be expanded according to x_{k-1} and (31) can be written in the form

$$\left(\mathrm{Eq}_0(X_{k-2}, 0, 0, \mathrm{Proj}) \cdot \mathrm{Eq}_0(X_{k-2}, 0, 1, \mathrm{Proj})\right) \cdot \overline{x_{k-1}} + \left(\mathrm{Eq}_0(X_{k-2}, 1, 0, \mathrm{Proj}) \cdot \mathrm{Eq}_0(X_{k-2}, 1, 1, \mathrm{Proj})\right) \cdot x_{k-1} = 0. \tag{32}$$

According to Theorem 11, (32) admits solutions in x_{k-1} if and only if

$$\prod_{A_2 \in \{0,1\}^2} \mathrm{Eq}_0(X_{k-2}, A_2, \mathrm{Proj}) = 0. \tag{33}$$

Equation (33) is an equation with $(k-2)$ unknowns. Each term of (33) can be expanded according to x_{k-2} and (33) can be written in the form

$$\left(\prod_{A_2 \in \{0,1\}^2} \mathrm{Eq}_0(X_{k-3}, 0, A_2, \mathrm{Proj})\right) \cdot \overline{x_{k-2}} + \left(\prod_{A_2 \in \{0,1\}^2} \mathrm{Eq}_0(X_{k-3}, 1, A_2, \mathrm{Proj})\right) \cdot x_{k-2} = 0. \tag{34}$$

In the end, we obtain an equation of only one unknown x_1 defined by

$$\left(\prod_{A_{k-1} \in \{0,1\}^{k-1}} \mathrm{Eq}_0(0, A_{k-1}, \mathrm{Proj})\right) \cdot \overline{x_1} + \left(\prod_{A_{k-1} \in \{0,1\}^{k-1}} \mathrm{Eq}_0(1, A_{k-1}, \mathrm{Proj})\right) \cdot x_1 = 0. \tag{35}$$

According to Theorem 11, (35) admits solutions if and only if

$$\prod_{A_k \in \{0,1\}^k} \mathrm{Eq}_0(A_k, \mathrm{Proj}) = 0. \tag{36}$$

When (36) is satisfied, the k equations for x_1 to x_k admit solutions. Equation (27) is then coherent and admits solutions.

When (36) is satisfied, solutions of (35) for x_1 are

$$S(x_1) = \prod_{A_{k-1} \in \{0,1\}^{k-1}} \mathrm{Eq}_0(0, A_{k-1}, \mathrm{Proj}) + p_1 \cdot \overline{\prod_{A_{k-1} \in \{0,1\}^{k-1}} \mathrm{Eq}_0(1, A_{k-1}, \mathrm{Proj})}. \tag{37}$$

After substituting $S(x_1)$ for x_1 into (27), we obtain a new equation $\mathrm{Eq}_1(x_2, \ldots, x_k, \mathrm{Proj}) = 0$ involving the $(k-1)$ unknowns $(x_2, \ldots, x_k)$, where

$$\mathrm{Eq}_1(x_2, \ldots, x_k, \mathrm{Proj}) = \mathrm{Eq}_0(x_1, x_2, \ldots, x_k, \mathrm{Proj})|_{x_1 \leftarrow S(x_1)}. \tag{38}$$

By applying the previous procedure, we can obtain $S(x_2)$ and $\mathrm{Eq}_2(x_3, \ldots, x_k, \mathrm{Proj})$. Then, it suffices to apply this procedure again $(k-2)$ times to obtain successively solutions $S(x_3)$ to $S(x_k)$. □

It is important to note that the order in which unknowns are treated affects only the parametric form of the k-tuple solution. This is due to the fact that the same k-tuple solution can be represented with several distinct parametric forms.

4.3.4. Partial Conclusions. Thanks to theorems presented above, it is possible to obtain a parametric representation of all the solutions of any set of simultaneously asserted relations with k unknowns, if a solution exists. In practice, due to the complexity of systems to be designed, proposed set of simultaneously asserted relations is generally inconsistent [23]. To simplify the work of the designer, we have proved complementary theorems to improve the robustness of our method to the lack of precision of the specifications (Section 4.4).

When several solutions exist, the comparison of solutions according to a given criterion can be envisaged since the Boolean algebra $F_n(B)$ is equipped with a partial order. To simplify the work of the designer too, we have developed a method to calculate the best solutions according to one or several criteria (Section 4.5).

4.4. Theorems to Cope with Inconsistencies of Specifications. In practice, it is very difficult for a designer to specify the whole requirements of a complex system without inconsistencies. It is the reason why requirements given by the designer are often declared as inconsistent according to Theorem 12. Since the inconsistency condition is a Boolean formula, it is possible to use it for the detection of the origin of inconsistencies. Two cases have to be considered as follows:

(i) Several requirements cannot be simultaneously respected. In this case, a hierarchy between requirements can be proposed in order to find a solution. The requirements which have the lower priority have to be corrected for becoming consistent with the requirements which have the higher priority. This strategy is based on Theorem 14.

(ii) The detected inconsistency refers to specific combinations of projection-functions for which the designer knows that they are impossible blocking the synthesis process, it is necessary to introduce new assumptions and to use Theorem 13.

Theorem 13 (Solution of a Boolean equation according to an assumption among the projection-functions). *The following problem*

$$\begin{aligned} &\textit{Equation to solve:} \\ &Eq_0\left(X_k, Proj\right) = 0 \\ &\textit{Assumptions:} \\ &\mathscr{A}\left(Proj\right) = 0 \end{aligned} \tag{39}$$

admits the same solutions as the following equation:

$$Eq_0\left(X_k, Proj\right) \leq \mathscr{A}\left(Proj\right). \tag{40}$$

Proof. According to $\mathscr{A}(\text{Proj}) = 0$, $\text{Eq}_0(X_k, \text{Proj}) = 0$ can be rewritten as

$$\begin{aligned} &\begin{cases} \text{Eq}_0\left(X_k, \text{Proj}\right) = 0 \\ \mathscr{A}\left(\text{Proj}\right) = 0 \end{cases} \Longleftrightarrow \mathscr{A}\left(\text{Proj}\right) + \text{Eq}_0\left(X_k, \text{Proj}\right) = 0 \\ &\Longleftrightarrow \mathscr{A}\left(\text{Proj}\right) + \overline{\mathscr{A}\left(\text{Proj}\right)} \cdot \text{Eq}_0\left(X_k, \text{Proj}\right) = 0 \\ &\Longleftrightarrow \begin{cases} \overline{\mathscr{A}(\text{Proj})} \cdot \text{Eq}_0\left(X_k, \text{Proj}\right) = 0 \\ \mathscr{A}\left(\text{Proj}\right) = 0 \end{cases} \\ &\Longleftrightarrow \begin{cases} \text{Eq}_0\left(X_k, \text{Proj}\right) \leq \mathscr{A}\left(\text{Proj}\right) \\ \mathscr{A}\left(\text{Proj}\right) = 0 \end{cases} \end{aligned} \tag{41}$$

Equation $\overline{\mathscr{A}(\text{Proj})} \cdot \text{Eq}_0(X_k, \text{Proj}) = 0$ is consistent if and only if the following condition is true (Theorem 12):

$$\overline{\mathscr{A}\left(\text{Proj}\right)} \cdot \prod_{A_k \in \{0,1\}^k} \text{Eq}_0\left(A_k, \text{Proj}\right) = 0. \tag{42}$$

By construction, this new condition is the subset of the initial condition ($\prod_{A_k \in \{0,1\}^k} \text{Eq}_0(A_k, \text{Proj}) = 0$) for which the proposed assumption is satisfied. All the other terms have been removed.

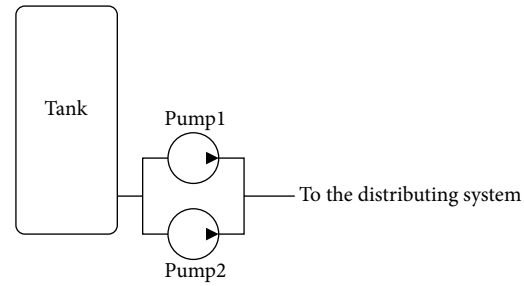


Figure 4: Structure of the water supply system.

If (42) is satisfied, (40) admits one or more k-tuple solutions where each component $S(x_i)$ is defined by

$$\begin{aligned} S\left(x_i\right) = &\ \overline{\mathscr{A}(\text{Proj})} \\ &\cdot \left(\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}\left(0, A_{k-i}, \text{Proj}\right) \right. \\ &\left. + p_i \cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}\left(1, A_{k-i}, \text{Proj}\right)} \right) \\ &+ \mathscr{A}\left(\text{Proj}\right) \cdot p_i. \end{aligned} \tag{43}$$

As $\mathscr{A}(\text{Proj}) = 0$, $S(x_i)$ can also be expressed as

$$\begin{aligned} S\left(x_i\right) = &\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}\left(0, A_{k-i}, \text{Proj}\right) + p_i \\ &\cdot \overline{\prod_{A_{k-i} \in \{0,1\}^{k-i}} \text{Eq}_{i-1}\left(1, A_{k-i}, \text{Proj}\right)}. \end{aligned} \tag{44}$$

When $\mathscr{A}(\text{Proj}) = 0$ is satisfied, the solutions of (40) are also solution to $\text{Eq}_0(X_k, \text{Proj}) = 0$. □

Theorem 14 (Solution of a Boolean equation system according to a priority rule between requirements). *The following problem*

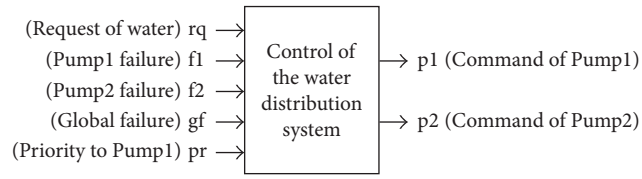
$$\begin{aligned} &\textit{Equations system to solve:} \\ &HR \quad \mathscr{F}_{\mathscr{H}}\left(X_k, Proj\right) = 0 \\ &LR \quad \mathscr{F}_{\mathscr{L}}\left(X_k, Proj\right) = 0 \\ &OR \quad \mathscr{F}_{\mathscr{O}}\left(X_k, Proj\right) = 0 \\ &\textit{Priority rule between requirements:} \\ &HR \gg LR, \end{aligned} \tag{45}$$

where

(i) $\mathscr{F}_{\mathscr{H}}(X_k, Proj) = 0$ *is the formal expression of the requirements which have the higher priority (HR);*

(ii) $\mathscr{F}_{\mathscr{L}}(X_k, Proj) = 0$ *is the formal expression of the requirements which have the lower priority (LR);*

(a) *Inputs and Outputs of the Controller*



(b) *General form of the Expected Control Laws*

$$\mathrm{p1}\,[k] = \mathrm{P1}\left(\mathrm{rq}\,[k], \mathrm{f1}\,[k], \mathrm{f2}\,[k]\,\mathrm{gf}\,[k], \mathrm{pr}\,[k], \mathrm{p1}\,[k-1], \mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p2}\,[k] = \mathrm{P2}\left(\mathrm{rq}\,[k], \mathrm{f1}\,[k], \mathrm{f2}\,[k]\,\mathrm{gf}\,[k], \mathrm{pr}\,[k], \mathrm{p1}\,[k-1], \mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p1}\,[0] = {}_b0 \qquad \mathrm{p2}\,[0] = {}_b0$$

(c) *Formal Specification*

Requirements:

R1 $\mathrm{P1} \cdot \mathrm{P2} = 0$ (* *The two pumps never operate simultaneously.* *)
R2 $\mathrm{F1} \leq \overline{\mathrm{P1}}$ (* *Pump 1 cannot operate if it is out of order (F1).* *)
R3 $\mathrm{F2} \leq \overline{\mathrm{P2}}$ (* *Pump2 cannot operate if it is out of order (F2).* *)
R4 $\mathrm{GF} \leq (\overline{\mathrm{P1}} \cdot \overline{\mathrm{P2}})$ (* *When a global failure is detected (GF), no pump can operate.* *)
R5 $(\mathrm{P1} + \mathrm{P2}) \leq \mathrm{Rq}$ (* *It is necessary to have are quest for pumps operate.* *)
R6 $\mathrm{Rq} \leq (\mathrm{P1} + \mathrm{P2})$ (* *It is sufficient to have a request for pumps operate.* *)

Priority rules:

R4 $\gg$ R6 (* *Failure requirements has priority on a functional requirement.* *)
{R2, R3} $\gg$ R6 (* *Failure requirements has priority on a functional requirement.* *)

Optimization criteria:

(1) Minimization of: $((\mathrm{P1} \cdot \overline{{}_pP1}) + (\mathrm{P2} \cdot \overline{{}_pP2}))$ (* *Minimization of the possibility to start a pump.* *)
(2) Maximization of: $((\mathrm{Pr} \cdot \mathrm{P1}) + (\overline{\mathrm{Pr}} \cdot \mathrm{P2}))$ (* *Maximization of the priority order between the two pumps.* *)

(d) *Solution obtained by symbolic calculation*

$$\mathrm{P1} = \mathrm{Rq} \cdot \overline{\mathrm{GF}} \cdot \overline{\mathrm{F1}} \cdot \left(\mathrm{F2} + \mathrm{Pr} \cdot \left({}_pP1 + \overline{{}_pP2}\right) + {}_p\mathrm{P}\,1 \cdot \overline{{}_pP2}\right)$$

$$\mathrm{P2} = \mathrm{Rq} \cdot \overline{\mathrm{GF}} \cdot \overline{\mathrm{F2}} \cdot \left(\mathrm{F1} + \overline{\mathrm{Pr}} \cdot \left({}_pP2 + \overline{{}_pP1}\right) + {}_p\mathrm{P}\,2 \cdot \overline{{}_pP1}\right)$$

(e) *Control laws of the water distribution system*

$$\mathrm{p1}\,[k] = \mathrm{rq}\,[k] \wedge \neg\mathrm{gf}\,[k] \wedge \neg\mathrm{f1}\,[k] \wedge \left(\mathrm{f2}\,[k] \vee \mathrm{pr}\,[k] \wedge \left(\mathrm{p1}\,[k-1] \vee \neg\mathrm{p2}\,[k-1]\right) \vee \mathrm{p1}\,[k-1] \wedge \neg\mathrm{p2}\,[k-1]\right)$$

$$\mathrm{p2}\,[k] = \mathrm{rq}\,[k] \wedge \neg\mathrm{gf}\,[k] \wedge \neg\mathrm{f1}\,[k] \wedge \left(\mathrm{f1}\,[k] \vee \mathrm{pr}\,[k] \wedge \left(\mathrm{p2}\,[k-1] \vee \neg\mathrm{p1}\,[k-1]\right) \vee \mathrm{p2}\,[k-1] \wedge \neg\mathrm{p1}\,[k-1]\right)$$

$$\mathrm{p1}\,[0] = {}_b0 \qquad \mathrm{p2}\,[0] = {}_b0$$

Figure 5: Details of the case study.

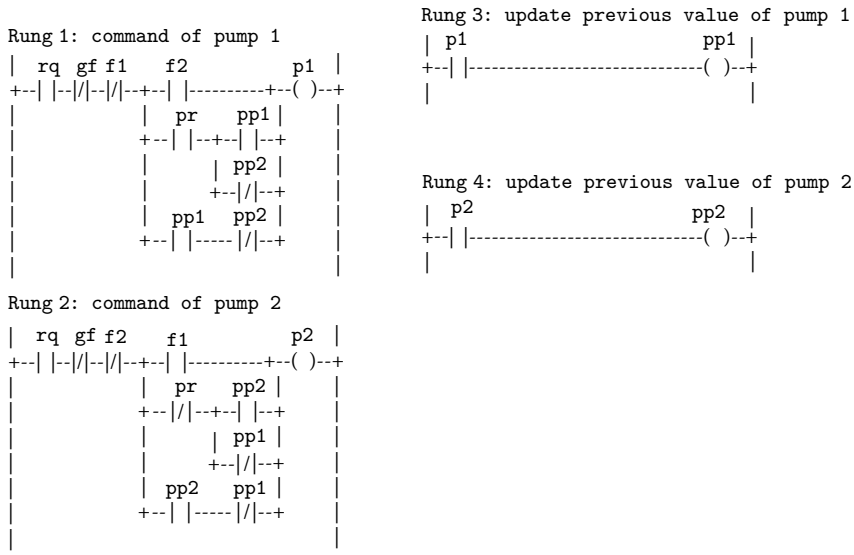


Figure 6: Ladder diagram of the code to implement into the PLC.

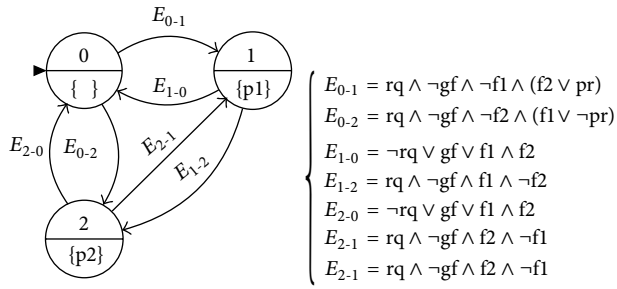


Figure 7: State model of the obtained control law.

(iii) $\mathcal{F}_{\mathcal{O}}(X_k, Proj) = 0$ *is the formal expression of the others requirements (OR);*

(iv) *HR* $\gg$ *LR is the priority rule between inconsistent requirements,*

admits the same solutions as the system of equations as follows:

$$\begin{aligned} &\mathcal{F}_{\mathcal{H}}\left(X_k, Proj\right) = 0 \\ &\mathcal{F}_{\mathcal{L}}\left(X_k, Proj\right) \leq \mathcal{I}\left(Proj\right) \\ &\mathcal{F}_{\mathcal{O}}\left(X_k, Proj\right) = 0, \end{aligned} \tag{46}$$

where $\mathcal{I}(Proj)$ *is the inconsistency condition between requirements "HR" and "LR":*

$$\mathcal{I}\left(Proj\right) = \prod_{A_k \in \{0,1\}^k} \left(\mathcal{F}_{\mathcal{H}}\left(A_k, Proj\right) + \mathcal{F}_{\mathcal{L}}\left(A_k, Proj\right)\right). \tag{47}$$

Proof. Thanks to Theorem 12, the inconsistency condition $\mathcal{I}(\text{Proj})$ between requirements "HR" and "LR" can be found by solving equation $\mathcal{F}_{\mathcal{H}}(X_k, \text{Proj}) + \mathcal{F}_{\mathcal{L}}(X_k, \text{Proj}) = 0$. We have

$$\mathcal{I}\left(\text{Proj}\right) = \prod_{A_k \in \{0,1\}^k} \left(\mathcal{F}_{\mathcal{H}}\left(A_k, \text{Proj}\right) + \mathcal{F}_{\mathcal{L}}\left(A_k, \text{Proj}\right)\right). \tag{48}$$

To remove the inconsistency between requirements "HR" and "LR" according to the priority rule "HR $\gg$ LR", it is necessary to restrict the range of requirement "LR" to the part for which there is no inconsistency, that is, $\mathcal{I}(\text{Proj}) = 0$. That is the case, when $\mathcal{F}_{\mathcal{L}}(X_k, \text{Proj}) = 0$ is replaced by $\mathcal{F}_{\mathcal{L}}(X_k, \text{Proj}) \leq \mathcal{I}(\text{Proj})$.

Thanks to Theorem 12, (49) admits always one or more k-tuple solutions and it is impossible to find a less restrictive condition over requirement "LR".

$$\begin{aligned} &\mathcal{F}_{\mathcal{H}}\left(X_k, \text{Proj}\right) = 0 \\ &\mathcal{F}_{\mathcal{L}}\left(X_k, \text{Proj}\right) \leq \mathcal{I}\left(\text{Proj}\right). \end{aligned} \tag{49}$$

□

4.5. Optimal Solutions of Boolean Equations over $F_n(B)$. The goal of this step is to be able to obtain automatically the parametric form of the k-tuples solutions of $F_n(B)$ which satisfy not only a given equation ($\text{Eq}(X_k, \text{Proj}) = 0$) of Boolean functions but also which maximize (or minimize) a Boolean formula of these Boolean functions ($\mathcal{F}_C(X_k, \text{Proj})$) corresponding to the desired optimization criterion.

Generally speaking, the search of the best solution tuples according to a given criterion when the space of solutions is composed of discrete values is a complex mathematical issue. It is sometimes necessary to make a side-by-side comparison of each solution in order to identify the best one. In our case, this exhaustive method which cannot be used as $F_n(B)$ is only provided by a partial order; two particular solutions cannot always be ordered between themselves.

Nevertheless, it is possible to obtain the researched parametric form of the k-tuples thanks to the following results.

(i) When an equation between Boolean functions has one or more solution tuples in $F_n(B)$, every Boolean formula onto these Boolean functions can be rewritten thanks to only projection-functions of $F_n(B)$ and free parameters of $F_n(B)$ which are describing these solution tuples.

(ii) Every Boolean formula expressed as a composition of projection-functions of $F_n(B)$ and free parameters of $F_n(B)$ has a unique maximum and a unique minimum. These extrema can be expressed thanks to only projection-functions of $F_n(B)$.

Hence it is then possible to rewrite the initial problem

$$\begin{aligned} &\text{Problem to solve:} \\ &\quad \mathrm{Eq}\left(X_k, \mathrm{Proj}\right) = 0 \\ &\text{Optimization Criterion:} \\ &\quad \text{Maximization of } \mathscr{F}_{\mathrm{C}}\left(X_k, \mathrm{Proj}\right), \end{aligned} \tag{50}$$

into a 2-equation system to solve

$$\begin{aligned} &\mathrm{Eq}\left(X_k, \mathrm{Proj}\right) = 0 \\ &\mathscr{F}_C\left(X_k, \mathrm{Proj}\right) = \max_{\{X_k \mid \mathrm{Eq}(X_k, \mathrm{Proj})=0\}} \left(\mathscr{F}_C\left(X_k, \mathrm{Proj}\right)\right). \end{aligned} \tag{51}$$

4.5.1. Extrema of a Boolean Formula according to Freely Chosen Members of $F_n(B)$. Considering the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$,

(i) let $(f^1_{\mathrm{Proj}}, \ldots, f^n_{\mathrm{Proj}})$ be the *n projection-functions* of $F_n(B)$;

(ii) let $(p_1, \ldots, p_k)$ be k elements of $F_n(B)$ considered as *freely chosen members*. Let "P_k" be the corresponding vector.

Any formula $\mathscr{F}(P_k, \mathrm{Proj})$ for which P_k are freely chosen members of $F_n(B)$ defines a subset of $F_n(B)$. According to the relation $\leq$, elements of this subset can be compared. In this specific case, the subset defined by $\mathscr{F}(P_k, \mathrm{Proj})$ admits a minimal element and a maximal element.

Theorem 15 (Minimum and Maximum of a Boolean formula). *Any formula $\mathscr{F}(P_k, Proj)$ for which P_k are freely chosen members of $F_n(B)$ admits a minimum and a maximum defined as follows:*

$$\begin{aligned} \min_{P_k \in F_n(B)^k} \left(\mathscr{F}\left(P_k, Proj\right)\right) &= \prod_{A_k \in \{0,1\}^k} \mathscr{F}\left(A_k, Proj\right) \\ \max_{P_k \in F_n(B)^k} \left(\mathscr{F}\left(P_k, Proj\right)\right) &= \sum_{A_k \in \{0,1\}^k} \mathscr{F}\left(A_k, Proj\right), \end{aligned} \tag{52}$$

Proof. To prove this theorem, it is necessary to establish that

(1) $\prod_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$ is a lower bound of $\mathscr{F}(P_k, \mathrm{Proj})$;

(2) It exists at least one specific combination of P_k for which $\mathscr{F}(P_k, \mathrm{Proj}) = \prod_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$;

(3) $\sum_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$ is an upper bound of $\mathscr{F}(P_k, \mathrm{Proj})$;

(4) It exists at least one specific combination of P_k for which $\mathscr{F}(P_k, \mathrm{Proj}) = \sum_{A_k \in \{0,1\}^k} \mathscr{F}(A_k, \mathrm{Proj})$.

Details of this proof can be found in [24]. □

4.5.2. Optimization Problem. Considering the Boolean algebra of n-variable switching functions $(F_n(B), +, \cdot, \bar{}, 0, 1)$,

(i) let $(f^1_{\mathrm{Proj}}, \ldots, f^n_{\mathrm{Proj}})$ be the *n projection-functions* of $F_n(B)$. Let "Proj" be the corresponding vector;

(ii) Let $(x_1, \ldots, x_k)$ be k elements of $F_n(B)$ considered as *unknowns*. Let "X_k" be the corresponding vector;

(iii) Let $(p_1, \ldots, p_k)$ be k elements of $F_n(B)$ considered as *freely chosen members*. Let "P_k" be the corresponding vector.;

(iv) Let $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ be the Boolean equation to solve;

(v) Let $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ be the Boolean formula of the given criterion to optimize (maximization or minimization).

The method we propose, to obtain the parametric form of the k-tuple of switching functions solution of $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ according to a given optimization criterion $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ is composed of five steps as follows.

(i) The first step is to establish the parametric form of the k-tuple solution to $\mathrm{Eq}(X_k, \mathrm{Proj}) = 0$ only, thanks to Theorem 12.

(ii) The second step is to establish the parametric form of the given optimization criterion $\mathscr{F}_{\mathrm{C}}(X_k, \mathrm{Proj})$ by substituting $S(x_i)$ for x_i. Let $\mathscr{F}_{\mathrm{SC}}(P_k, \mathrm{Proj})$ be the result of this substitution.

(iii) The third step is to calculate the extremum of $\mathscr{F}_{\mathrm{SC}}(P_k, \mathrm{Proj})$ according to Theorem 15. Let $\mathscr{F}_{\mathrm{EC}}(\mathrm{Proj})$ be the Boolean formula of this extremum.

(iv) The fourth step is to replace the given criterion by the equivalent relation

$$\mathscr{F}_C\left(X_k, \mathrm{Proj}\right) = \mathscr{F}_{\mathrm{EC}}\left(\mathrm{Proj}\right). \tag{53}$$

(v) The fifth step is to establish the parametric form of the k-tuple solution of the equivalent problem

$$\begin{aligned} &\mathrm{Eq}\left(X_k, \mathrm{Proj}\right) = 0 \\ &\mathscr{F}_{\mathrm{Crit}}\left(X_k, \mathrm{Proj}\right) = \mathscr{F}_{\mathrm{ExtCrit}}\left(\mathrm{Proj}\right). \end{aligned} \tag{54}$$

4.5.3. Partial Conclusions. Thanks to theorems presented in this section, it is now possible to obtain a parametric representation of the optimal solutions according to a given criterion, of any set of simultaneously asserted relations with k unknowns if a solution exists.

The proposed method also permits to associate simultaneously or sequentially several criteria.

(i) When several criteria are treated simultaneously, the optimization problem can admit no solution. That is the case when the given criteria are antagonist.

(ii) When several criteria are treated sequentially, the obtained solutions satisfy the criteria with a given priority order. An example of optimization with several criteria treated sequentially is presented in the next section.

5. Algebraic Synthesis of Logical Controllers with Optimization Criteria and Incoherent Requirements

5.1. Control System Specifications. The studied system is the controller of a water supply system composed of two pumps which are working in redundancy (Figure 4). The water distribution is made when it is necessary according to the possible failures of elements (the pumps and the distributing system).

The expected behavior of the control system regarding the application requirements can be expressed by the set of assertions given hereafter:

(i) The two pumps never operate simultaneously.

(ii) A pump cannot operate if it is out of order.

(iii) When a global failure is detected, no pump can operate.

(iv) Pumps can operate if and only if a water distribution request is present.

(v) Priority is given according to "pr" (pump1 has priority when "pr" is true).

(vi) In order to reduce the wear of the pumps, it is necessary to restrict the number of starting of the pumps.

5.1.1. Inputs and Outputs of the Controller. The Boolean inputs and outputs of this controller are given in Figure 5(a). Each pump is controlled thanks to a Boolean output ("p1" and "p2"). The controller is informed of water distribution requests thanks to the input "req." Inputs "f1" and "f2" inform the controller of a failure of the corresponding pump and "gf" indicates a global failure of the installation. The values o or 1 of input "Pr" decide which pump has priority.

5.1.2. Control Laws to Synthetize. Our approach does not allow identifying automatically which state variables must be used. They are given by the designer according to its interpretation of the specification.

For the water distribution system, we propose to use 2 state variables, one for each output. According to this choice, 2 7-variable switching functions ($P1$ and $P2$) have to be synthesized (Figure 5(b)). They represent the unknowns of our problem. For this case study, the 7 projection-functions of $F_7(B)$ are therefore as follows.

(i) The 5 switching functions (Rq, F1, F2, GF, and Pr) which characterize the behavior of the inputs of the controller and are defined as follows:

$$\begin{aligned} &\mathrm{Rq}: B^7 \longrightarrow B \\ &\quad (\mathrm{rq}[k], \ldots, \mathrm{p2}[k-1]) \longmapsto \mathrm{rq}[k]. \end{aligned} \tag{55}$$

(ii) The 2 switching functions ($_pP1$ and $_pP2$) which characterize the previous behavior of the state variables of the controller and are defined as follows:

$$\begin{aligned} &{}_pP1: B^7 \longrightarrow B \\ &\quad (\mathrm{rq}[k], \ldots, \mathrm{p2}[k-1]) \longmapsto \mathrm{p1}[k-1]. \end{aligned} \tag{56}$$

5.2. Algebraic Formalization of Requirements. The complete formalization of the behavior of the water distribution system is given in Figure 5(c). In order to illustrate the power of expression of relations Equality and Inclusion, several examples (generic assertions and equivalent formal relations illustrated in the case study) are given hereafter. It is important to note that the relation Inclusion permits to express distinctly necessary conditions and sufficient conditions. This relation is the cornerstone of our approach.

(i) Pump1 and Pump2 never operate simultaneously: $P1 \cdot P2 = 0$;

(ii) If Pump1 operates, Pump2 cannot operate: $P1 \leq \overline{P2}$;

(iii) It is necessary to have a request for pumps operate: $(P1 + P2) \leq \mathrm{Rq}$;

(iv) It is sufficient to have a request for pumps operate: $\mathrm{Rq} \leq (P1 + P2)$;

(v) When Pump1 is failed, it is sufficient to have a request for Pump2 operate: $\mathrm{F1} \cdot \mathrm{Rq} \leq P2$;

(vi) When Pump1 is failed, it is necessary to have a request for Pump2 operate: $\mathrm{F1} \cdot P2 \leq \mathrm{Rq}$.

It is possible to prove that some of these formal expressions are equivalent (e.g., the first two). When a designer hesitates between two forms, he has the possibility, by using symbolic calculation, to check if the proposed relations are equivalent or not.

As $P1$ and $_pP1$ represent the behavior of pump1 at, respectively, times $[k]$ and $[k-1]$, it is also possible to express relations about starts and stops of this pump as follows.

(i) It is necessary to have a request to start pump1: $(P1 \cdot \overline{{}_pP1}) \leq \mathrm{Rq}$

(ii) When pump1 operates, it is sufficient to have a global failure to stop pump1: $({}_pP1 \cdot \mathrm{GF}) \leq (\overline{P1} \cdot {}_pP1)$.

5.3. Synthesis Process. In traditional design methods, the design procedure of a logic controller is not a linear process, but an iterative one converging to an acceptable solution. At the beginning of the design, requirements are neither complete nor without errors. Most often, new requirements are added during the search of solutions, and others are corrected. This complementary information is given by the designer after analysis of the partial solutions he found or when inconsistencies have been detected. If we do not make the hypothesis that the specifications are complete and consistent, designing a controller with a synthesis technique is also an iterative process in which the designer plays an important role.

5.3.1. Analysis of Requirements. For this case study, we choose to start with requirements R1 to R6. For this subset of requirements, the result given by your software tool was the following inconsistency condition: $\mathcal{F} = \mathrm{Rq} \cdot \mathrm{GF} + \mathrm{Rq} \cdot \mathrm{F1} \cdot \mathrm{F2}$.

Since requirements are declared inconsistent, we have to give complementary information to precise our specification. By analyzing each term of this formula, it is possible to detect the origin of the inconsistency:

(i) $\mathrm{Rq} \cdot \mathrm{GF}$: what happens if we have simultaneously a request and a global failure? We consider that requirement R4 is more important than requirement R6 ($\mathrm{R4} \gg \mathrm{R6}$) as no pump can operate for this configuration.

(ii) $\mathrm{Rq} \cdot \mathrm{F1} \cdot \mathrm{F2}$: what happens if we have simultaneously a request and a failure of each pump? We consider that requirements R2 and R3 are more important than requirement R6 ($\{\mathrm{R2}, \mathrm{R3}\} \gg \mathrm{R6}$).

With these priority rules, all the requirements are now coherent and the set of all the solutions can be computed.

5.3.2. Optimal Solutions. For choosing a control law of the water supply system among this set of possible solutions, we will now take into account the given optimization criteria. The first criterion aims at minimizing the number of starting of each pump in order to reduce its wear. The second criterion aims at maximizing the use of the pump indicated by the value of parameter Pr. The method we propose allows proving that proposed criteria cannot be treated simultaneously since they are antagonist (to strictly the priority use of the pump fixed by parameter Pr, it is necessary to permute pumps when Pr changes of value, implying a supplementary start of a pump). Details can be found in [25].

All the priorities rules and optimization criteria used for this case study are given in Figure 5(c). The solution we obtain is proposed in Figure 5(d).

5.3.3. Implementing Control Laws. The synthesized control laws presented in Figure 5(e) have been obtained by translating the expression of the two unknowns according to the projection-functions into relations between recurrent Boolean equations. These control laws can be automatically translated in the syntax of the ladder diagram language [1] before being implemented into a PLC. The code is composed of only 4 rungs (Figure 6).

The synthesized control laws can be given under the form of an automatically built input/output automaton with guarded transitions [21] (Figure 7).

6. Discussion

In our approach, the synthesis of control laws is based on the symbolic calculation, a prototype software tool has been developed to avoid tedious calculus and to aid the designer during the different steps of the synthesis. This tool (that can be obtained on request by the authors) performs all the computations required for inconsistencies detection between requirements and for control laws generation. In this tool, all the Boolean formulas are stored in the form of reduced ordered binary decision diagrams, which allows efficient calculations. For example, the computations for synthesizing a controller for the water supply system that we developed above have been made in less than 10 ms onto a classical laptop.

Our approach has been tested on several studies cases (some of them are available online: http://www.lurpa.ens-cachan.fr/-226050.kjsp.). The feedbacks of these experiences allowed us to identify some of its limits and its possibilities; the most important are given below.

We have first to recall that our method can only be used for binary systems (systems whose inputs and outputs of their controller are Boolean values). Nevertheless, in practice many systems, like manufacturing systems, transport systems, and so on, are fully or partially binary.

In our opinion the main advantage of our approach is that, contrary to traditional engineering approaches, the synthesized control laws are not depending on designer's skill or of his correct interpretation of the system requirements. On the other hand, the quality of the synthesis results highly depends on the relevance of the requirements proposed by the designer. This step of formalization, by the designer, of the informal requirements of the system to be controlled is the Achilles heel of all synthesis methods, including the Supervisory Control Theory (SCT), and cannot be automated.

The objective comparison of our approach with other synthesis methods, and more especially with SCT, is very difficult because the models used and the theoretical basics are very different. Nevertheless, we tested both approaches on same study cases. One of them, the control of an automatic parking gate, has been published in [26]. The results obtained in this case are summarized in Table 1.

Furthermore, one may note that the supervisor that is synthesized by SCT is optimal in the sense where it is the most permissive; that is, the one that reduces the less the plant behavior in order to force it to respect the specifications. As shown in this paper our method allows to cope with inconsistencies in specifications, what is not possible with SCT, and also allows to find optimal controllers by choosing different optimization criteria (most permissive, most restrictive, most safe controller, etc.).

7. Conclusion

Many research works in the field of DES aim at formalizing steps of the systems life cycle. Since 20 years, significant progresses have been obtained for the synthesis, verification, performance evaluation, and diagnosis of DESs. Nevertheless, one of the common difficulties of these works is the translation of informal expression of the knowledge of a system into formal requirements. Few works have paid attention to this important task which is very error prone. In this paper, we proposed an iterative process that allows coping with inconsistencies of the requirements during the synthesis of the controller. The framework in which we proposed this approach is an algebraic synthesis method. Since the problem is located in the frontier between formal and informal, intervention of the designer is necessary. Nevertheless, we have shown that this intervention can be guided by the results of the formal method provides.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] International Electrotechnical Commission, *IEC 61131-3, IEC 61131-3 Standard: Programmable Controllers-Part 3: Programming Languages*, International Electrotechnical Commission, 2nd edition, 2003.

[2] G. Frey and L. Litz, "Formal methods in PLC programming," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 2431–2436, October 2000.

[3] B. Berard, M. Bidoit, A. Finkel et al., *Systems and Software Verification: Model-Checking Techniques and Tools*, Springer, New York, NY, USA, 1st edition, 1999.

[4] H. Bel Mokadem, B. Bérard, V. Gourcuff, O. De Smet, and J. Roussel, "Verification of a timed multitask system with UPPAAL," *IEEE Transactions on Automation Science and Engineering*, vol. 7, no. 4, pp. 921–932, 2010.

[5] J. L. Boulanger, Ed., *Industrial Use of Formal Methods: Formal Verification (ISTE)*, Wiley-ISTE, New York, NY, USA, 2012.

[6] P. J. G. Ramadge and W. M. Wonham, "Control of discrete event systems," *Proceedings of the IEEE*, vol. 77, no. 1, pp. 81–98, 1989.

[7] P. Gohari and W. M. Wonham, "On the complexity of supervisory control design in the RW framework," *IEEE Transactions on Systems, Man, and Cybernetics B*, vol. 30, no. 5, pp. 643–652, 2000.

[8] Y. Hietter, *Synthése algèbrique de lois de commande pour les systemes a évènements discrets logiques [Ph.D. thesis]*, ENS Cachan, Cachan, France, 2009.

[9] H.-M. Hanisch, A. Lueder, and J. Thieme, "Modular plant modeling technique and related controller synthesis problems," in *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 686–691, October 1998.

[10] J.-M. Roussel and A. Giua, "Designing dependable logic controllers using the supervisory control theory," in *Proceedings of the 16th IFACWorld Congress*, cDRom paper 4427, p. 6, Praha, Czech Republic, 2005.

[11] D. A. Huffman, "The synthesis of sequential switching circuits," *Journal of the Franklin Institute*, vol. 257, no. 3-4, pp. 161–303, 1954.

[12] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[13] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, pp. 129–153, Princeton University Press, Princeton, NJ, USA, 1956.

[14] S. Rudeanu, *Lattice Functions and Equations (Discrete Mathematics and Theoretical Computer Science)*, Springer, New York, NY, USA, 2001.

[15] F. M. Brown, *Boolean Reasoning: the Logic of Boolean Equations*, Dover, Mineola, NY, USA, 2003.

[16] A. Pnueli and R. Rosner, "On the synthesis of a reactive module," in *Proceedings of the 16th ACM symposium on Principles of programming languages (POPL '89)*, pp. 179–190, ACM, New York, NY, USA, 1989.

[17] E. Filiot, N. Jin, and J. Raskin, "Antichains and compositional algorithms for LTL synthesis," *Formal Methods in System Design*, vol. 39, no. 3, pp. 261–296, 2011.

[18] J. Machado, B. Denis, J. J. Lesage, J. M. Faure, and J. Fereira, "Logic controllers dependability verification using a plant model," in *Proceedings of the 3rd IFAC Workshop on Discrete-Event System Design (DESDes '06)*, pp. 37–42, Rydzyna, Poland, 2006.

[19] Y. Hietter, J.-M. Roussel, and J.-L. Lesage, "Algebraic synthesis of transition conditions of a state model," in *Proceedings of the 9th International Workshop on Discrete Event Systems (WODES '08)*, pp. 187–192, Göteborg, Sweden, May 2008.

[20] Y. Hietter, J.-M. Roussel, and J. J. Lesage, "Algebraic synthesis of dependable logic controllers," in *Proceedings of the 17th World Congress, International Federation of Automatic Control (IFAC '08)*, pp. 4132–4137, Seoul, South Korea, July 2008.

[21] A. Guignard, *Symbolic generation of the automaton representing an algebraic description of a logic system [M.S. thesis]*, ENS Cachan, Cachan, France, 2011.

[22] R. P. Grimaldi, *Discrete and Combinatorial Mathematics: An Applied Introduction*, Addison-Wesley Longman, Boston, Mass, USA, 5th edition, 2004.

[23] J.-M. Roussel and J.-J. Lesage, "Algebraic synthesis of logical controllers despite inconsistencies in specifications," in *Proceeding of the 11th International Workshop on Discrete Event Systems (WODES '12)*, pp. 307–314, Guadalajara, Mexico, 2012.

[24] H. Leroux, *Algebraic Synthesis of Logical Controllers with Optimization Criteria*, ENS Cachan, Cachan, France, 2011.

[25] H. Leroux and J. -M. Roussel, "Algebraic synthesis of logical controllers with optimization criteria," in *Proceedings of the 6th International Workshop on Verification and Evaluation of Computer and Communication Systems (VECOS '12)*, pp. 103–114, Paris, France, 2012.

[26] M. Cantarelli and J. Roussel, "Reactive control system design using the supervisory control theory: evaluation of possibilities and limits," in *Proceedings of 9th International Workshop on Discrete Event Systems (WODES '08)*, pp. 200–205, Göteborg, Sweden, May 2008.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

Generation of Single Input Change Test Sequences for Conformance Test of Programmable Logic Controllers

Julien Provost, *Member, IEEE*, Jean-Marc Roussel, Jean-Marc Faure, *Member, IEEE*

***Abstract*—Conformance test is a functional test technique which is aiming to check whether an implementation, seen as a black-box with inputs/outputs, conforms to its specification. Numerous theoretical worthwhile results have been obtained in the domain of conformance test of finite state machines. The optimization criterion which is usually selected to build the test sequence is the minimum-length criterion. Based on experimental results, this paper focuses on the generation of a Single Input Change (SIC) test sequence from a specification model represented as a Mealy machine; such a sequence is aiming at preventing from erroneous test verdicts due to incorrect detection of synchronous input changes by the Programmable Logic Controller (PLC) under test. A method based on symbolic calculus to obtain the part of the specification that can be tested with a SIC sequence is first presented. Then, an algorithm to build the SIC test sequence is detailed; three solutions are proposed, according to the connectivity properties of the SIC-testable part.**

***Index Terms*—Conformance test, Formal Methods, Programmable Logic Controller, Test Sequence, Mealy machine, Test Verdict, Single Input Change**

I. Introduction

PROGRAMMABLE LOGIC CONTROLLERS (PLCs) are industrial automation components that are widely used to implement control functions, even in critical systems like power production and distribution, rail transport, chemical processes, water distribution, etc. This explains why numerous research works have been carried out since more than ten years to develop methods that avoid flaws to be introduced during the development of PLC software [1]. These researches are based on two main approaches: model-based (model-driven) engineering [2]–[5] and formal verification and validation (V&V) techniques [6]–[9] or a combination of both [10]. Whatever the interest of the results obtained in these works, it must be noted that all of them are based on *models*. Formal V&V techniques for instance have been applied to models of the specification of the control logic, in the form of IEC 60848 models, state-charts, Signal Interpreted Petri Nets, Net Condition Event Systems [11]–[13] or of PLC programs, in IEC 61131-3 or IEC 61499 languages [14]–[19].

However, validation of a *real* PLC which executes a control program requires the conformance test of this component be performed, even if the specification and program models have been verified and validated, and a certified code generator has been used to produce the executable code. Conformance test is a functional test, i.e. the system under test, named implementation, is seen as a black-box (its internal structure is unknown) with observable inputs/outputs; the overall aim is to check whether this implementation behaves as specified (see Fig. 1). Conformance test of PLCs is advocated by certification bodies and standards [20], [21], which explains the growing interest of companies in several industrial domains for efficient hardware-in-the-loop techniques [22]–[24] to improve the existing practices.

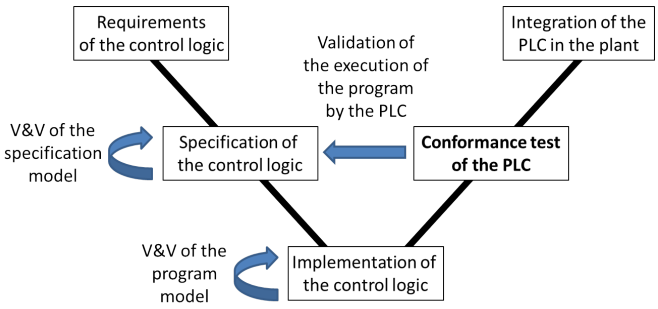


Fig. 1. Place of the work in the life-cycle of PLC software

A promising solution to develop such techniques is to benefit from the results of the researches of the Discrete Event Systems community in the domain of conformance test of formal models. In these works, the specification is a formal model: a Mealy (or finite state) machine [25], a transition system [26], [27], or a timed automaton [28] for instance. The first formalism has been selected for this study because it is well suited to the modeling of logic systems specifications; moreover, functional correctness must be tested before time correctness. As industrial specifications are not expressed in formal languages but in standardized, tailored-made languages, translation rules of industrial specification languages into formal ones are to be developed in order to use these theoretical results for conformance test of PLCs; this issue has been solved in [29] where a method to obtain

Manuscript received February 5 2014; accepted March 28, 2014

J. Provost is with the Assistant Professorship for Safe Embedded Systems, Technische Universität München, DE-85748 Garching bei München, Germany. E-mail: julien.provost@tum.de

J.-M. Roussel and J.-M. Faure are with LURPA, ENS Cachan, FR-94230 Cachan, France. E-mail: jean-marc.roussel@lurpa.ens-cachan.fr, jean-marc.faure@lurpa.ens-cachan.fr

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

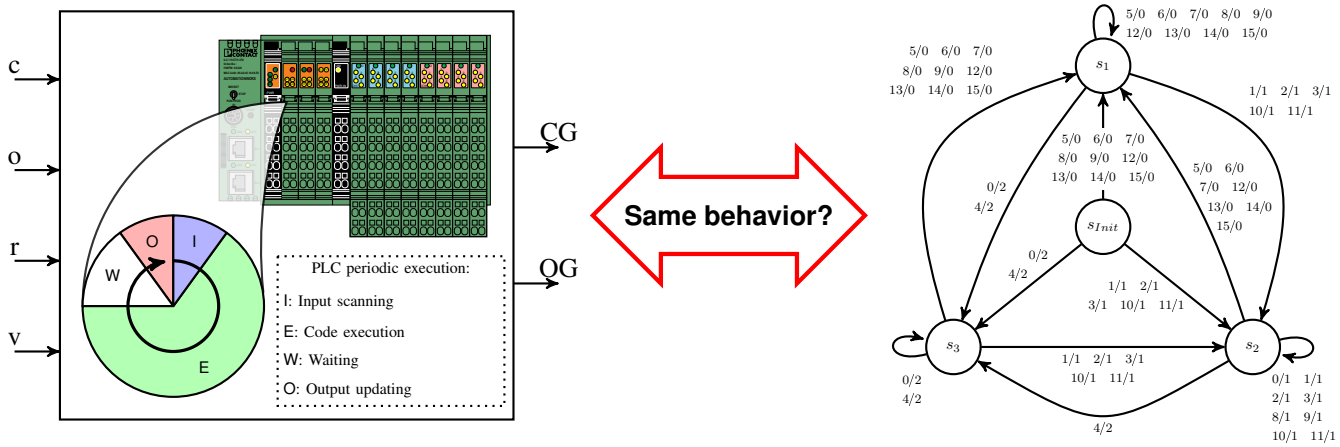


Fig. 2. Aim of the conformance test. Left part: PLC with periodic I/O scanning cycle; right part: Mealy machine describing the specification of the control.

from a Grafcet [30] an equivalent Mealy machine is presented.

Once the formal model of the specification obtained, a *test objective* is to be defined. It is possible, for instance, to test whether some particular states that correspond to hazardous or recovery states can be reached from the initial state or whether some state changes or transition sequences are possible for specific input combinations. When critical systems are considered, as this is the case in this work, a usual test objective is to cross at least once each edge of the directed graph that represents the structure of the machine; this permits to check every state change from each state of the formal model. A test sequence that meets this test objective is termed *complete*.

Then, the test sequence that will be applied to the PLC during the execution of the test can be constructed from the specification model. A test sequence is an ordered list of couples (input combination, expected output combination), termed *test steps*, where the input and expected output combinations correspond respectively to the left and right elements of the label for the considered transition of the Mealy machine; in other words, a test sequence represents the external view of a PLC that executes a control code in conformance with its specification. To avoid tedious, time-consuming and error-prone tasks, the construction of a complete test sequence must be automated; an usual solution is to select the algorithm presented in [31] that minimizes the length of the sequence. A minimum-length test sequence will indeed minimize the duration of the execution of the test, if the duration of each test step is constant.

However, extensive experimental studies have shown in that the execution of a conformance test with a minimum-length test sequence may lead to erroneous test verdicts because synchronous input changes may be detected as asynchronous by the PLC under test [32], [33]. The aim of this paper is to propose another algorithm to construct test sequences. Rather than looking for a minimum-length solution, the overall idea is to construct a test sequence that does not contain synchronous changes of two or more inputs from one test step to the immediately following one. Such a sequence is termed a SIC (Single Input Change) sequence because the value of only one input is modified between two consecutive test steps. It must be noted that the expression *SIC* (or *adjacent*) *sequence* has been already introduced in another domain: test of electronic circuits [34]–[40]. However, the valuable results of these works cannot be directly applied to the issue that we address because they were not focusing on the same type of fault: hardware faults were considered while this work focuses on errors in the PLC code. Moreover, a white-box test was possible in those references whereas the structure of the implementation is unknown in this work (black-box approach); the construction of the test sequence cannot be based on the knowledge of this structure but only of the specification.

Nevertheless, it has not been proved that it is always possible to construct a *complete SIC* sequence *starting from the initial state* of the specification model; this will be the first issue addressed in this paper. Once this issue solved, a solution to construct the SIC sequence will be proposed.

The next section reminds the notations used in this work. The concept of SIC-testability, feature of a Mealy machine that represents the possibility to build an initializable, consistent and complete SIC test sequence from this machine is introduced in the third section; a method to verify whether a Mealy machine is fully SIC-testable as well as to determine the SIC-testable part of a non-fully testable machine is also proposed in this section. The fourth section focuses on the generation of this sequence and provides three solutions according to the connectivity properties of the SIC-testable part. Last, conclusions and prospects for further works are given.

II. Background

The aim of the section is to define the notations used in this paper and to remind previous results. The notations and definitions will be illustrated through an example with 4 logic inputs (c, o, r and v), and 2 logic outputs (OG and CG); then, 16 input combinations (c_I) and 4 output combinations (c_O) can be defined. The PLC where this control is implemented is presented in Fig. 2, left part; the control specification in the form of a Mealy machine is presented in Fig. 2, right part.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

A. Notations of the input and output combinations

A PLC owns n logic input variables and m logic output variables; the value of each of them is either *true* or *false*. 2^n input (2^m output) combinations can then be constructed from this set of input (output) variables by assigning a weight to each variable. An input combination c_I will be represented in three different manners in this paper:

- The first representation, noted symbol(c_I), is the more compact one; symbol(c_I) is indeed an integer that belongs to $[0, 2^n - 1]$ and is defined as follows:
  symbol(c_I) = $\sum_{i=0}^{n-1} c_I[n-1-i] \times 2^{(n-1-i)}$, where:
  - $c_I[n-1-i]$ is an integer that belongs to $[0, 1]$ and is equal to 1 if the i^{th} input variable is *true* and 0 otherwise,
  - $2^{(n-1-i)}$ is the weight of this variable[1].

  This representation will be used in the graphical and tabular descriptions of a Mealy machine.
- The second representation, noted minterm(c_I), is a Boolean expression. A minterm is the conjunction of all the n logic input variables in their positive or complemented form. This representation is very efficient for symbolic calculus and will be used to check the SIC-testability of a Mealy machine.
- The third representation, noted $\mathbb{1}(c_I)$, is that of the set of the only variables which are *true* for the given combination and is well appropriate when defining the SIC relation between two combinations.

The same rules apply for the output combinations c_O. Table I gives the correspondence between these representations for the example introduced in Fig. 2.

B. Formal definition of a Mealy machine

Conformance test of Mealy machines is a mature technique that previously yielded numerous sound theoretical results; good syntheses on this topic are available in [25], [41]. This explains why this formalism was selected to represent formally the specification model.

However, a Mealy machine is theoretically defined on two event alphabets: the input and output alphabets. Since the inputs and outputs of a PLC are logic variables and not events, these two alphabets are to be built prior to defining the Mealy machine that represents the specification of the controller. This will be performed by considering each PLC input (respectively output) combination as an input (resp. output) event.

Let us note I and O the non-empty sets of PLC inputs and outputs (I and O contain logic variables). If the cardinality of I (resp. O) is $|I|$ (resp. $|O|$), there exist $2^{|I|}$ (resp. $2^{|O|}$) distinct input (resp. output) combinations c_I (resp. c_O). Let us note C_I the set of the input combinations and C_O the set of the output combinations.

Using this definition of input and output combinations, the specification of a PLC that executes a control code can be represented by a Mealy machine $(S, s_{Init}, C_I, C_O, \delta, \lambda)$, where:

- S is a non-empty set of states.
- s_{Init} is the initial state, $s_{Init} \in S$.
- C_I is the input alphabet, $|C_I| = 2^{|I|}$.
- C_O is the output alphabet, $|C_O| = 2^{|O|}$.
- δ is the transition function, defined as follows:

$$\begin{array}{rll} \delta: & S \times C_I & \rightarrow S \\ & (s_s, c_I) & \mapsto s_t = \delta(s_s, c_I) \end{array} \quad (1)$$

- λ is the output function, defined as follows:

$$\begin{array}{rll} \lambda: & S \times C_I & \rightarrow C_O \\ & (s_s, c_I) & \mapsto c_o = \lambda(s_s, c_I) \end{array} \quad (2)$$

The specification of the controller is compulsorily deterministic: there is only one initial state and δ and λ are two functions. Moreover, to avoid misinterpretation errors during the test, the Mealy machine is:

- completely defined: δ and λ are total functions[2];

$$\forall (s, c_I) \in S \times C_I, \begin{cases} \exists! \delta(s, c_I) \in S \\ \exists! \lambda(s, c_I) \in C_O \end{cases} \quad (3)$$

- limited to its reachable part;

$$\forall s \in S, \ \exists [c_I^0, \cdots, c_I^n] \in C_I^* \mid \begin{cases} s^1 = \delta(s_{Init}, c_I^0) \\ \forall k \geqslant 1, \ s^{k+1} = \delta(s^k, c_I^k) \\ s = s^n \end{cases} \quad (4)$$

- without transient evolution, i.e. no inputs change introduces successive changes of states or emitted outputs.

$$\forall (s, c_I) \in S \times C_I, \begin{cases} \delta(\delta(s, c_I), c_I) = \delta(s, c_I) \\ \lambda(\delta(s, c_I), c_I) = \lambda(s, c_I) \end{cases} \quad (5)$$

C. Formal definition of a test sequence

A test sequence is an ordered list of couples (input combination, expected output combination) which represents the external view of the expected behavior of a PLC that executes a correct control code. Formally, a test sequence is defined as follows:

$$\left[(c_I^0, c_O^0), (c_I^1, c_O^1), ..., (c_I^n, c_O^n)\right] \in (C_I \times C_O)^* \quad (6)$$

However, the input combinations c_I^k and the expected output combinations c_O^k are not independent. The expected output combination c_O is that associated to the transition which goes from a source state s_s to a target state s_t for the input combination c_I. Hence, an elementary conformance test step et is defined by the following 4-tuple:

$$\begin{array}{l} et = (s_s, c_I, s_t, c_O) \in S \times C_I \times S \times C_O \\ \text{where} \begin{cases} s_t = \delta(s_s, c_I) \\ c_O = \lambda(s_s, c_I) \end{cases} \end{array} \quad (7)$$

A test sequence TS is an ordered list of elementary test steps et which must be:

P1: initializable, i.e. the source state of the first test step is the initial state of the PLC's behavior model, and the input combination c_I^0 is such that the target state is stable:

$$\begin{cases} s^0 = s_{Init} \\ \delta(\delta(s^0, c_I^0), c_I^0) = \delta(s^0, c_I^0) \\ \lambda(\delta(s^0, c_I^0), c_I^0) = \lambda(s^0, c_I^0) \end{cases} \quad (8)$$

[1]The weights are assigned arbitrarily to the variables.

[2]$\exists!$: There exists exactly one.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

TABLE I
EQUIVALENCE BETWEEN THE DIFFERENT REPRESENTATIONS OF THE INPUT AND OUTPUT COMBINATIONS

(logic inputs, weight)																	
	(c,8)	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
	(o,4)	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
	(r,2)	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
	(v,1)	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
symbol(c_I)		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
minterm(c_I)		$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$	$\overline{c}\cdot o\cdot r\cdot v$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$	$c\cdot\overline{o}\cdot r\cdot v$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$	$c\cdot o\cdot\overline{r}\cdot v$	$c\cdot o\cdot r\cdot\overline{v}$	$c\cdot o\cdot r\cdot v$
$\mathbb{1}(c_I)$		$\{\}$	$\{v\}$	$\{r\}$	$\{r,v\}$	$\{o\}$	$\{o,v\}$	$\{o,r\}$	$\{o,r,v\}$	$\{c\}$	$\{c,v\}$	$\{c,r\}$	$\{c,r,v\}$	$\{c,o\}$	$\{c,o,v\}$	$\{c,o,r\}$	$\{c,o,r,v\}$

(logic outputs, weight)					
	(CG,2)	0	0	1	1
	(OG,1)	0	1	0	1
symbol(c_O)		0	1	2	3
minterm(c_O)		$\overline{CG}\cdot\overline{OG}$	$\overline{CG}\cdot OG$	$CG\cdot\overline{OG}$	$CG\cdot OG$
$\mathbb{1}(c_O)$		$\{\}$	$\{OG\}$	$\{CG\}$	$\{CG,OG\}$

P2: consistent, i.e. the source state of the $(k+1)^{th}$ elementary test step is equal to the target state of the k^{th} elementary test step.

$$TS = \big[(s^0, c_I^0, \delta(s^0, c_I^0), \lambda(s^0, c_I^0)), \cdots, (s^n, c_I^n, \delta(s^n, c_I^n), \lambda(s^n, c_I^n))\big] \mid \forall k \geqslant 0, s^{k+1} = \delta(s^k, c_I^k) \quad (9)$$

Moreover, if the test objective is to cross at least once each transition of the Mealy machine (usual objective when the control of critical systems is considered), the test sequence must be:

P3: complete, i.e. there is at least one test step for each element of the transition function:

$$\forall (s, c_I) \in (S \backslash s_{Init} \times I),\ (s, c_I, \delta(s, c_I), \lambda(s, c_I)) \in TS \quad (10)$$

III. SIC-TESTABILITY

SIC-testability is a feature of a Mealy machine that represents the possibility to build an initializable, consistent and complete SIC test sequence from this machine. This concept is illustrated in Fig. 3. The example *3a)* is non-SIC-testable because the test step that corresponds to the self-loop on the state s_2 with the input combination $\overline{a}\cdot\overline{b}$ cannot be preceded by a test step with an input combination where only one of the variables a and b is true; both possible preceding test steps correspond to the input combination $a\cdot b$. This non-SIC-testable transition may lead to a biased verdict: if the input change from $a\cdot b$ to $\overline{a}\cdot\overline{b}$ when the machine is in the state s_2 is erroneously interpreted by the PLC, the target state could be either s_2 (as if it was correctly interpreted) or s_1, thus, potentially rejecting a correct implementation. A similar reasoning is possible for the pinpointed transition of *3b)*; the test step that corresponds to the transition from the state s_2 to the state s_1 with the input combination $\overline{a}\cdot\overline{b}$ cannot be preceded by a test step with an input combination where only one of the variables a and b is true; the only possible preceding test step corresponds to the input combination $a\cdot b$. This non-SIC-testable transition may lead to a non-valid verdict: whatever the interpretation (correct or erroneous) of the input change from $a\cdot b$ to $\overline{a}\cdot\overline{b}$, the target state will be s_1. Thus, it cannot be ensured that this specific transition of the implementation has been tested, and an incorrect implementation may be accepted. On the opposite, the example *3c)* is SIC-testable; it is possible to find for any transition a preceding transition whose input combination differs from only one input.

This section proposes first a formal definition of a SIC test sequence, then presents a method to determine the SIC-testable part of a Mealy machine, part of this machine from which such a sequence can be built. If this part contains all test steps that can be defined from the machine, the machine is said fully SIC-testable, else a coverage rate of the test steps can be defined.

A. Definition of a SIC test sequence

A SIC test sequence is an initializable and consistent (relations (8) and (9) satisfied) test sequence that is based on a SIC input sequence. To express formally this latter property, the SIC relation between two input combinations must be first defined. This definition relies on the representation of an input combination by the subset of I that contains the only variables which are *true* for this combination. Thus, two input combinations c_I and c'_I satisfy a SIC relation if and only if[3]:

$$card((\mathbb{1}(c_I)\backslash\mathbb{1}(c'_I)) \cup (\mathbb{1}(c'_I)\backslash\mathbb{1}(c_I))) = 1 \quad (11)$$

For example, the input combinations which are represented by the minterms $\overline{c}\cdot\overline{o}\cdot r\cdot v$ and $c\cdot\overline{o}\cdot r\cdot v$ satisfy a SIC relation since $card((\{r,v\}\backslash\{c,r,v\}) \cup (\{c,r,v\}\backslash\{r,v\})) = card(\varnothing \cup \{c\}) = 1$

In the remainder of this paper, this symmetrical relation is noted[4]: $c_I\ R_{Gray}\ c'_I$. It must be noted that n SIC relations can be stated for each input combination c_I of a PLC with n logic inputs.

Hence, a test sequence TS is a SIC test sequence if and only if it satisfies the following property:

P5: it is based on a SIC input sequence, i.e.:

$$\forall k > 0,\ c_I^{k+1}\ R_{Gray}\ c_I^k \quad (12)$$

[3] $card(A)$ is the cardinality of set A.
$\mathbb{1}(c_I)\backslash\mathbb{1}(c'_I)$ is the subset of I composed with the elements of $\mathbb{1}(c_I)$ which are not in $\mathbb{1}(c'_I)$.

[4] The subscript Gray has been introduced because two combinations that satisfy this relation may be seen as two adjacent combinations of a Gray code.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

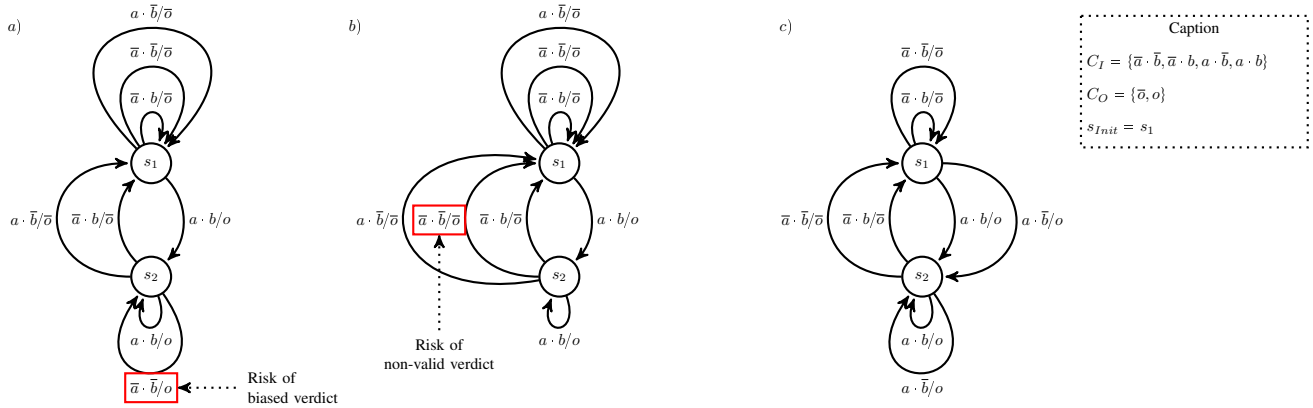


Fig. 3. Examples of non-SIC-testable Mealy machines: a) and b), and a SIC-testable Mealy machine: c)

B. Computation of the SIC-testable part of a Mealy machine

The SIC-testable part of a Mealy machine may be obtained by computing a set R_{SIC} of couples (s_s, c_I), where s_s is the source state of a transition of the Mealy machine and c_I an input combination. Each element of R_{SIC} defines also an elementary test step (s_s, c_I, s_t, c_O) because the target state s_t and the output combination c_O are then completely known from the structure of the machine. The set R_{SIC} is computed iteratively by a fixed point calculation; the set at the i^{th} iteration of this calculation will be noted $R_{SIC}(i)$.

As the SIC sequence must be initializable, the initial set $R_{SIC}(0)$ contains the couples (s_{Init}, c_I) where s_{Init} is the initial state, and c_I is an input combination such that, if s_t is the target state of the transition $(s_{Init}, c_I, s_t, c_O)$, $\delta(s_t, c_I) = s_t$, i.e. there exists a self-loop on s_t for this input combination.

$$R_{SIC}(0) = \left\{(s_{Init}, c_I^0) \mid c_I^0 \in C_I, \delta(\delta(s_{Init}, c_I^0)) = \delta(s_{Init}, c_I^0)\right\} \quad (13)$$

In practice, every logic input of a PLC which is connected to a test bench can be set or reset before the initialization of the PLC. Hence, the state s_t is a state that can become and stay active when the PLC is initialized after a given input combination has been defined.

The following sets $R_{SIC}(k+1)$ are determined by using the two following construction rules:

- If an elementary step (s_t, c_I, s_t, c_O) belongs to a SIC test sequence, it is always possible to add to this sequence an elementary step $(s_t, c_I', \delta(s_t, c_I'), \lambda(s_t, c_I'))$ where c_I' satisfies: $c_I'\ R_{Gray}\ c_I$.
- If an elementary test step (s_s, c_I, s_t, c_O) belongs to a SIC test sequence, the elementary test step (s_t, c_I, s_t, c_O) can be added to this sequence.

These rules can be formally expressed by the following statement:

$$R_{SIC}(k+1) = R_{SIC}(k) \cup \left\{ \left(s_k, c_I^{k+1}\right) \cup \left(\delta\left(s_k, c_I^{k+1}\right), c_I^{k+1}\right) \mid \exists (s_k, c_I^k) \in R_{SIC}(k) \mid \left\{ \begin{array}{l} \delta(s_k, c_I^k) = s_k \\ c_I^{k+1}\ R_{Gray}\ c_I^k \end{array} \right\} \right. \quad (14)$$

The computation stops when $R_{SIC}(k+1) = R_{SIC}(k)$. The Mealy machine is then SIC-testable if $R_{SIC}(k+1)$ contains as many couples as there are potential test steps. Otherwise, the final set $R_{SIC}(k+1)$, denoted R_{SIC}^{Maxi}, defines the SIC-testable part. A SIC coverage rate, defined as follows, permits to quantify the SIC-testability:

$$\text{SIC coverage rate} = \frac{|R_{SIC}^{Maxi}|}{|S \backslash s_{Init}| \times |C_I|} \quad (15)$$

This rate can be seen as a metrics of the ability of the specification to be used to build a complete SIC test sequence. Improving the coverage rate requires the specification be modified, which is not always possible for cost and time reasons.

C. Illustration on the example

Table II presents the results of this calculation for the example presented in Fig. 2. Each cell of the table contains the value of the couple $(\delta(s, c_I), \lambda(s, c_I))$. A circled couple means that the same state is both source and target of the transition (self-loop structure: $\delta(s, c_I) = s$). The behavior represented in this table is deterministic and completely defined since every cell contains one and only one state name. This behavior does not contain any transient evolution since the value of each cell is either a circled value or leads to a cell with a circled value ($\delta(s, c_I) = s$ or $\delta(\delta(s, c_I), c_I) = \delta(s, c_I)$). The number k of the iteration during which the couple (s_s, c_I) was added to $R_{SIC}(k)$ is at the top-left corner of each cell. For example, the couple associated to the cell $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ is obtained at iteration 0 (initialization), because s_3 is reachable from s_{Init} $(\delta(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}) = s_3)$, and this couple corresponds to a self-loop on s_3 $(\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}) = s_3)$. The couple $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$ is obtained in the first iteration, as $\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v\ R_{Gray}\ \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$, and so on. The fixed-point calculation stops at the third iteration, excluding the initialization. The final set contains only 40 couples; the couples that do not belong to this set are represented by colored cells. Hence, the Mealy machine of Fig. 2 is not fully SIC-testable. Its SIC-testable part R_{SIC}^{Maxi} is represented by the cells that are not colored. Its SIC coverage rate is equal to 5/6.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

TABLE II
ILLUSTRATION OF THE FIXED POINT CALCULATION. IN THIS TABLE, THE OUTPUT COMBINATIONS ARE OMITTED FOR CLARITY REASONS.

source state s_s \ minterm(c_I)	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v$	$\overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}$	$\overline{c}\cdot\overline{o}\cdot r\cdot v$	$\overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}$	$\overline{c}\cdot o\cdot\overline{r}\cdot v$	$\overline{c}\cdot o\cdot r\cdot\overline{v}$	$\overline{c}\cdot o\cdot r\cdot v$	$c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}$	$c\cdot\overline{o}\cdot\overline{r}\cdot v$	$c\cdot\overline{o}\cdot r\cdot\overline{v}$	$c\cdot\overline{o}\cdot r\cdot v$	$c\cdot o\cdot\overline{r}\cdot\overline{v}$	$c\cdot o\cdot\overline{r}\cdot v$	$c\cdot o\cdot r\cdot\overline{v}$	$c\cdot o\cdot r\cdot v$
s_{Init}	s_3	s_2	s_2	s_2	s_3	s_1	s_1	s_1	s_1	s_1	s_2	s_2	s_1	s_1	s_1	s_1
s_1	$^1s_3$	$^1s_2$	$^1s_2$	$^1s_2$	$^1s_3$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^1s_2$	$^1s_2$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$	$^0(s_1)$
s_2	$^1(s_2)$	$^0(s_2)$	$^0(s_2)$	$^0(s_2)$	$^2s_3$	$^1s_1$	$^1s_1$	$^1s_1$	$^1(s_2)$	$^1(s_2)$	$^0(s_2)$	$^0(s_2)$	$^2s_1$	$^2s_1$	$^1s_1$	$^1s_1$
s_3	$^0(s_3)$	$^1s_2$	$^1s_2$	s_2	$^0(s_3)$	$^1s_1$	$^1s_1$	s_1	$^1s_1$	s_1	s_2	s_2	$^1s_1$	s_1	s_1	s_1

D. Symbolic computation of the SIC-testable part

As already mentioned, the tabular representation of a Mealy machine is appropriate to explain the principle of computations on small-sized models but is not suitable to perform these computations on non-trivial models. This explains why a symbolic representation of a set of input combinations has been introduced to avoid explicit enumeration of this set during the fixed point calculation.

A set C of combinations c can be represented by a Boolean expression $Exp(C)$ defined as the disjunction of the minterms contained in C:

$$Exp(C) = \mathop{+}_{c\in C} \text{minterm}(c) \tag{16}$$

During the fixed-point calculation, a set of input combinations C can be extended by adding all input combinations c'_I satisfying a SIC relation with at least one of the input combinations c_I in C. The extended set C' is defined as follows:

$$C' = C \cup \{c'_I \mid \exists c_I \in C :\ c'_I\ R_{Gray}\ c_I\}, \tag{17}$$

Using symbolic notation, the Boolean expression of the extended set C' is defined as follows:

$$Exp(C') = \mathop{+}_{i\in I} \left(Exp(C)_{|i\leftarrow false} + Exp(C)_{|i\leftarrow true}\right) \tag{18}$$

The example below illustrates this operation.

$$\begin{aligned}
C &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v\,,\ \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}\,,\ \overline{c}\cdot\overline{o}\cdot r\cdot v\}\\
Exp(C) &= \overline{c}\cdot\overline{o}\cdot r + \overline{c}\cdot\overline{o}\cdot v\\
Exp(C') &= (\overline{o}\cdot r + \overline{o}\cdot v) + (\overline{c}\cdot r + \overline{c}\cdot v) + (\overline{c}\cdot\overline{o})\\
&\quad + (\overline{c}\cdot\overline{o})\\
&= \overline{c}\cdot\overline{o} + \overline{c}\cdot r + \overline{c}\cdot v + \overline{o}\cdot r + \overline{o}\cdot v\\
C' &= \{\overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v}\ ,\ \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v}\ ,\ c\cdot\overline{o}\cdot r\cdot\overline{v}\ ,\\
&\quad \overline{c}\cdot o\cdot r\cdot\overline{v}\ ,\ \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v\ ,\ c\cdot\overline{o}\cdot\overline{r}\cdot v\ ,\\
&\quad \overline{c}\cdot o\cdot\overline{r}\cdot v\ ,\ \overline{c}\cdot\overline{o}\cdot r\cdot v\ ,\ c\cdot\overline{o}\cdot r\cdot v\ ,\\
&\quad \overline{c}\cdot o\cdot r\cdot v\}
\end{aligned} \tag{19}$$

This symbolic representation of a set of input combinations speeds up the computation defined in the equation (14).

IV. SIC TEST SEQUENCE GENERATION

Once R_{SIC}^{Maxi} determined, a SIC test sequence can be constructed by using a graphical representation of this set in the form of a graph where:

- each node represents a couple (s, c_I) that is included in R_{SIC}^{Maxi};
- n arcs start from a node that corresponds to a couple (s, c_I) such that $\delta(s, c_I) = s$. The target nodes of these arcs represent the couples (s, c'_I) such that c'_I satisfies $c'_I\ R_{Gray}\ c_I$. These arcs correspond to input changes between two test steps; the cost associated to these arcs is then equal to 1;
- only one arc starts from a node that corresponds to a couple (s, c_I) such that $\delta(s, c_I) \neq s$. The target node of this arc is the node that represents the couple $(\delta(s, c_I), c_I)$. This arc corresponds to the expected evolution from a source state to a target state during the execution of one test step; the cost associated to this arc is then equal to 0.

Fig. 4 represents a part of this graph. Each node corresponds to a couple (s_s, c_I) where s_s corresponds to its line and c_I corresponds to its row. Since the couples $(s_3, \overline{c}\cdot\overline{o}\cdot r\cdot v)$ and $(s_3, \overline{c}\cdot o\cdot r\cdot v)$ are not in R_{SIC}^{Maxi} there is no node associated to these couples. In this figure, only the arcs related to the couple $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ are represented. Since this couple can be tested in the same experimental test step than $(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$, there is one arc from $(s_{Init}, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ leading to $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$. From this couple, there are four outgoing arcs leading to couples $(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$, $(s_3, \overline{c}\cdot\overline{o}\cdot r\cdot\overline{v})$, $(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v})$ and $(s_3, c\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$. Then, since the couple $\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v) = s_2$, there is one outgoing arc from $\delta(s_3, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$ to $\delta(s_2, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot v)$. On the opposite, since $\delta(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v}) = s_3$, there are four outgoing arcs from $(s_3, \overline{c}\cdot o\cdot\overline{r}\cdot\overline{v})$.

A SIC test sequence can be then constructed by looking for a path that traverses at least once each node of this graph. To reduce the duration of test execution, a minimum-length SIC test sequence can be searched; the optimization problem to solve in this case is a particular solution of a well-known problem in graph theory: the Travelling Salesman Problem [42] – or pre-Hamiltonian path –. The general formulation of this problem is the following: Find a minimum-length closed path that traverses at least once each **node** of the graph. By

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

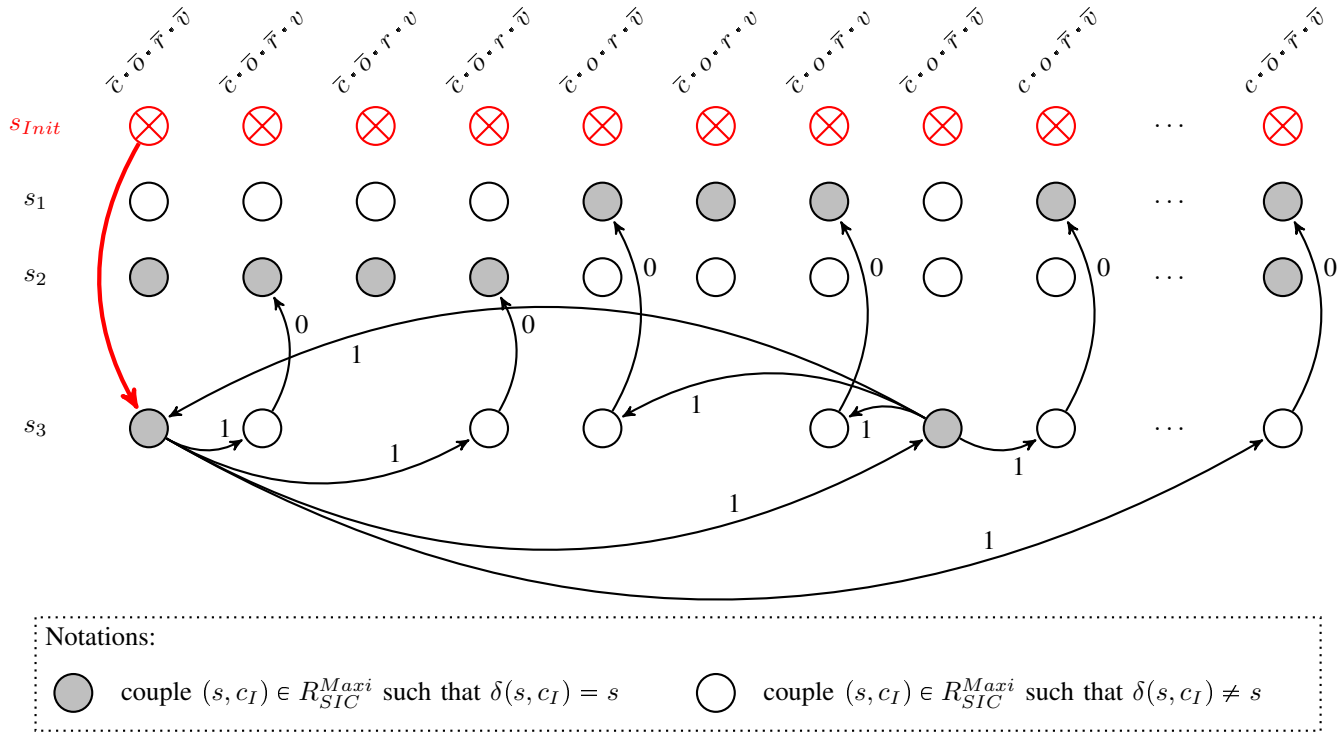


Fig. 4. Part of the graph used to generate a SIC test sequence

TABLE III
SIC TEST SEQUENCE FOR THE SIC-TESTABLE PART OF THE EXAMPLE

s_s:	s_1	s_1	s_3	s_2	s_2	s_3	s_3	s_2	s_1	s_1	s_2	s_1	s_1	s_2	s_2	s_1	s_1	s_2	s_1	s_2	s_2	s_1	s_2	s_2	s_1	s_3	s_1	s_3	s_1	s_2	s_1	s_3	s_1	s_3	s_3
c:	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	1
o:	0	0	0	0	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	0	1	0	0	1	1	1	1	1	0	1	1	1	1	0	0
r:	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	1	1	1	0	0	0	0	0
v:	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
s_t:	s_1	s_3	s_2	s_2	s_3	s_3	s_2	s_1	s_1	s_2	s_1	s_1	s_2	s_2	s_1	s_1	s_2	s_1	s_2	s_2	s_1	s_2	s_2	s_1	s_3	s_1	s_3	s_1	s_2	s_1	s_3	s_1	s_3	s_3	s_1

definition, the length between two nodes is equal to the sum of the costs associated to the collection of arcs that define the shortest path between these two nodes.

However, possible to construct a single SIC test sequence that traverses each node at least once and starts from the initial state with any input combination if and only if the graph that represents R_{SIC}^{Maxi} is strongly connected[5]. Then, a strategy to construct SIC test sequences whatever the connectivity of the graph has been set up (see Fig. 5). If the graph is only connected (and not strongly connected), a single SIC test sequence can be constructed but this sequence must start by an elementary test step that contains a particular input combination. When the graph is not connected, several SIC test sequences shall be constructed; during test execution, the PLC shall be initialized between two of these sequences because each of them starts from the initial state by definition (relation (8)).

For the example presented in Fig. 2 it is possible to generate a single SIC-test sequence that covers its SIC-testable part. This sequence is given in Table III where the top and bottom lines have been added to relate this sequence to Fig. 2 and Table II. This test sequence contains 35 test steps and permits to test the 40 couples (s, c_I) of the SIC-testable part of the specification since some test steps permit to test both couples (s, c_I) and $(\delta(s, c_I), c_I)$, as already mentioned; for example, test step 2 permits to test both $(s_1, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$ and $(s_2, \overline{c}\cdot\overline{o}\cdot\overline{r}\cdot\overline{v})$, and so on for all test steps whose source and target states are different. The test sequence given in Table III is obtained in approximately 4 seconds; this computation lasts longer than that of a minimum-length sequence because the problem to solve is harder.

[5]A graph is strongly connected if and only if it contains a path from n_i to n_j and a path from n_j to n_i for every pair of nodes n_i, n_j

V. CONCLUSION

Even if numerous theoretical results on conformance test of Mealy machines have been published, application to conformance test of PLCs is not completely straightforward because the technological features of these industrial components are not taken into account in the theoretical studies, as pinpointed in [32], [33].

A promising solution to tackle out biased or non-valid test results due to asynchronism between input events that are assumed to be synchronous is to construct a test sequence, termed *SIC test sequence*, where no synchronous input events are present by definition. This paper has presented how the part

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

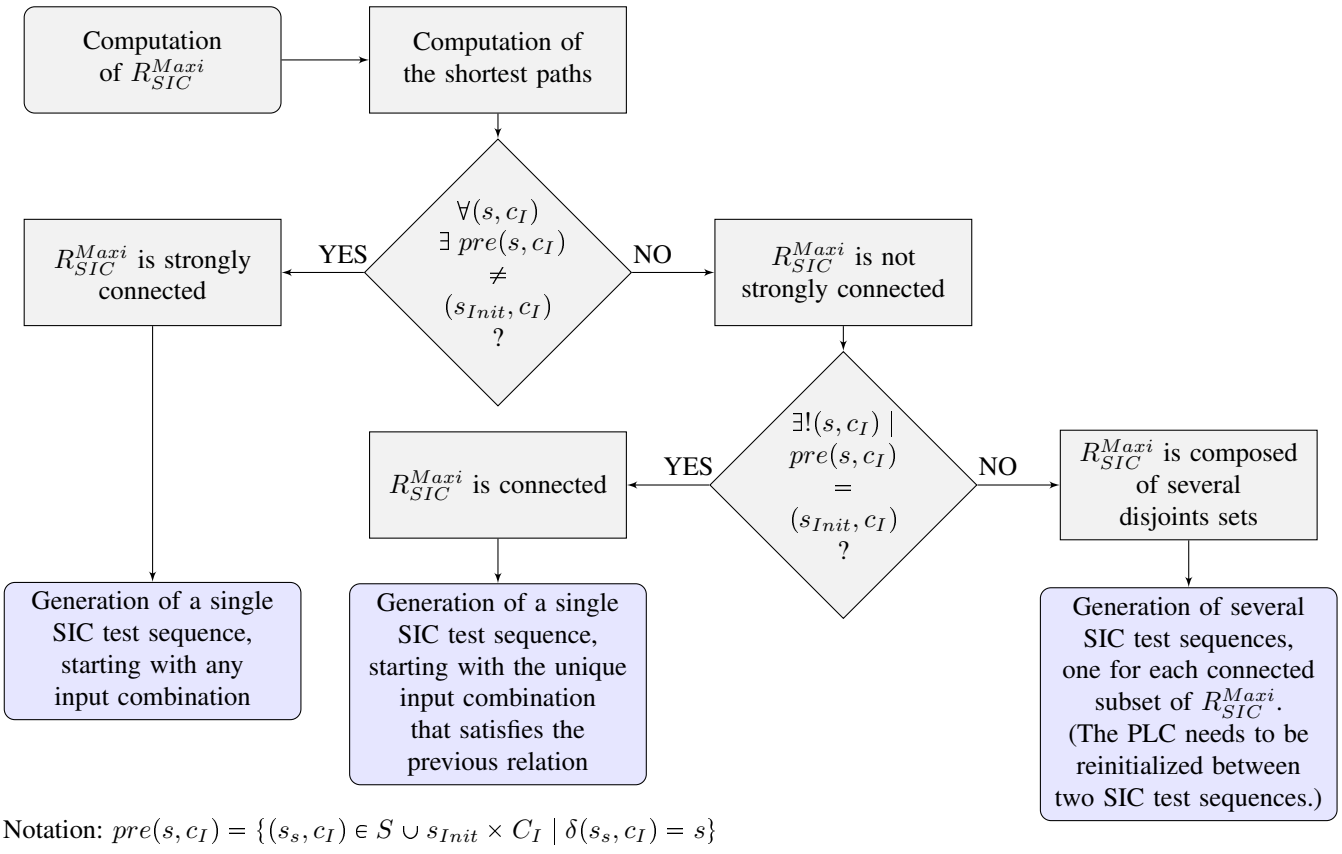


Fig. 5. Flowchart showing the different cases to consider when generating a SIC test sequence

of the Mealy machine that can be tested with such a sequence can be determined and how to construct this sequence.

A coverage rate has been also defined. This rate is however not always equal to 100%. If the objective of the conformance test is to test every transition of the Mealy machine, it will be necessary to use a non-SIC sequence for the transitions which cannot be tested with the SIC sequence. The following stategies shall be then considered:

- test execution for the configurations of the controller that lessen the error rate (periodic I/O scanning and no inputs distribution) as shown in [32], [33];
- multiple execution of the same test sequence and statistical analyses of the results.

Further works are aiming at extending the scope of this study by considering construction of test sequences for timed systems – the formal model that will be used to build this sequence will be a class of timed automata – and analysis approaches based on discrete event systems theory that are complementary to conformance test, like identification or enforcement, to validate the behavior of a PLC.

REFERENCES

[1] V. Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 3, pp. 1234–1249, 2013.

[2] R. Drath, A. Luder, J. Peschke, and L. Hundt, "AutomationML - the glue for seamless automation engineering," in *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*. IEEE, 2008, pp. 616–623.

[3] E. Estévez and M. Marcos, "Model based validation of industrial control systems," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 2, pp. 302–310, 2012.

[4] M. Witsch and B. Vogel-Heuser, "Towards a formal specification framework for manufacturing execution systems," *Industrial Informatics, IEEE Transactions on*, vol. 8, no. 2, pp. 311–320, 2012.

[5] M. Wehrmeister, C. Pereira, and F. Rammig, "Aspect-oriented model-driven engineering for embedded systems applied to automation systems," *Industrial Informatics, IEEE Transactions on*, vol. 9, no. 4, pp. 2373–2386, 2013.

[6] G. Frey and L. Litz, "Formal methods in PLC programming," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4. IEEE, 2000, pp. 2431–2436.

[7] S. Patil, V. Vyatkin, and M. Sorouri, "Formal verification of intelligent mechatronic systems with decentralized control logic," in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[8] M. Perin and J.-M. Faure, "Building meaningful timed models of closed-loop DES for verification purposes," *Control Engineering Practice*, 2012.

[9] S. Preuße, H.-C. Lapp, and H.-M. Hanisch, "Closed-loop system modeling, validation, and verification," in *17th International Conference on Emerging Technologies and Factory Automation (ETFA 2012)*, 2012.

[10] C. Seidner and O. H. Roux, "Formal methods for systems engineering behavior models," *IEEE Transactions on Industrial Informatics*, vol. 4, no. 4, pp. 280–291, 2008.

[11] V. Vyatkin and H.-M. Hanisch, "A modeling approach for verification of IEC 1499 function blocks using net condition/event systems," in *Emerging Technologies and Factory Automation, 1999. Proceedings. ETFA'99. 1999 7th IEEE International Conference on*, vol. 1. IEEE, 1999, pp. 261–270.

[12] X. Weng and L. Litz, "Verification of logic control design using SIPN and model checking: methods and case study," in *American Control Conference, 2000. Proceedings of the 2000*, vol. 6. IEEE, 2000, pp. 4072–4076.

[13] S. Klein, G. Frey, J.-J. Lesage, and L. Litz, "Supporting the changeability

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2014.2315972, IEEE Transactions on Industrial Informatics

of SIPN-based logic control algorithms by verification and validation," in *Proc. of IMACS-IEEE int. conf. on Computational Engineering in Systems Applications*, 2003.

[14] G. Canet, S. Couffin, J.-J. Lesage, A. Petit, and P. Schnoebelen, "Towards the automatic verification of PLC programs written in Instruction List," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, vol. 4. IEEE, 2000, pp. 2449–2454.

[15] S. Lampérière-Couffin and J.-J. Lesage, "Formal verification of the sequential part of PLC programs," in *Workshop on Discrete Event Systems (WODES'00)*. Springer, 2000, pp. 247–254.

[16] V. Gourcuff, O. de Smet, and J.-M. Faure, "Efficient representation for formal verification of PLC programs," in *Proceedings of 8th International Workshop On Discrete Event Systems (WODES'06)*, Ann Arbor USA, 07 2006, pp. pp. 182–187.

[17] G. Čengić and K. Åkesson, "On formal analysis of IEC 61499 applications, part a: Modeling," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 2, pp. 136–144, 2010.

[18] ——, "On formal analysis of IEC 61499 applications, part b: Execution semantics," *Industrial Informatics, IEEE Transactions on*, vol. 6, no. 2, pp. 145–154, 2010.

[19] D. Soliman, K. Thramboulidis, and G. Frey, "Transformation of function block diagrams to UPPAAL timed automata for the verification of safety applications," *Annual Reviews in Control*, 2012.

[20] IEC 60880, *Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions*, 2nd ed. International Electrotechnical Commission, 2006.

[21] IEC 61850-10, *Communications Networks and Systems in Substations - Part 10: Conformance testing*, 2nd ed. International Electrotechnical Commission, 2005.

[22] D. Maclay, "Simulation gets into the loop," *IEE Review*, vol. 43, no. 3, pp. 109–112, 1997.

[23] F. Gu, W. S. Harrison, D. M. Tilbury, and C. Yuan, "Hardware-in-the-loop for manufacturing automation control: Current status and identified needs," in *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*. IEEE, 2007, pp. 1105–1110.

[24] N. Schetinin, N. Moriz, B. Kumar, A. Maier, S. Faltinski, and O. Niggemann, "Why do verification approaches in automation rarely use HIL-test?" in *Industrial Technology (ICIT), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1428–1433.

[25] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines - a survey," in *Proceedings of the IEEE*, vol. 84, no. 8, 1996, pp. 1090–1123.

[26] J. Tretmans, "Model based testing with labelled transition systems," in *Formal Methods and Testing*, ser. Lecture Notes in Computer Science, R. M. Hierons, J. P. Bowen, and M. Harman, Eds. Springer, 2008, vol. 4949, pp. 1–38.

[27] S. Pickin, C. Jard, T. Jéron, J.-M. Jézéquel, and Y. Le Traon, "Test synthesis from UML models of distributed software," *Software Engineering, IEEE Transactions on*, vol. 33, no. 4, pp. 252–269, 2007.

[28] M. Krichen and S. Tripakis, "Conformance testing for real-time systems," *Formal Methods in System Design*, vol. 34, no. 3, pp. 238–304, 2009.

[29] J. Provost, J.-M. Roussel, and J.-M. Faure, "Translating Grafcet specifications into Mealy machines for conformance test purposes," *Control Engineering Practice*, vol. 19, no. 9, pp. 947–957, 2011.

[30] IEC 60848, *GRAFCET specification language for sequential function charts*, 2nd ed. International Electrotechnical Commission, 2002.

[31] S. Naito and M. Tsunoyama, "Fault detection for sequential machines by transitions tours," in *Proceedings of the IEEE Fault Tolerant Computer Symposium*, 1981, pp. 238–243.

[32] J. Provost, J.-M. Roussel, and J.-M. Faure, "Testing programmable logic controllers from finite state machines specification," in *Proceedings of the 3rd Workshop on Dependable Control of Discrete Systems (DCDS'11)*, 2011.

[33] ——, "Conformance test of programmable logic controllers — execution of minimum-length test sequences," École Normale Supérieure de Cachan, France, Tech. Rep., 2014. [Online]. Available: http://www.lurpa.ens-cachan.fr/-244795.kjsp

[34] M. A. Gharaybeh, M. L. Bushnell, and V. D. Agrawal, "Classification and test generation for path-delay faults using single struck-at fault tests," *Journal of Electronic Testing*, vol. 11, no. 1, pp. 55–67, 1997.

[35] R. David, P. Girard, C. Landrault, S. Pravossoudovitch, and A. Virazel, "Hardware generation of random single input change test sequences," *Journal of Electronic Testing*, vol. 18, no. 2, pp. 145–157, 2002.

[36] A. Virazel, R. David, P. Girard, C. Landrault, and S. Pravossoudovitch, "Delay fault testing: Choosing between random SIC and random MIC test sequences," *Journal of Electronic Testing: Theory and Applications*, vol. 17, no. 3-4, pp. 233–241, 2001.

[37] I. Voyiatzis, T. Haniotakis, and C. Halatsis, "Algorithm for the generation of SIC pairs and its implementation in a BIST environment," *IEE Proceedings-Circuits, Devices and Systems*, vol. 153, no. 5, pp. 427–432, 2006.

[38] W. Yi, F. Xing-hua, and W. Dai-qiang, "An implementation of random single input change technique for low-power test," in *Proceeding of the 2nd International Conference on Anti-counterfeiting, Security and Identification*, 2008, pp. 352–355.

[39] B. Ye, T. Li, Q. Zhao, D. Zhou, X. Wang, and M. Luo, "A low power test pattern generation for built-in self-test based circuits," *International Journal of Electronics*, vol. 98, no. 3, pp. 301–309, 2011.

[40] F. Liang, L. Zhang, S. Lei, G. Zhang, K. Gao, and B. Liang, "Test patterns of multiple SIC vectors: theory and application in BIST schemes," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 21, no. 4, 2013.

[41] M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, Eds., *Model-Based Testing of Reactive Systems, Advanced Lectures*. Springer, 2005, vol. 3472 of Lecture Notes in Computer Science.

[42] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

Julien Provost Julien Provost received the Ph.D. degree from École Normale Supérieure de Cachan, France, in 2011. Then, he joined Chalmers University of Technology, Sweden, as a Post-Doctoral Researcher for two years. Currently, he is Assistant Professor at Technische Universität München, Germany where he holds the Assistant Professorship for Safe Embedded Systems. His research interest focuses on the development of formal methods for specification, verification and validation of safe-critical distributed Discrete Event Systems (DES).

Jean-Marc Roussel Jean-Marc Roussel received the Ph.D. degree in 1994. He is currently Associate Professor of Automatic Control at École Normale Supérieure de Cachan (France). His research fields are modeling, synthesis and analysis of control systems with formal methods.

Jean-Marc Faure Jean-Marc Faure received the Ph.D. degree from École Centrale de Paris in 1991. He is currently Professor of Automatic Control and Automation Engineering at the Institut Superieur de Mecanique de Paris and researcher at École Normale Superieure de Cachan, France. His research fields are modeling, synthesis and analysis of Discrete Event Systems (DES) with special focus on formal verification and conformance test methods to improve dependability of critical systems.

J.-M. Faure is member of the IEEE and Associate Editor of the Journal T-ASE since 2012. He is chair of the steering committee of the IFAC workshop series "Dependable Control of Discrete Systems" and has served in many committees of IFAC and IEEE conferences.

Reliability Engineering and System Safety 96 (2011) 267–277

Contents lists available at ScienceDirect

Reliability Engineering and System Safety

journal homepage: www.elsevier.com/locate/ress

Review

Algebraic determination of the structure function of Dynamic Fault Trees

G. Merle *, J.-M. Roussel, J.-J. Lesage

LURPA - ENS Cachan, 61 avenue du Président Wilson, Cachan 94230, France

ARTICLE INFO

Article history:
Received 3 April 2010
Received in revised form
2 October 2010
Accepted 4 October 2010
Available online 20 October 2010

Keywords:
Dynamic Fault Tree
Algebraic modeling
Structure function

ABSTRACT

This paper presents an algebraic framework allowing to algebraically model dynamic gates and determine the structure function of any Dynamic Fault Tree (DFT). This structure function can then be exploited to perform both the qualitative and quantitative analysis of DFTs directly, even though this latter aspect is not detailed in this paper. We illustrate our approach on a DFT example from the literature.

Contents

* Corresponding author. Tel.: +33 6 23 79 54 51; fax: +33 1 47 40 22 20.
E-mail addresses: guillaume.merle@lurpa.ens-cachan.fr (G. Merle), jean-marc.roussel@lurpa.ens-cachan.fr (J.-M. Roussel), jean-jacques.lesage@lurpa.ens-cachan.fr (J.-J. Lesage).

0951-8320/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ress.2010.10.001

1. Introduction

The structure function of a Static Fault Tree (SFT) – a fault tree (FT) which only contains gates OR, AND, and K-out-of-N – is a Boolean function which represents the failure of the top event (*TE*) according to the failure of the basic events (*BEs*) of the FT. This algebraic model is classically used to perform both the qualitative and quantitative analysis of SFTs directly. For complex systems, these analyses are most often performed thanks to BDD-based methods [9,19] or other combinatorial techniques [1,17].

The introduction of dynamic gates – gates PAND, FDEP, and Spare – in FTs has changed the nature of the relation between the *TE* and the *BEs*. In a Dynamic Fault Tree (DFT), the failure of the *TE* depends not only on the failure of the *BEs* but also on the order of occurrence of these failures. As this last aspect is not taken into account in the Boolean model of failures (which only expresses whether a *BE* has occurred or not), a classical Boolean function cannot represent the dynamic relations between the *TE* and the *BEs* that exist in a DFT.

In a previous article, we presented the basics of an algebraic framework allowing to algebraically model dynamic gates PAND and FDEP, and determine the structure function of any Dynamic Fault Tree (DFT) containing these gates [13]. In this paper, we extend our previous work to Spare gates in order to be able to determine the structure function of any DFT. This structure function is based on a specific algebraic model of failures which allows to take into account the order of occurrence of failures. As this algebraic model is an extension of the Boolean model used for SFTs, all the results previously obtained for SFTs are preserved.

This paper is organised as follows. The most common approaches used to perform the analysis of DFTs are presented in Section 2. The algebraic framework that we introduce to model DFTs is detailed in Section 3, and the algebraic model of dynamic gates which can be determined from it is presented in Section 4. This algebraic model allows to determine the canonical form of the structure function of any DFT, as shown in Section 5, and our approach is illustrated on a DFT example in Section 6. Finally, we show how the qualitative analysis of DFTs can be performed directly from the canonical form of the structure function in Section 7.

2. State of the art

Several approaches have been used to avoid the problem of the determination of the structure function of DFTs. These approaches can be either modular or global.

Global approaches consist in solving the whole DFT directly, whereas modular approaches consist in:

- dividing the DFT into independent static and dynamic subtrees (or modules) prior to analysis: if a substree contains static gates only, it is considered as static; if a subtree contains at least one dynamic gate, it is considered as dynamic;
- solving the modules separately; and
- combining the results of the various modules to get the overall result for the entire tree.

Various methods exist to analyze the static and dynamic modules of DFTs. On the one hand, solving static modules can be done by using Binary Decision Diagrams (BDDs), other combinatorial techniques, or even some DFT Analysis models such as Markov Chains. On the other hand, solving dynamic modules is generally done using Input/Output Interactive Markov Chains [4–6], Stochastic Petri Nets (SPN) [1,7], or Temporal Bayesian Networks [2,3,15]. On the one hand, Markov Chains provide the cut sequences of the (sub)tree, which are the failure sequences that lead to the states of the Markov Chain in which the *TE* of the (sub)tree fails. They also provide the failure probability of the *TE* of the (sub)tree by solving the set of differential equations which is equivalent to the Markov Chain. On the other hand, the reachability graph of SPNs provides the cut sequences of the (sub)tree, and the failure probability of the *TE* can be computed after converting the SPN into its corresponding Markov Chain. However, in both cases, the failure of the components of the system is most often modeled by exponential time-to-failure distributions. Temporal Bayesian Networks allow to address this limit by allowing to consider other distributions. However, Bayesian Networks only allow to perform the quantitative analysis of the dynamic (sub)tree, and the inference algorithms used limit the distributions considered to Gaussian distributions [10] and mixtures of truncated exponentials [16].

An analytic approach was introduced in [23] to analyze DFTs by modelling the dynamic gate PAND and by determining simplification theorems. The authors focus on three temporal gates: gates PAND, Simultaneous-AND (SAND), and Priority-OR (POR). Gate SAND was created to address the ambiguity encountered in the definition of gate PAND regarding the simultaneity of input events, whereas gate POR was created from the definition of the Exclusive-OR gate found in [22]. Each event of the FT is assigned a sequence value which allows to know the order in which events occur. The authors propose an extension of truth tables, denoted as Temporal Truth Tables and based on these sequence values, to prove theorems allowing to simplify the FTs which contain the three temporal gates considered. Nevertheless, this approach allows to perform the qualitative analysis of FTs, only, and the only dynamic gate considered is gate PAND.

We have not found in the literature any attempt to provide an algebraic model for all dynamic gates allowing to determine the structure function of a DFT explicitly as it is currently the case for SFTs. The goal of the algebraic determination of the structure function of DFTs is to be able to perform their analysis directly whatever the distribution considered for basic events. We present such an algebraic framework in Section 3.

3. Algebraic framework for the modeling of Dynamic Fault Trees

3.1. Temporal model of non-repairable events

The structure function of SFTs is based on a Boolean model of events, and of basic events in particular. With this simple model, the only aspect which is taken into account is the presence or absence of failure. However, this Boolean model cannot render the order of occurrence of events which is necessary for the modeling of dynamic gates. To count on the temporal aspect of events, we consider the top event, the intermediate events, and the basic events as *temporal functions*, which are piecewise right-continuous

on $\mathbb{R}^+ \cup \{+\infty\}$, whose range is $\mathbb{B} = \{0,1\}$. In accordance with [22], we consider all events as non-repairable, each of them being perfectly defined by its unique date of occurrence—noted $d(a)$ for an event a. A generic timing diagram of such an event a is given in Fig. 1. In this paper, we denote by $\mathcal{E}_{nr}$ the set of these temporal functions, which corresponds to the set of non-repairable events.

With this temporal model, non-repairable events can be ordered according to their date of occurrence. This characteristics is the cornerstone of our approach, and it is used to model each of the operators which are needed to model both the static and dynamic gates of FTs.

However, specific attention must be paid to the simultaneity of events when modeling the order of occurrence of events, as it is explained below.

3.2. Simultaneity in DFTs

In a FT, simultaneity among events may arise in two ways. Independent basic events can occur simultaneously if they have a discrete probability distribution with a non-null probability mass exactly at the same time. Because the failure probability distributions are usually considered as continuous functions with infinite support, the simultaneous occurrence has null probability, and can be neglected. A second case of simultaneity may arise at any level of a FT when there are repeated basic events. FTs with repeated events represent the most powerful combinatorial model in dependability [11], and require ad hoc analysis techniques. Nevertheless, the presence of repeated events across modules of dynamic gates has not yet been explored in its full generality. In [24], repeated events are allowed, but the paper does not provide any algorithm to derive the list of the cut sequences.

Let us consider the DFT in Fig. 2, in which event A is a repeated basic event.

If basic events A, B, C, and D occur according to sequences $[B, C, A]$, $[C, B, A]$, or $[D, A]$, intermediate events G and H occur simultaneously at the same time as A occurs. This example shows that intermediate nodes of a FT can occur simultaneously because of the presence of repeated basic events. The simultaneity problem has been briefly addressed in [4], and has been solved by resorting to the concept of "non-determinism", a concept that is not easy to accept in engineering practice because many engineers believe that the behavior of technical systems, and in particular control systems, must necessarily be deterministic. We assert that a choice must be made regarding the semantics of simultaneous events, and dynamic gates. For instance, in the case of simultaneous events in input to a PAND gate, two choices are possible (Fig. 2):

- if the order relation is considered strictly, when intermediate events G and H occur simultaneously, TE 1 does not occur, and gate PAND would then be considered as being "non-inclusive"; and
- if the order relation is not considered strictly, when intermediate events G and H occur simultaneously, TE 1 occurs at the same time as G or H, and gate PAND would then be considered as being "inclusive".

Both interpretations of the order relation can be taken into account, and algebraically modeled.

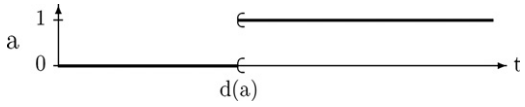


Fig. 1. A non-repairable event.

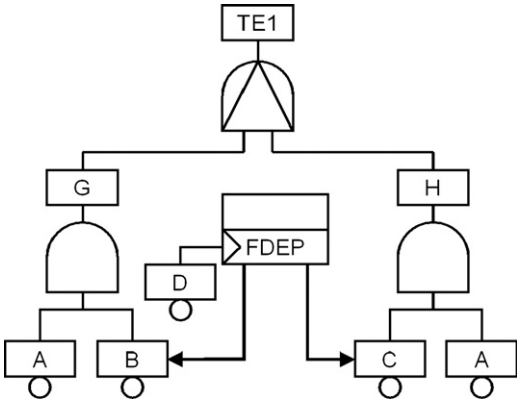


Fig. 2. An example of DFT with one repeated basic event.

3.3. Boolean operators

Any elements of $\mathcal{E}_{nr}$ can be composed thanks to a rewriting of classical Boolean operators. The temporal definition of Boolean operators OR and AND, based on the dates of occurrence of a and b (which are denoted by $d(a)$ and $d(b)$, respectively), is

$$d(a+b) = \begin{cases} d(a) & \text{if } d(a) < d(b), \\ d(a) & \text{if } d(a) = d(b), \\ d(b) & \text{if } d(a) > d(b), \end{cases} \quad d(a \cdot b) = \begin{cases} d(b) & \text{if } d(a) < d(b) \\ d(a) & \text{if } d(a) = d(b) \\ d(a) & \text{if } d(a) > d(b) \end{cases}$$

Indeed, $a+b$ occurs as soon as a or b occurs, and $a \cdot b$ occurs as soon as a and b have occurred. It can be noted that operators OR and AND are commutative.

The identity elements of operators OR and AND in $\mathcal{E}_{nr}$ are denoted by $\bot$, and $\top$, respectively, to which these dates can be assigned:

$$d(\bot) = +\infty, \quad d(\top) = 0$$

$\bot$ is the never-occurring event whereas $\top$ is the always-occurring event.

3.4. Temporal operators

To model the order of occurrence of events, we introduce an operator non-inclusive BEFORE (BF, with symbol $\lhd$), and an operator SIMULTANEOUS (SM, with symbol $\triangle$), whose formal definitions, based on the dates of occurrence of a and b, are

$$d(a \lhd b) = \begin{cases} d(a) & \text{if } d(a) < d(b), \\ +\infty & \text{if } d(a) = d(b), \\ +\infty & \text{if } d(a) > d(b), \end{cases} \quad d(a \triangle b) = \begin{cases} +\infty & \text{if } d(a) < d(b) \\ d(a) & \text{if } d(a) = d(b) \\ +\infty & \text{if } d(a) > d(b) \end{cases}$$

The result of the composition of two events a and b by operators BF and SM is illustrated by the timing diagrams in Figs. 3 and 4, respectively, in three cases: Case 1: $d(a) < d(b)$, Case 2: $d(a) = d(b)$, Case 3: $d(a) > d(b)$.

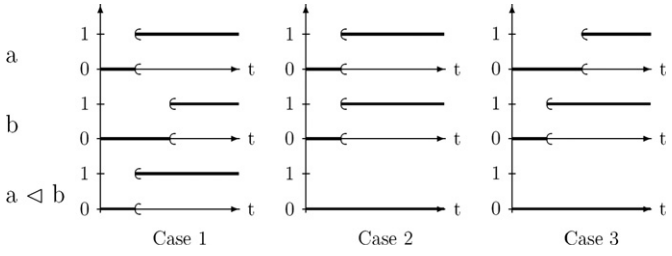


Fig. 3. Timing diagrams of operator non-inclusive BEFORE (BF).

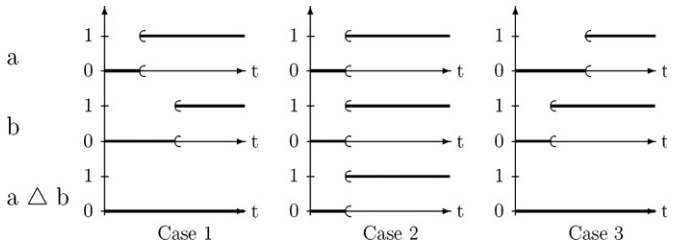


Fig. 4. Timing diagrams of operator SIMULTANEOUS (SM).

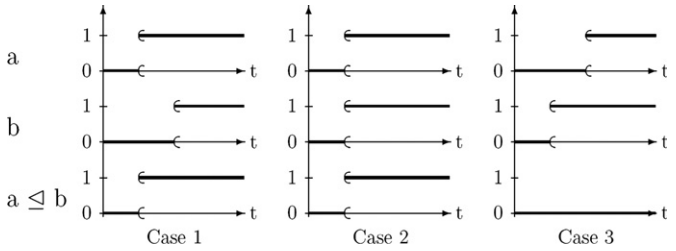


Fig. 5. Timing diagrams of operator INCLUSIVE BEFORE (IBF).

Based on the previous two operators, we can introduce a non-strict or INCLUSIVE BEFORE (IBF, with symbol $\trianglelefteq$) operator

$$a \trianglelefteq b = a \triangleleft b + a \triangle b \tag{1}$$

whose definition, based on the dates of occurrence of a and b, is

$$d(a \trianglelefteq b) = \begin{cases} d(a) & \text{if } d(a) < d(b) \\ d(a) & \text{if } d(a) = d(b) \\ +\infty & \text{if } d(a) > d(b) \end{cases}$$

The result of the composition of two events a and b by operator IBF is illustrated by the timing diagrams in Fig. 5 in three cases: Case 1: $d(a) < d(b)$, Case 2: $d(a) = d(b)$, Case 3: $d(a) > d(b)$.

According to these timing diagrams, and to (1), $a \trianglelefteq b$ occurs in two cases: when a occurs strictly before b, Case 1 (which corresponds to $a \triangleleft b$); and when a occurs at the same time as b, Case 2 (which corresponds to $a \triangle b$).

3.5. Algebraic structure of $\mathcal{E}_{nr}$

Static Fault Tree Analysis is mainly based on the structure function of SFTs, which is determined and simplified thanks to a substructure of the Boolean algebra of Boolean values: the Abelian dioid $\{\{0,1\},+,\cdot,0,1\}$.[1]

We have demonstrated in [14] that by providing a new temporal definition of events (Section 3.1) and by rewriting the Boolean operators + and · (Section 3.3), the framework obtained still has an Abelian dioid structure. The properties that are commonly used for the simplification of SFTs can hence still be applied with our model, and their structure functions can be determined as usual. In particular, operators OR and AND satisfy the four additional following theorems, which are theorems which hold on Boolean algebras:

$$a + (b \cdot c) = (a+b) \cdot (a+c) \tag{2}$$

$$a + \top = \top \tag{3}$$

$$a + (a \cdot b) = a \tag{4}$$

$$a \cdot (a+b) = a \tag{5}$$

$(\mathcal{E}_{nr}, +, \cdot)$ thus has an algebraic structure which allows to express gates OR, AND, and K-out-of-N, and to determine the structure function of SFTs as it is commonly done by using the classical Boolean algebra of Boolean variables. Temporal operators non-inclusive BEFORE and Inclusive BEFORE satisfy several theorems which are useful for calculation, some of which are presented in Appendix A. The proofs of these theorems can be found in [14]. Furthermore, the temporal operators introduced in this section allow to determine an algebraic model of dynamic gates, as shown in Section 4.

4. Algebraic model of dynamic gates

Based on the algebraic framework introduced in Section 3, the algebraic model of dynamic gates is now going to be developed. The temporal operators introduced in Section 3.4 allow to take into account and algebraically model both a strict and a non-strict order relation. However, a non-strict inclusive interpretation of dynamic gates seems more coherent with the designers' expectations. For this reason, in the remainder of this paper, we define an algebraic model of dynamic gates by means of operator IBF ($\trianglelefteq$), only, even though it is easy to define an algebraic model of dynamic gates by means of operator BF ($\triangleleft$) as well.

Furthermore, in accordance with [13], we assume that basic events are statistically independent, and have a continuous failure time distribution, so that they cannot occur simultaneously. Hence, for any two basic events a and b with the above characteristics, the following relation holds:

$$a \triangle b = \bot \tag{6}$$

The algebraic models of gates PAND and FDEP are presented in Sections 4.1 and 4.2, respectively. The algebraic model of Spare gates is presented in an increasing order of complexity: Spare gates with 2 and 3 input events are studied in Sections 4.3 and 4.4, respectively. Besides, we consider that there is only one type of Spare gate which is the Warm Spare gate and that Cold and Hot Spare gates [20] are particular cases of Warm Spare gates. Both of them are studied in Section 4.6.

4.1. Algebraic model of gate PAND

According to [20], gate PAND is defined in Fig. 6.

An algebraic model of gate PAND can hence be determined as

$$Q = (A \cdot B) \cdot (A \trianglelefteq B)$$

This model can be simplified, thanks to the theorems of Appendix A, as follows[2]:

$$Q = (A \cdot B) \cdot (A \trianglelefteq B)$$

$$Q \overset{(41)}{=} B \cdot (A \trianglelefteq B)$$

4.2. Algebraic model of gate FDEP

According to [4,8,20], the FDEP gate – Functional Dependency gate – is a dynamic gate comprised of a trigger input event – either a basic event or the output of another gate of the tree – and a set of dependent basic events. Fig. 7 provides a pictorial depiction of an FDEP gate with two dependent basic events A and B, T representing the trigger event. When the trigger event occurs, the dependent basic events are forced to occur.

In Fig. 7, basic events A and B can fail by themselves, or can be forced to fail by the trigger event T. In accordance with [3], we

[1] $\{\{0,1\},+,\cdot,0,1\}$ is an Abelian dioid because + and · are commutative, idempotent, and, respectively, allow 0 and 1 as their identity element, · is distributive over +, and · allows 0 as an absorbing element.

[2] In the equation below, the notation $\overset{(41)}{=}$ indicates that the expression $B \cdot (A \trianglelefteq B)$ is obtained from the expression $(A \cdot B) \cdot (A \trianglelefteq B)$ by applying theorem (41) from the Appendix. This notation will be used in the remainder of this paper.

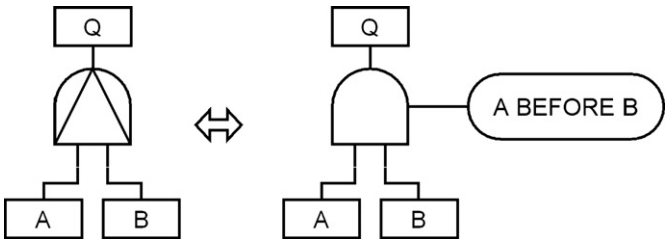


Fig. 6. Definition of gate PAND from [20].

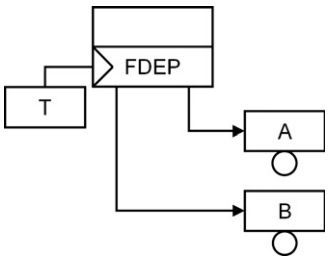


Fig. 7. An FDEP gate with two dependent basic events *A* and *B*.

choose to denote the global failure of basic events *A*, and *B* by the substituted variables A_T, and B_T to explicitly indicate the effect of trigger *T*: basic event *A* fails (A_T) if it is forced to fail by the trigger event (*T*) or if it fails by itself before the trigger event fails ($A \trianglelefteq T$). The algebraic model of gate FDEP thus is

$$\begin{cases} A_T &= T + (A \trianglelefteq T) \\ B_T &= T + (B \trianglelefteq T) \end{cases}$$

This model can be simplified, thanks to the theorems of Appendix A, as follows:

$$\begin{cases} A_T &= T + (A \trianglelefteq T) \overset{(40)}{=} A + T \\ B_T &= T + (B \trianglelefteq T) \overset{(40)}{=} B + T \end{cases}$$

Thanks to this simplification, it appears that the behavior of dynamic gate FDEP is equivalent to the behavior of static gate OR, as it has been suggested by some authors [20].

4.3. Algebraic model of Spare gates with two input events

In this section, we completely detail the algebraic model of Spare gates with two input events in the main configurations which may be encountered in DFTs. The different cases are treated in an increasing order of complexity, from a single Spare gate in Section 4.3.1 to two Spare gates sharing a spare event in Section 4.3.2, and even to the generalization to *n* Spare gates sharing a spare event in Section 4.3.3.

4.3.1. Algebraic model of a single Spare gate

Let us consider a Spare gate with two input events – the primary event *A* and one spare event *B* – as shown in Fig. 8.

As stated in [20], the output *Q* of the gate occurs when the primary and all spares have failed, so when *A* and *B* have failed, in this case. *A* and *B* are basic events and cannot fail simultaneously – $A \triangle B = \perp$ – so *Q* will occur if *A* and *B* fail according to sequences [*A*,*B*] or [*B*,*A*]. It is important to note that in sequence [*A*,*B*], *B* fails while in its active mode (denoted as B_a), whereas in sequence [*B*,*A*], *B* fails while in its dormant mode (denoted as B_d). It is essential to distinguish both failure modes by using two different variables, for quantitative analysis purposes. Indeed, *B* does not have the same failure distribution when it fails during its dormant mode ($B \equiv B_d$) or during its active mode ($B \equiv B_a$). As we aim at making possible the quantitative analysis of DFTs from their structure function, this structure function must hence provide sufficient information to know whether spare events are in their dormant or active mode.

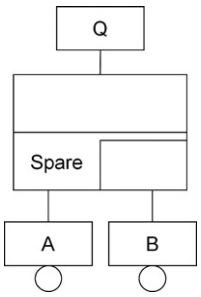


Fig. 8. A single Spare gate with one primary event *A* and one spare event *B*.

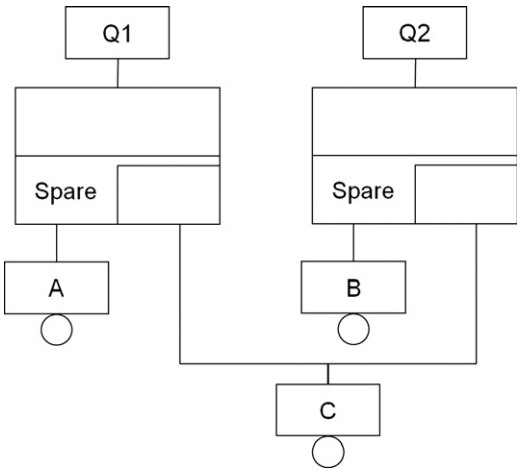


Fig. 9. Two Spare gates sharing a spare event *C*.

The algebraic model of gate Spare can hence be expressed as

$$Q = B_a \cdot (A \triangleleft B_a) + A \cdot (B_d \triangleleft A)$$

Furthermore, as *B* cannot be both in an active state and in a dormant state, we have

$$B_d \cdot B_a = \perp$$

4.3.2. Algebraic model of two Spare gates sharing a spare event

Let us now consider two Spare gates with two input events – with primary events *A* and *B* – sharing a spare event *C*, as shown in Fig. 9.

If we focus on the Spare gate on the left side, *Q* 1 will occur as soon as *A* and *C* have failed – as stated in Section 4.3.1 – or if *A* fails and *C* is made unavailable because *B* has failed before *A*. As a consequence, the algebraic model of this Spare gate is

$$\begin{cases} Q1 = C_a \cdot (A \triangleleft C_a) + A \cdot (C_d \triangleleft A) + A \cdot (B \triangleleft A) \\ C_d \cdot C_a = \perp \end{cases}$$

The algebraic expression for the Spare gate on the right side can be determined in the same way by symmetry. Consequently, the final algebraic model of any of two Spare gates sharing a spare event is

$$\begin{cases} Q1 = C_a \cdot (A \triangleleft C_a) + A \cdot (C_d \triangleleft A) + A \cdot (B \triangleleft A) \\ Q2 = C_a \cdot (B \triangleleft C_a) + B \cdot (C_d \triangleleft B) + B \cdot (A \triangleleft B) \\ C_d \cdot C_a = \perp \end{cases}$$

4.3.3. Algebraic model of n Spare gates sharing a spare event

Let us consider *n* Spare gates with 1 output event Q_i and two input events: a primary event P_i – $i \in \{1, \cdots, n\}$ and a spare event *S*.

If we focus on the first Spare gate, Q_1 will occur as soon as P_1 and S have failed – as stated in Section 4.3.1 – or if P_1 fails and S is made unavailable because the primary event of any of the other Spare gates has failed before P_1. As a consequence, the algebraic model of the first Spare gate is

$$\begin{cases} Q_1 = S_a \cdot (P_1 \lhd S_a) + P_1 \cdot (S_d \lhd P_1) + \sum_{i \neq 1} P_1 \cdot (P_i \lhd P_1) \\ S_d \cdot S_a = \perp \end{cases}$$

The algebraic expression for Q_i, $i \in \{1, \ldots, n\}$, can be determined in the same way by symmetry. Consequently, the final algebraic model of any of n Spare gates sharing a spare event is

$$\begin{cases} Q_i = S_a \cdot (P_i \lhd S_a) + P_i \cdot (S_d \lhd P_i) + \sum_{j \neq i} P_i \cdot (P_j \lhd P_i) \\ S_d \cdot S_a = \perp \end{cases}$$

4.4. Algebraic model of a single Spare gate with three input events

Let us consider a Spare gate with three input events – the primary event A and two spare events B and C – as shown in Fig. 10.

As stated in [20], the output Q of the gate occurs when the primary and all spares have failed, so when A, B, and C have failed. A, B, and C are basic events and cannot fail simultaneously so Q will occur if A, B, and C fail according to sequences [A,B,C], [A,C,B], [B,A,C], [B,C,A], [C,A,B], or [C,B,A]. It is important to note that, when the quantitative analysis will be performed from the structure function, B and C will not have the same distribution function in the six sequences. For instance, in sequence [A,B,C], both B and C fail during their active mode (denoted by B_a and C_a), whereas in sequence [B,C,A], both B and C fail during their dormant mode (denoted by B_d and C_d). The algebraic model of gate Spare can hence be expressed as

$$\begin{aligned} Q = {} & C_a \cdot (A \lhd B_a) \cdot (B_a \lhd C_a) + B_a \cdot (A \lhd C_d) \cdot (C_d \lhd B_a) \\ & + C_a \cdot (B_d \lhd A) \cdot (A \lhd C_a) + A \cdot (B_d \lhd C_d) \cdot (C_d \lhd A) \\ & + B_a \cdot (C_d \lhd A) \cdot (A \lhd B_a) + A \cdot (C_d \lhd B_d) \cdot (B_d \lhd A) \end{aligned}$$

As B and C cannot be both in an active state and in a dormant state, we have

$$\begin{cases} B_d \cdot B_a = \perp \\ C_d \cdot C_a = \perp \end{cases}$$

The algebraic model of many Spare gates with many common spare events can be deduced from these models by considering the same approach. It will not be detailed here though.

4.5. General case of n Spare gates with m input events sharing $p \leq (m-1)$ spare events

In the general case, the algebraic model of n Spare gates with m input events sharing $p \leq (m-1)$ spare events is complex. However, a few information can be provided regarding its complexity.

The algebraic model of each Spare gate can be divided into two parts:

- A first part which describes the failure of the gate "by itself", i.e. without taking into account the influence of the other Spare gates. In the case of two Spare gates sharing a spare event described in Section 4.3.2, this first part, respectively, is $C_a \cdot (A \lhd C_a) + A \cdot (C_d \lhd A)$ and $C_a \cdot (B \lhd C_a) + B \cdot (C_d \lhd B)$ for Q 1 and Q 2; it can be noted that this first part does not depend on the main components of the other gates.
- A second part which describes the influence of the other Spare gates on the Spare gate considered. In the case of two Spare gates sharing a spare event described in Section 4.3.2, this second part, respectively, is $A \cdot (B \lhd A)$ and $B \cdot (A \lhd B)$ for Q 1 and Q 2.

To illustrate this complexity, if we consider the case of n Spare gates with three input events sharing two spare events, the algebraic model of each Spare gate contains two parts:

- the first part contains 3! terms
- the second part contains:
  - $6(n-1)+(n-1)!$ terms if there are three Spare gates or more ($n \geq 3$);
  - $6(n-1)$ terms if there are less than three Spare gates ($n < 3$).

If we consider Fig. 11 which shows a benchmark from [25] containing four Spare gates with three input events sharing two spare events ($n = 4$), the algebraic model of each Spare gate includes 30 terms:

- six terms which correspond to the first part, and which represent the 3! possible failure sequences of the inputs of the Spare gate;
- $24 = 6 \times (4-1)+(4-1)!$ terms which correspond to the second part, and hence to the influence of the three other gates. These 24 terms are divided as follows:
  - six terms which correspond to the 6 order-3 sequences in which the spare events *VMS* 1 and *VMS* 2 fail first;
  - six terms which correspond to the 6 order-3 sequences in which the spare events *VMS* 1 and *VMS* 2 fail second;
  - six terms which correspond to the 6 order-3 sequences in which the spare events *VMS* 1 and *VMS* 2 fail last;
  - six terms which correspond to the 3! order-3 sequences in which the failures of the main components only are sufficient to engender the failure of the gate.

4.6. Specific case of Cold and Hot Spare events

The algebraic models presented in Sections 4.3 and 4.4 are models of Spare gates in the general case of Warm Spare events. These algebraic models can be simplified in the specific cases of Cold and Hot Spare events:

- if a spare event S is a Cold Spare event, it cannot fail while in a dormant state, so S_d will never occur and any expression containing S_d in the algebraic models can be removed;
- if a spare event S is a Hot Spare event, it will have the same distribution function when in an active and in a dormant state, so $S_a \equiv S_d \equiv S$ and the algebraic models can be simplified.

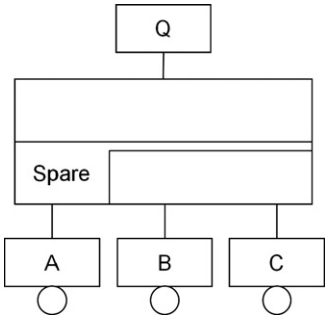


Fig. 10. A single Spare gate with one primary event A and two spare events B and C.

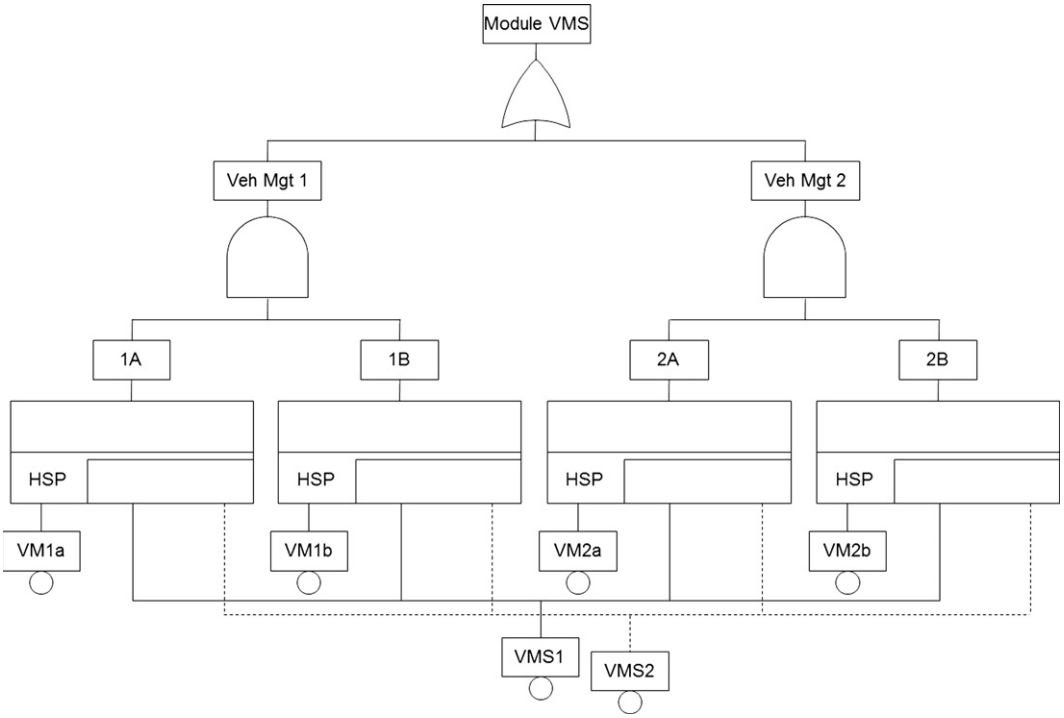


Fig. 11. A benchmark from [25] with four Spare gates with three input events sharing two spare events.

Thanks to the algebraic model of all dynamic gates, we are now going to show how the structure function of any DFT can be determined and simplified to a canonical form.

5. Determination of the canonical form of the structure function of Dynamic Fault Trees

The algebraic models of dynamic gates presented in Section 4 allow to determine the structure function of any DFT. It can be interesting to manipulate this structure function to obtain a sum-of-product canonical form since such a canonical form of the structure function can be much useful to perform both the qualitative and the quantitative analysis of the DFT.

On the one hand, each product term of this canonical form is an algebraic expression which engenders the occurrence of the *TE* of the DFT and which holds for a given number of failure sequences. The cut sequences of each product term, and hence of the whole DFT, can hence be determined from this canonical form. On the other hand, the failure probability of the *TE* can be determined from this canonical form thanks to the standard inclusion – exclusion formula [21].

Given a DFT with n basic events $\{b_i, i \in (1,\ldots,n)\}$, the structure function for the *TE* becomes an expression containing at most the n basic events, and operators $+$, $\cdot$, $\lhd$, $\triangle$, and $\trianglelefteq$. In [13], we showed that the structure function of any DFT with gates PAND and FDEP could be developed and simplified, thanks to the theorems presented in Appendix A, to arrive to a standardized sum-of-product *canonical form* where each product term contains operator $\cdot$, and ordered pairs of variables linked by operator $\lhd$ only, as in

$$TE = \sum \left(\prod b_i \cdot \prod (b_j \lhd b_k) \right), \quad j \notin \{i,k\} \tag{7}$$

However, in this paper, we extend the work presented in [13] to Spare gates, and the algebraic model of Spare gates defined in Sections 4.3–4.6 involves the same temporal operator that is used to model gates PAND and FDEP, so the expression (7) still holds in the case of a DFT with Spare gates.

In this canonical form of the structure function, each product term $\prod b_i \cdot \prod (b_j \lhd b_k)$ is not a single cut sequence, but an algebraic expression providing a sufficient condition on the order of basic event failures that leads to the *TE* which may contain more than one cut sequence, and actually is a *cut sequence set* (*CSS*).

If we suppose that there are n product terms in (7), the canonical form can be rewritten in the compact form

$$TE = \sum_{i=1}^{n} CSS_i \tag{8}$$

Nevertheless, a CSS may be included in one or more CSSs. CSS_i is included in one of the CSS_j if it satisfies the criterion [18]

$$CSS_i \cdot \sum_{j \neq i} CSS_j = CSS_i \tag{9}$$

If CSS_i is included in one of the CSS_j, it is redundant, and can be removed from the structure function (8). Iterative application of the criterion (9) removes all the redundant CSSs, and returns the minimal set $\mathbb{S}_{min}$ of non-redundant CSSs.

If $\mathbb{S}_{min}$ contains $(m \leq n)$ cut sequence sets, the minimal canonical form of the structure function can be expressed as

$$TE = \sum_{i=1}^{m} CSS_i, CSS_i \in \mathbb{S}_{min} \tag{10}$$

This minimal canonical form of the structure function can be exploited to perform the qualitative analysis – as it will be shown through an example in the next section – and the quantitative analysis of any DFT (even though this latter aspect will not be detailed in this paper).

6. Determination of the structure function of a DFT example

We propose to determine the structure function of a DFT example extracted from [2] which is depicted in Fig. 12.

This DFT models the failure of a cardiac assist system (HCAS) which is divided into four modules: Trigger, CPU unit, motor

section, and pumps. The Trigger consists of a crossbar switch (CS) and a system supervisor (SS). The failure of either CS or SS triggers the failure of both CPUs. The CPU unit is a Warm Spare, which has a primary P and a spare unit B having a dormancy of 0.5. For the motor section to function, either MOTOR or MOTORC need to be working. The pumps unit is comprised of two Cold Spares, each having a primary pump (PUMP_1 and PUMP_2), and sharing a common spare pump (Backup_PUMP). In order for the pumps unit to fail, all three pumps need to fail and CSP_1 needs to fail before (or at the same time as) CSP_2, i.e. PAND gate.

The structure function of this DFT could be determined without any further simplification. However, in an educational purpose, we propose to divide this DFT into three subtrees whose structure functions will be successively determined. These three subtrees are as follows:

- subtree 1, which corresponds to the failure of the CPU unit: this subtree contains one OR gate, one FDEP gate, and one Warm Spare gate, and is hence dynamic;
- subtree 2, which corresponds to the failure of the motor section: this subtree contains a single AND gate and is hence static;
- subtree 3, which corresponds to the failure of the pumps unit: this subtree contains one PAND gate and two Cold Spare gates, and is hence dynamic.

If we denote by TE_1, TE_2, and TE_3 the top events of these three subtrees, the structure function of the DFT in Fig. 12 can be expressed as

$$TE = TE_1 + TE_2 + TE_3$$

The structure function of each one of these three subtrees can now be determined thanks to the algebraic model of dynamic gates presented in Section 4.

6.1. Determination of the structure function for TE_2

Subtree 2 is static since it contains a single AND gate. Its structure function can thus be determined directly as

$$TE_2 = MOTOR \cdot MOTORC$$

6.2. Determination of the structure function for TE_3

Subtree 3 is dynamic since it contains gates PAND and Cold Spare. The algebraic model of gate PAND presented in Section 4.1 allows to express TE_3 as

$$TE_3 = CSP2 \cdot (CSP1 \trianglelefteq CSP2)$$

According to the algebraic model of two Spare gates sharing a spare event presented in Section 4.3.2,

$$\begin{cases} CSP1 = BP \cdot (P1 \lhd BP) + P1 \cdot (P2 \lhd P1) \\ CSP2 = BP \cdot (P2 \lhd BP) + P2 \cdot (P1 \lhd P2) \end{cases}$$

where BP denotes the active state BP_a of the spare pump since it cannot fail while in a dormant state, and $P1$ and $P2$ denote $PUMP\_1$ and PUM P_2, respectively, for the sake of clarity.

The following result will be exploited to determine the structure function:

$$\begin{aligned} A \trianglelefteq ((A \cdot B) + C) &\overset{(31)}{=} (A \trianglelefteq (A \cdot B)) \cdot (A \trianglelefteq C) \overset{(32)}{=} ((A \trianglelefteq A) + (A \trianglelefteq B)) \cdot (A \trianglelefteq C) \\ &\overset{(28)}{=} (A + (A \trianglelefteq B)) \cdot (A \trianglelefteq C) \overset{(39)}{=} A \cdot (A \trianglelefteq C) \overset{(41)}{=} A \trianglelefteq C \end{aligned} \tag{11}$$

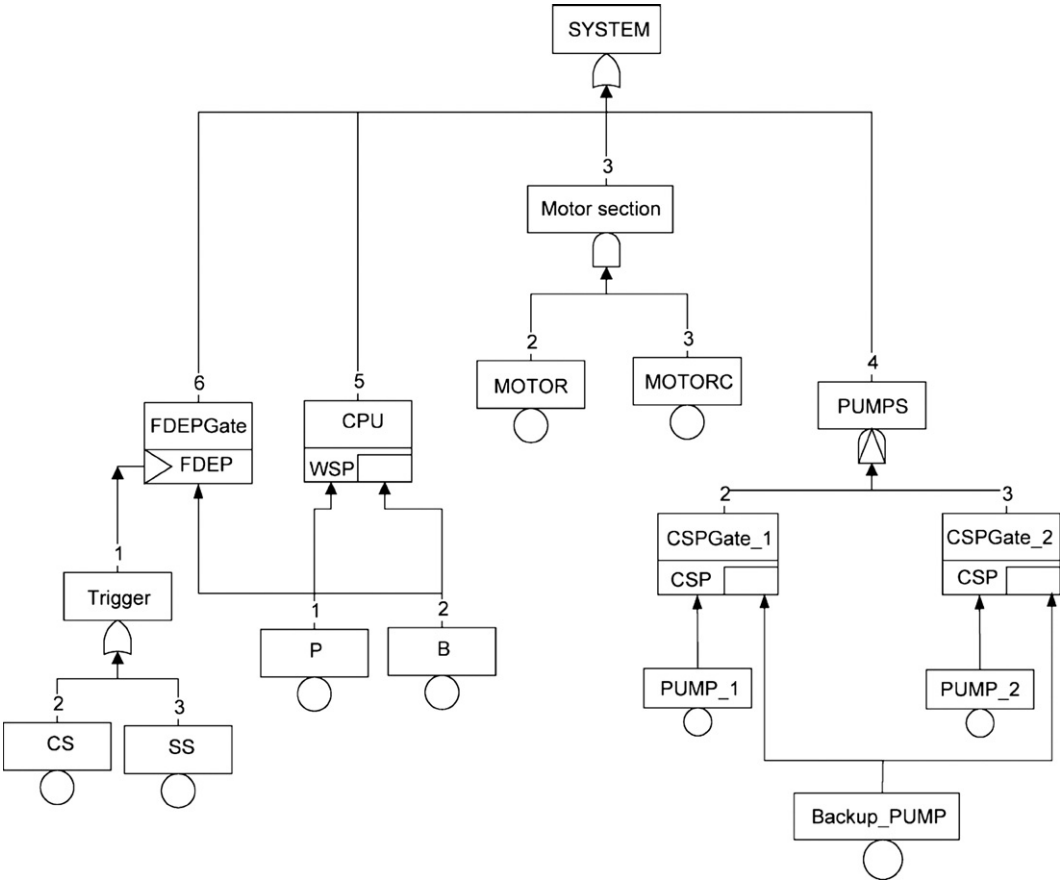


Fig. 12. The HCAS Dynamic Fault Tree from [2].

$CSP1 \trianglelefteq CSP2$ can now be expressed as

$$\begin{aligned}
CSP1 \trianglelefteq CSP2 &= [BP \cdot (P1 \lhd BP) + P1 \cdot (P2 \lhd P1)] \trianglelefteq CSP2 \\
&\overset{(35)}{=} (BP \cdot (P1 \lhd BP)) \trianglelefteq CSP2 + (P1 \cdot (P2 \lhd P1)) \trianglelefteq CSP2 \\
&\overset{(36)}{=} (BP \trianglelefteq CSP2) \cdot ((P1 \lhd BP) \trianglelefteq CSP2) \\
&\quad + (P1 \trianglelefteq CSP2) \cdot ((P2 \lhd P1) \trianglelefteq CSP2) \\
&\overset{(37)}{=} (BP \trianglelefteq CSP2) \cdot (P1 \lhd BP) \cdot (P1 \trianglelefteq CSP2) \\
&\quad + (P1 \trianglelefteq CSP2) \cdot (P2 \lhd P1) \cdot (P2 \trianglelefteq CSP2) \\
&= (P1 \lhd BP) \cdot (BP \trianglelefteq CSP2) \cdot (P1 \trianglelefteq CSP2) \\
&\quad + (P2 \lhd P1) \cdot (P1 \trianglelefteq CSP2) \cdot (P2 \trianglelefteq CSP2) \\
&\overset{(1),(6)}{=} (P1 \trianglelefteq BP) \cdot (BP \trianglelefteq CSP2) \cdot (P1 \trianglelefteq CSP2) \\
&\quad + (P2 \trianglelefteq P1) \cdot (P1 \trianglelefteq CSP2) \cdot (P2 \trianglelefteq CSP2) \\
&\overset{(42)}{=} (P1 \trianglelefteq BP) \cdot (BP \trianglelefteq CSP2) + (P2 \trianglelefteq P1) \cdot (P1 \trianglelefteq CSP2) \\
&\overset{(1),(6)}{=} (P1 \lhd BP) \cdot (BP \trianglelefteq CSP2) + (P2 \lhd P1) \cdot (P1 \trianglelefteq CSP2)
\end{aligned}$$

On the one hand,

$$\begin{aligned}
BP \trianglelefteq CSP2 &= BP \trianglelefteq [BP \cdot (P2 \lhd BP) + P2 \cdot (P1 \lhd P2)] \\
&\overset{(11)}{=} BP \trianglelefteq [P2 \cdot (P1 \lhd P2)] \overset{(32)}{=} (BP \trianglelefteq P2) + (BP \trianglelefteq (P1 \lhd P2)) \\
&\overset{(6),(33)}{=} (BP \trianglelefteq P2) + (BP \lhd P1) + BP \cdot P1 \cdot (P2 \trianglelefteq P1) \\
&\overset{(1),(6)}{=} (BP \lhd P2) + (BP \lhd P1) + BP \cdot P1 \cdot (P2 \lhd P1)
\end{aligned}$$

On the other hand,

$$\begin{aligned}
P1 \trianglelefteq CSP2 &= P1 \trianglelefteq [BP \cdot (P2 \lhd BP) + P2 \cdot (P1 \lhd P2)] \\
&\overset{(25)}{=} P1 \trianglelefteq [BP \cdot (P2 \lhd BP) + P1 \cdot P2 \cdot (P1 \lhd P2)] \\
&\overset{(11)}{=} P1 \trianglelefteq [BP \cdot (P2 \lhd BP)] \overset{(32)}{=} (P1 \trianglelefteq BP) + (P1 \trianglelefteq (P2 \lhd BP)) \\
&\overset{(6),(33)}{=} (P1 \trianglelefteq BP) + (P1 \lhd P2) + P1 \cdot P2 \cdot (BP \trianglelefteq P2) \\
&\overset{(1),(6)}{=} (P1 \lhd BP) + (P1 \lhd P2) + P1 \cdot P2 \cdot (BP \lhd P2)
\end{aligned}$$

Consequently,

$$\begin{aligned}
CSP1 \trianglelefteq CSP2 &= (P1 \lhd BP) \cdot [(BP \lhd P2) + (BP \lhd P1) + BP \\
&\quad \cdot P1 \cdot (P2 \lhd P1)] + (P2 \lhd P1) \cdot [(P1 \lhd BP) \\
&\quad + (P1 \lhd P2) + P1 \cdot P2 \cdot (BP \lhd P2)] \overset{(26)}{=} (P1 \lhd BP) \cdot [(BP \lhd P2) + BP \cdot P1 \\
&\quad \cdot (P2 \lhd P1)] + (P2 \lhd P1) \cdot [(P1 \lhd BP) + P1 \cdot P2 \cdot (BP \lhd P2)] \\
&= (P1 \lhd BP) \cdot (BP \lhd P2) + BP \cdot P1 \cdot (P1 \lhd BP) \cdot (P2 \lhd P1) \\
&\quad + (P2 \lhd P1) \cdot (P1 \lhd BP) + P1 \cdot P2 \cdot (P2 \lhd P1) \cdot (BP \lhd P2) \\
&\overset{(4),(25)}{=} (P1 \lhd BP) \cdot (BP \lhd P2) + (P2 \lhd P1) \\
&\quad \cdot (P1 \lhd BP) + P1 \cdot (BP \lhd P2) \cdot (P2 \lhd P1)
\end{aligned}$$

Since $BP \equiv BP_a$ cannot fail before $P1$ and $P2$ as it can only fail in its active mode,

$$CSP1 \trianglelefteq CSP2 = (P1 \lhd BP) \cdot (BP \lhd P2) + (P2 \lhd P1) \cdot (P1 \lhd BP)$$

Finally,

$$\begin{aligned}
TE_3 &= [BP \cdot (P2 \lhd BP) + P2 \cdot (P1 \lhd P2)] \\
&\quad \cdot [(P1 \lhd BP) \cdot (BP \lhd P2) + (P2 \lhd P1) \cdot (P1 \lhd BP)] \\
&\overset{(26)}{=} P2 \cdot (P1 \lhd P2) \cdot (P1 \lhd BP) \cdot (BP \lhd P2) \\
&\quad + BP \cdot (P2 \lhd BP) \cdot (P2 \lhd P1) \cdot (P1 \lhd BP) \\
&\overset{(27)}{=} P2 \cdot (P1 \lhd BP) \cdot (BP \lhd P2) + BP \cdot (P2 \lhd P1) \cdot (P1 \lhd BP)
\end{aligned}$$

6.3. Determination of the structure function for TE_1

Subtree 1 is dynamic since it contains gates FDEP and Warm Spare. The model of the Spare gate presented in Section 4.3.1 is valid when the input events of the Spare gate are independent basic events which can consequently not occur simultaneously. However, in the case of subtree 1, basic events P and B are basic events which have a common cause failure represented by the Trigger, and they can hence occur simultaneously when the trigger occurs. Nevertheless, this particular aspect can be taken into account in our model by introducing an additional term related to the simultaneous occurrence of P_T and B_T – $P_T \triangle B_T$ – in the algebraic model of the Spare gate. TE_1 can hence first be expressed as

$$TE_1 = B_{a_T} \cdot (P_T \lhd B_{a_T}) + P_T \cdot (B_{d_T} \lhd P_T) + P_T \triangle B_T$$

where B_a and B_d denote the active and dormant state of the spare unit B, according to the algebraic model of the Warm Spare gate presented in Section 4.3.1. As explained in Section 4.2, the substituted variables B_{a_T}, B_{d_T}, B_T, and P_T explicitly indicate the effect of trigger T and denote the global failure of basic events B and P. Thus we have

$$\begin{cases} B_T = B + T \\ B_{a_T} = B_a + T \\ B_{d_T} = B_d + T \\ P_T = P + T \end{cases}$$

The additional term $P_T \triangle B_T$ can first be determined since $T = CS + SS$:

$$\begin{aligned}
P_T \triangle B_T &= (P + CS + SS) \triangle (B + CS + SS) \\
&= ((CS + SS) \lhd B) \cdot ((CS + SS) \lhd P) \\
&= ((CS + SS) \lhd B_d) \cdot ((CS + SS) \lhd P) \\
&\quad + ((CS + SS) \lhd P) \cdot ((CS + SS) \lhd B_a)
\end{aligned}$$

Consequently,

$$\begin{aligned}
TE_1 &= (B_a + CS + SS) \cdot ((P + CS + SS) \lhd (B_a + CS + SS)) \\
&\quad + (P + CS + SS) \cdot ((B_d + CS + SS) \lhd (P + CS + SS)) \\
&\quad + ((CS + SS) \lhd B_d) \cdot ((CS + SS) \lhd P) \\
&\quad + ((CS + SS) \lhd P) \cdot ((CS + SS) \lhd B_a)
\end{aligned}$$

Finally, this structure function can be developed thanks to the use of theorems (15) and (19), and simplified to the following form:

$$\begin{aligned}
TE_1 &= (P \lhd B_a) \cdot (B_a \lhd (CS + SS)) \\
&\quad + (P \lhd (CS + SS)) \cdot ((CS + SS) \lhd B_a) \\
&\quad + (B_d \lhd P) \cdot (P \lhd (CS + SS)) + (B_d \lhd (CS + SS)) \cdot ((CS + SS) \lhd P) \\
&\quad + ((CS + SS) \lhd B_d) \cdot ((CS + SS) \lhd P) + ((CS + SS) \lhd P) \cdot ((CS + SS) \lhd B_a)
\end{aligned}$$

Some of the terms of this structure function can be grouped to obtain the final simplified following form:

$$TE_1 = CS + SS + P \cdot (B_d \lhd P) + B_a \cdot (P \lhd B_a)$$

6.4. Determination of the structure function of the whole DFT in Fig. 12

The structure function of the DFT in Fig. 12 can finally be determined as

$$\begin{aligned}
TE = CS + SS + MOTOR \cdot MOTORC + P \cdot (B_d \lhd P) + B_a \cdot (P \lhd B_a) \\
+ BP \cdot (P2 \lhd P1) \cdot (P1 \lhd BP) + P2 \cdot (P1 \lhd BP) \cdot (BP \lhd P2)
\end{aligned}$$

This structure function can be used to perform the qualitative analysis of the DFT in Fig. 12 directly, as explained in Section 7.

7. Qualitative analysis of DFTs based on the structure function

The canonical form of the structure function of the DFT in Fig. 12, which was determined previously, is a sum-of-product form whose each product term can provide *minimal cut sets* or *minimal cut sequences*. Two cases may happen:

- if a product term does not contain the temporal operator BF ($\lhd$), it is static and provides *minimal cut sets* for the DFT;
- if a product term contains the temporal operator BF ($\lhd$), it is dynamic and provides *minimal cut sequences* for the DFT. In some cases, a set of minimal cut sequences may represent all the possible sequences which correspond to a minimal cut set and can hence be reduced to this minimal cut set.

The structure function of the DFT in Fig. 12 contains seven terms. On the one hand, three terms do not contain the temporal operator BF ($\lhd$). They are static and can hence provide three minimal cut sets for the DFT:

$CS, SS, (MOTOR \cdot MOTORC)$

On the other hand, four terms contain the temporal operator BF ($\lhd$). They are dynamic and can hence provide the minimal cut sequences of the DFT:

$[B_d, P], [P, B_a], [P2, P1, BP], [P1, BP, P2]$

The minimal cut sets and sequences of the DFT in Fig. 12 can then be determined as

$CS, SS, (MOTOR \cdot MOTORC)$

$[B_d, P], [P, B_a], [P2, P1, BP], [P1, BP, P2]$

In this case, it can be noted that the two minimal cut sequences $[B_d, P]$ and $[P, B_a]$ are logically equivalent to the single minimal cut set $P \cdot B$. However, the minimal cut set does not render the two states of the basic event B which will be needed to perform the quantitative analysis of the DFT. This is the reason why these two minimal cut sequences were not reduced to the equivalent minimal cut set $B \cdot P$.

This canonical form of the structure function thus provides a hybrid result for the qualitative analysis of DFTs by allowing to determine both minimal cut sets and minimal cut sequences. Furthermore, as it was shown above, a set of minimal cut sequences may sometimes be equivalent to a single minimal cut set. Two cases may happen:

- if these minimal cut sequences contain spare events, they must not be reduced to their equivalent minimal cut set since the knowledge of the state in which spare events fail will be needed to perform the quantitative analysis of the DFT;
- if these minimal cut sequences do not contain spare events, they can be reduced to their equivalent minimal cut set. Indeed, even though both results are equivalent, minimal cut sets represent a more concise – and hence more useful – result to the practitioner than the corresponding set of minimal cut sequences.

8. Conclusion

In this paper, we presented an algebraic framework allowing to determine the structure function of any DFT, as it is commonly the case for SFTs. Furthermore, we showed that this structure function can be simplified to a canonical form for any DFT. Starting from this canonical form of the structure function, the qualitative analysis of the DFT can be performed directly by determining both the minimal cut sets and sequences of the DFT.

Regarding the quantitative analysis of DFTs, which has not been developed in this paper, it can also be performed from the structure function of DFTs thanks to appropriate probabilistic models of all dynamic gates (PAND, FDEP, and Spare). We have already determined such probabilistic models, which do not depend on the failure distribution considered for basic events, but they could not be presented in this paper.

Even though cut sequences can be extracted quite easily from the structure function, ongoing work is currently addressed to the systematic determination of the *minimal set* of *minimal cut sequences*. Besides, the work presented in this paper allowed to propose a formal background for the determination of the structure function of DFTs, and future work will be dedicated to the elaboration of efficient algorithms allowing to automatically perform the calculation of this structure function and the qualitative analysis of DFTs.

Appendix A. Development and simplification theorems

The temporal operators non-inclusive BEFORE and Inclusive BEFORE introduced in Section 3.4 satisfy the following theorems (their proofs can be found in [12]), for any non-repairable events a, b, and c. These theorems will allow to calculate and simplify the structure function of DFTs.

A.1. Theorems satisfied by operator non-inclusive BEFORE

Operator non-inclusive BEFORE satisfies the following theorems, for all $a,b,c \in \mathcal{E}_{nr}$:

$$a \lhd a = \bot \tag{12}$$

$$\bot \lhd a = \bot \tag{13}$$

$$a \lhd \bot = a \tag{14}$$

$$a \lhd (b + c) = (a \lhd b) \cdot (a \lhd c) \tag{15}$$

$$a \lhd (b \cdot c) = (a \lhd b) + (a \lhd c) \tag{16}$$

$$a \lhd (b \lhd c) = (a \lhd b) + (a \cdot b \cdot ((c \lhd b) + (c \triangle b))) \tag{17}$$

$$a \lhd (b \trianglelefteq c) = (a \lhd b) + (a \cdot b \cdot (c \lhd b)) \tag{18}$$

$$(a + b) \lhd c = (a \lhd c) + (b \lhd c) \tag{19}$$

$$(a \cdot b) \lhd c = (a \lhd c) \cdot (b \lhd c) \tag{20}$$

$$(a \lhd b) \lhd c = (a \lhd b) \cdot (a \lhd c) \tag{21}$$

$$(a \trianglelefteq b) \lhd c = (a \trianglelefteq b) \cdot (a \lhd c) \tag{22}$$

$$a + (a \lhd b) = a \tag{23}$$

$$(a \lhd b) + b = a + b \tag{24}$$

$$a \cdot (a \lhd b) = a \lhd b \tag{25}$$

$$(a \lhd b) \cdot (b \lhd a) = \bot \tag{26}$$

$$(a \lhd b) \cdot (b \lhd c) \cdot (a \lhd c) = (a \lhd b) \cdot (b \lhd c) \tag{27}$$

A.2. Theorems satisfied by operator Inclusive BEFORE

Operator Inclusive BEFORE satisfies the following theorems, for all $a,b,c \in \mathcal{E}_{nr}$:

$$a \trianglelefteq a = a \tag{28}$$

$$\bot \trianglelefteq a = \bot \tag{29}$$

$$a \trianglelefteq \bot = a \tag{30}$$

$$a \trianglelefteq (b + c) = (a \trianglelefteq b) \cdot (a \trianglelefteq c) \tag{31}$$

$$a \trianglelefteq (b \cdot c) = (a \trianglelefteq b) + (a \trianglelefteq c) \tag{32}$$

$$a \trianglelefteq (b \lhd c) = (a \lhd b) + (a \cdot b \cdot (c \trianglelefteq b)) + (a \triangle b) \cdot (b \lhd c) \tag{33}$$

$$a \trianglelefteq (b \trianglelefteq c) = (a \lhd b) + (a \cdot b \cdot (c \lhd b)) + (a \triangle b) \cdot (b \trianglelefteq c) \tag{34}$$

$$(a + b) \trianglelefteq c = (a \trianglelefteq c) + (b \trianglelefteq c) \tag{35}$$

$$(a \cdot b) \trianglelefteq c = (a \trianglelefteq c) \cdot (b \trianglelefteq c) \tag{36}$$

$$(a \lhd b) \trianglelefteq c = (a \lhd b) \cdot (a \trianglelefteq c) \tag{37}$$

$$(a \trianglelefteq b) \trianglelefteq c = (a \trianglelefteq b) \cdot (a \trianglelefteq c) \tag{38}$$

$$a+(a \trianglelefteq b) = a \tag{39}$$

$$b+(a \trianglelefteq b) = a+b \tag{40}$$

$$a \cdot (a \trianglelefteq b) = a \trianglelefteq b \tag{41}$$

$$(a \trianglelefteq b) \cdot (b \trianglelefteq c) \cdot (a \trianglelefteq c) = (a \trianglelefteq b) \cdot (b \trianglelefteq c) \tag{42}$$

A.3. Simplification theorems

Temporal operators satisfy the following theorems, for all $a,b,c \in \mathcal{E}_{nr}$:

$$(a \trianglelefteq b)+(a \triangleleft b) = a \trianglelefteq b \tag{43}$$

$$(a \triangleleft b) \cdot (a \triangle b) = \perp \tag{44}$$

$$(a \trianglelefteq b) \cdot (a \triangleleft b) = a \triangleleft b \tag{45}$$

$$(a \triangleleft b) \cdot (b \trianglelefteq a) = \perp \tag{46}$$

$$(a \trianglelefteq b) \cdot (a \triangle b) = a \triangle b \tag{47}$$

$$(a \trianglelefteq b) \cdot (b \trianglelefteq a) = a \triangle b \tag{48}$$

$$(a \triangleleft b)+(a \triangle b)+(b \triangleleft a) = a+b \tag{49}$$

$$(a \cdot (b \triangleleft a))+(a \triangle b)+(b \cdot (a \triangleleft b)) = a \cdot b \tag{50}$$

$$(a \triangleleft b)+(a \triangle b)+(a \cdot (b \triangleleft a)) = a \tag{51}$$

$$(a \trianglelefteq b)+(b \trianglelefteq a) = a+b \tag{52}$$

$$(a \cdot (b \trianglelefteq a))+(b \cdot (a \trianglelefteq b)) = a \cdot b \tag{53}$$

$$(a \trianglelefteq b)+(a \cdot (b \trianglelefteq a)) = a \tag{54}$$

$$(a \triangleleft b) \cdot (b \triangleleft c) \cdot (a \trianglelefteq c) = (a \triangleleft b) \cdot (b \triangleleft c) \tag{55}$$

References

[1] Bobbio A, Codetta Raiteri D. Parametric fault trees with dynamic gates and repair boxes. In: Proceedings of the annual reliability and maintainability symposium (RAMS 2004), Los Angeles, CA, USA, 2004. p. 459–65.

[2] Boudali H, Dugan JB. A discrete-time Bayesian network reliability modeling and analysis framework. Reliability Engineering and System Safety 2005;87(3):337–49.

[3] Boudali H, Dugan JB. A continuous-time Bayesian network reliability modeling, and analysis framework. IEEE Transactions on Reliability 2006;55(1):86–97.

[4] Boudali H, Crouzen P, Stoelinga M. Dynamic fault tree analysis through input/output interactive Markov chains. In: Proceedings of the international conference on dependable systems and networks (DSN 2007). IEEE Computer Society; 2007. p. 25–38.

[5] Boudali H, Crouzen P, Stoelinga M. A compositional semantics for dynamic fault tree in terms of interactive Markov chains. In: Proceedings of the international symposium on automated technology for verification and analysis (ATVA'07). Lecture notes in computer science, vol. 4762. Springer Verlag; 2007. p. 441–56.

[6] Boudali H, Crouzen P, Stoelinga M. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Transactions on Dependable and Secure Computing 2010;7(2):128–43.

[7] Codetta Raiteri D. The conversion of dynamic fault trees to stochastic petri nets, as a case of graph transformation. Electronic Notes on Theoretical Computer Science 2005;127(2):45–60.

[8] Dugan JB, Bavuso SJ, Boyd MA. Dynamic fault-tree models for fault-tolerant computer systems. IEEE Transactions on Reliability 1992;41(3):363–77.

[9] Dugan JB, Sullivan KJ, Coppit D. Developing a low-cost high-quality software tool for dynamic fault-tree analysis. IEEE Transactions on Reliability 2000;49(1):49–59.

[10] Lauritzen SL, Jensen F. Stable local computation with conditional Gaussian distributions. Statistics and Computing 2001;11(2):191–203.

[11] Malhotra M, Trivedi K. Power-hierarchy among dependability model types. IEEE Transactions on Reliability 1994;43(3):493–502.

[12] Merle G, Roussel JM, Lesage JJ. Algebraic framework for the modelling of priority dynamic fault trees. Internal Report; 2008. Available: 〈http://www.lurpa.ens-cachan.fr/isa/aadft/documents/LURPA-2008-Framework.pdf〉.

[13] Merle G, Roussel JM, Lesage JJ, Bobbio A. Probabilistic algebraic analysis of fault trees with priority dynamic gates and repeated events. IEEE Transactions on Reliability 2010;59(1):250–61.

[14] Merle G. Algebraic modelling of dynamic fault trees, contribution to qualitative and quantitative analysis. PhD Thesis, École Normale Supérieure de Cachan, France, 2010.

[15] Montani S, Portinale L, Bobbio A, Varesio M, Codetta-Raiteri D. DBNet, a tool to convert dynamic fault trees into dynamic Bayesian networks. Università del Piemonte Orientale, Technical Report TR-INF-2005-08-02-UNIPMN; 2005.

[16] Moral S, Rumí R, Salmerón A. Mixtures of truncated exponentials in hybrid Bayesian networks. In: Proceedings of the 6th European conference on symbolic and quantitative approaches to reasoning with uncertainty. Lecture notes in artificial intelligence, vol. 2143; 2001. p. 145–67.

[17] Ortmeier F, Schellhorn G, Thums A, Reif W, Hering B, Trappschuh H. Safety analysis of the height control system for the Elbtunnel. Reliability Engineering and System Safety 2003;81(3):259–68.

[18] Rauzy A. Mathematical foundations of minimal cutsets. IEEE Transactions on Reliability 2001;50(4):389–96.

[19] Reay KA, Andrews JD. A fault tree analysis strategy using binary decision diagrams. Reliability Engineering and System Safety 2002;78(1):45–56.

[20] Stamatelatos M, Vesely W. Fault tree handbook with aerospace applications. NASA Office of Safety and Mission Assurance, vol. 1.1, 2002. p. 1–205.

[21] Trivedi K. Probability & statistics with reliability, queueing & computer science applications. 2nd ed. Wiley; 2001.

[22] Vesely WE, Goldberg FF, Roberts NH, Haasl DF. Fault tree handbook. Washington, DC, USA: US Nuclear Regulatory Commission; 1981.

[23] Walker M, Papadopoulos Y. Qualitative temporal analysis: towards a full implementation of the fault tree handbook. Control Engineering Practice 2009;17(10):1115–25.

[24] Yuge T, Yanagi S. Quantitative analysis of a fault tree with priority AND gates. Reliability Engineering and System Safety 2008;93(11):1577–83.

[25] Zhu H, Zhou S, Dugan JB, Sullivan KJ. A benchmark for quantitative fault tree reliability analysis. In: Proceedings of the annual reliability and maintainability symposium 2001 (RAMS 2001), Philadelphia, PA, USA, 2001. p. 86–93.