Diagnosticabilité modulaire appliquée au Diagnostic en ligne des Systèmes Embarqués Logiques

Ramla Saddem

N° d'ordre : 206

ECOLE CENTRALE DE LILLE

THESE

Présentée en vue
d'obtenir le grade de

DOCTEUR

En

Automatique, Génie Informatique, Traitement du Signal et Image

Par

Ramla SADDEM

DOCTORAT DELIVRE SIMULTANEMENT PAR L'ECOLE CENTRALE DE LILLE ET L'ECOLE NATIONALE DES SCIENCES DE L'INFORMATIQUE DANS LE CADRE D'UNE COTUTELLE INTERNATIONALE DE THESE

Diagnosticabilité modulaire appliquée au Diagnostic en ligne des Systèmes Embarqués Logiques

Soutenue le 10 décembre 2012 devant le jury d'examen :

Président	Eric NIEL, Professeur, INSA de Lyon, France
Rapporteur	Jean François PETIN, Professeur, Université de Lorraine, France
Rapporteur	Ouajdi KORBAA, Professeur, Université de Sousse, Tunisie
Examinateur	Riadh ROBBANA, Professeur, INSAT de Tunis, Tunisie
Directeurs de thèse	Armand TOGUYENI, Professeur, EC Lille, France Moncef TAGINA, Professeur, ENSI, Tunisie

Thèse préparée dans les laboratoires LAGIS et SOIE

Ecole Doctorale SPI 072 (Lille I, Lille III, Artois, ULCO, UVHC, EC Lille)
PRES Université Lille Nord-de-France

A *la mémoire de ma mère*

A *mon papa chéri*

A *ma sœur Rabeb*

A *mes p'tits Rim, Rahma et Rayene*

Remerciements

Ce travail de recherche a été préparé dans le cadre d'une co-tutelle entre l'Ecole Centrale de Lille, Laboratoire d'Automatique, Génie Informatique et Signal (LAGIS) et l'Ecole Nationale des Sciences de l'Informatique, Laboratoire Stratégie d'Optimisation et Informatique IntelligentE (SOIE), financé en partie par l'Institut Français de Tunis que je tiens à remercier et par le projet FerroCOTS, du programme FUI Vague 7.

Je tiens à exprimer toute ma gratitude et mes plus vifs remerciements envers mes directeurs de recherche, Pr. Armand Toguyéni, Professeur à l'Ecole Centrale de Lille, d'abord pour m'avoir invité et accueilli, ensuite pour m'avoir formé, guidé et soutenu et enfin pour son amitié, sa disponibilité, ses qualités humaines et professionnelles, et ses aides infinies dans la recherche et dans l'enseignement, et également le Pr. Moncef Tagina, Professeur à l'Ecole Nationale des Sciences de l'Informatique, pour le suivi de mes travaux de recherche depuis mon PFE, pour sa confiance, sa gentillesse et pour ses encouragements.

J'exprime mes remerciements à Monsieur Ouajdi Korbaa, Professeur à l'Université de Sousse, et à Monsieur Jean François Petin, Professeur à l'Université de Lorraine, pour avoir accepté de rapporter sur ma thèse. Je remercie également Monsieur Eric Niel, Professeur à l'INSA de Lyon, et Monsieur Riadh Robbana, Professeur à l'INSAT de Tunis, pour avoir accepté d'examiner ma thèse.

Remerciements à tous les doctorants et permanents de l'équipe STF du LAGIS, à tous les personnels de l'Ecole Centrale de Lille, à toutes mes amies du laboratoire Soie en Tunisie, Faten, Raoudha et Imtiez et à tous ceux qui, de près ou de loin, m'ont apporté leur aide pour réaliser ce travail. Merci à tous mes amis de Lille, Amouna, Asma, Manel, Rahma, Donia, Sana, Marouene, Nihel, Slah, Walid, Beya, Yacine, Amira, Wided, … Merci pour tous les bons moments que nous avons partagé ensemble. Merci à mes amis de Paris Aymen, Ines, Imen, Mouna, Rekka, Sana, Dali, Rim, safwene, Hassene, et une pensée particulière à la newyorkaise Rania, grâce à ses soutiens en deuxième années ENSI, j'ai pu franchir le domaine de la recherche.

Et enfin j'exprime mes remerciements à ma famille. Malgré la distance, elle est toujours présente et proche de mon cœur. Je lui dédie ce modeste travail, qu'elle y perçoit l'infinité de ma reconnaissance et y trouve l'expression de mon amour sans limite … Maman, t'es partie sans prévenir et sans voir l'achèvement de ce travail que t'as tant attendue ... Que Dieu t'accorde sa miséricorde … Tu me manques ... Papa, merci pour tes prières, ta grande confiance, ta patience, ton soutien, tes encouragements, tout ce que je fais, c'est pour que tu sois fier et content de moi ... Mes petits, merci pour votre confiance envers votre « Grande sœur », merci pour votre amour et j'espère être tout le temps à la hauteur de votre confiance. Rabeb, ma chérie, merci pour ton soutien et ton amour. Merci à mes tantes, en particulier, tata Radhiya et à tata Essya, pour l'amour et l'attention qu'elles me réservent sans oublier mes oncles, mes cousins et cousines et en particulier je voudrais remercier Sameh, Wafa et Safa mes cousines-sœurs et mon cousin Slim pour leur générosité et leur aides. Je vous aime. Je voudrais aussi remercier mes deux « sœurs » Rabaa et Amal et leurs familles pour leur amitié et leur amour qu'elles ne cessent de me prouver dans les meilleurs moments comme dans les pires.

Table des matières

Introduction générale 21
1. Motivations et objectifs 21
2. Contributions 21
3. Organisation du mémoire 23
I Diagnostic des systèmes embarqués logiques : contexte et problématique 25
Introduction du chapitre I 25
I.I Sûreté de fonctionnement et tolérance aux fautes 25
I.I.1 Terminologie 25
I.I.2 Attributs de la sûreté de fonctionnement 30
I.I.3 Moyens de la sûreté de fonctionnement 31
I.I.4 Conclusion 33
I.II Les systèmes embarqués logiques 34
I.II.1 Systèmes embarqués 34
I.II.2 Systèmes logiques et sécurité 37
I.II.3 Systèmes temps réel 40
I.II.4 Conclusion 40
I.III Problématique du diagnostic des systèmes embarqués logiques 41
I.III.1 Diagnostic des systèmes logiques 41
I.III.2 Diagnostic des composants programmables 48
I.III.3 Diagnostic des systèmes embarqués 53
Conclusion du chapitre I 55
II Outils et méthodes pour le diagnostic des SEL 57
Introduction du chapitre II 57
II.I Outils formels pour la modélisation du comportement des systèmes discrets 57
II.I.1 Automates à états finis et langages 57
II.I.2 Automates temporisés 62
II.I.3 Réseaux de Petri temporisés 67
II.I.4 Réseaux de Petri temporels 70
II.I.5 Conclusion 76

II.I.6 Comparaison de l'expressivité des AT et des RdP T-temporels 76
II.II Outils pour le diagnostic des systèmes discrets 79
II.II.1 Observateur discret 79
II.II.2 Diagnostiqueur de Sampath 81
II.II.3 Vérificateur de Jiang (Jiang, et al., 2001) 83
II.II.4 Les Signatures Temporelles Causales 84
Conclusion du chapitre II 88
III Diagnosticabilité pour la réduction de l'explosion combinatoire des SEL 89
Introduction du chapitre III 89
III.I Diagnosticabilité de Sampath (Sampath, et al., 1995)(Sampath, et al., 1996) : Approche centralisée 90
III.I.1 La méthode de Sampath 91
III.I.2 La méthode de Yoo 93
III.I.3 La méthode de Cassez 94
III.I.4 Conclusion 97
III.II Approches décentralisées 98
III.III Codiagnosticabilité et codiagnosticabilité conditionnelle 104
III.IV Approches modulaires 108
III.IV.1 Diagnosticabilité distribuée 108
III.IV.2 Diagnosticabilité de Debouk 109
III.IV.3 Diagnosticabilité modulaire de Contant 114
Conclusion du chapitre III 119
IV Diagnostic et model-checking applicables aux SEL 123
Introduction du chapitre IV 123
IV.I Diagnosticabilité des systèmes temporisés 123
IV.I.1 Expressivité des systèmes temporisés 124
IV.I.2 Vivacité d'un automate temporisé 125
IV.I.3 Test de diagnosticabilité 127
IV.I.4 Mise en œuvre du test de diagnosticabilité 131
IV.II Synthèse de diagnostiqueurs pour les systèmes temporisés : cas général 134
IV.III Synthèse de diagnostiqueurs pour les systèmes temporisés non-déterministes ... 136

IV.III.1 Test d'un diagnostiqueur potentiel 137
IV.III.2 Méthode de déterminisation d'un automate temporisé (Baier, et al., 2009) 139
IV.IV Calcul du délai minimal pour diagnostiquer une faute 148
IV.V Test de diagnosticabilité des systèmes temporisés par model- checking 149
IV.V.1 Présentation du model-ckecking 149
IV.V.2 Logique temporelle 150
IV.V.3 Présentation de KRONOS 157
IV.V.4 Présentation de UPPAAL 160
IV.V.5 Principe d'application du model-checking au test de la diagnosticabilité 165
Conclusion du chapitre IV 167
V Approche diagnostiqueur pour la surveillance des SEL 169
Introduction du chapitre V 169
V.I Les exigences de surveillance 169
V.II Approche composants 170
V.II.1 Décomposition d'un système en composants 170
V.II.2 Le concept de composant logique 172
V.III Abstraction de comportement par réduction d'un graphe d'états 175
V.III.1 Méthodes de réduction des systèmes logiques : état de l'art 175
V.III.2 Nouvelle méthode de réduction pour le diagnostic des SEL 176
V.IV Méthode de modélisation pour le diagnostic des systèmes embarqués logiques 182
V.IV.1 Description de la carte FerroCOTS 182
V.IV.2 Modèle générique d'un port d'entrée ou de sortie 184
V.IV.3 Transformation de modèles pour la construction systématique de l'automate temporisé du comportement normal d'un composant logique 187
V.IV.4 Obtention du modèle complet d'un composant logique 190
V.V Relation entre profondeur d'une modélisation d'un système complexe et explosion combinatoire 190
V.VI Diagnosticabilité modulaire des automates temporisés 191
V.VI.1 Adaptation de la diagnosticabilité modulaire 191
V.VI.2 Application au cas de la carte FerroCOTS 193
Conclusion du chapitre V 196

VI Signatures Temporelles Causales pour le diagnostic des SEL 199
Introduction du chapitre VI 199
VI.I Problématiques : cohérence d'une base de STC vs correction de l'algorithme d'interprétation 200
VI.II Méthode pour le contrôle de la cohérence d'une base de STC à l'aide des RdP T-temporels 203
VI.II.1 Réécriture d'un triplet à contrainte temporelle de type période 203
VI.II.2 Réécriture d'un triplet à contrainte temporelle de type date 204
VI.II.3 Réécriture d'un triplet à contrainte temporelle de type durée 205
VI.II.4 Règles de réécriture de STC 205
VI.II.5 Cas d'étude en productique 208
VI.III Méthode pour la vérification de la cohérence d'une base de STC par le model-checking 212
VI.III.1 Présentation de la méthode 212
VI.III.2 Application au diagnostic des défaillances d'un convoyeur à bande 214
VI.IV Algorithme d'interprétation basé sur la notion de monde 215
VI.V Application au cas de la carte FerroCOTS 218
Conclusion du chapitre VI 220
Conclusion générale 221
Bibliographie 225
Annexe A. Méthodes de réduction des systèmes logiques 237
La méthode de ligne correspondante 237
La méthode d'implication graphique 239
La méthode de partitionnement successif 240
Annexe B. Algorithme de réduction d'un graphe d'états à transitions événementielles gardées 241
Annexe C. Modèles comportementaux d'une fonction logique 245

Table des figures

Figure I-1 – Critères de classification des défaillances 27
Figure I-2 – Défaillance vs panne 28
Figure I-3 – Relation faute-défaillance dans un système (Boulanger, 2010) 28
Figure I-4 – Modes de défaillance (Zwingelstein, 1995) 29
Figure I-5 – Synthèse des relations entre fiabilité et disponibilité 31
Figure I-6 – Synthèse des paramètres relatifs à la maintenabilité 31
Figure I-7 – Schéma général de la surveillance en ligne des processus industriels (Zwingelstein, 1995) 33
Figure I-8 – Evolution du nombre de transistors dans les processeurs Intel d'après la loi de Moore (Kadionik, 2005) 35
Figure I-9 – Structure typique d'un système embarqué (Grisel, et al., 2012) 36
Figure I-10 – Exemple de schéma blocs-diagramme 37
Figure I-11 – Exemple de programmation d'une fonction logique en VHDL 37
Figure I-12 – Normes de sûreté de fonctionnement pour les systèmes industriels 39
Figure I-13 – Principaux termes utilisés dans la description de la structure d'un système 41
Figure I-14 – Modélisation structurelle d'un additionneur 4 bits 42
Figure I-15 – Exemple de système pour le diagnostic par la théorie logique 43
Figure I-16 – Schéma de principe de la structure d'un composant numérique 49
Figure I-17 – Schéma de principe du test des composants numériques 50
Figure I-18 – Architecture matricielle d'un FPGA 51
Figure I-19 – Structure d'un bloc logique programmable (Manfoumbi, et al., 2003) 52
Figure I-20 – Méthodes de détection des fautes sur les FPGA (Manfoumbi, et al., 2003) 53
Figure I-21 – Vue conceptuelle d'un système (van Gemund, et al., 2004) 54
Figure I-22 – Exemple illustratif de la méthode d'analyse des traces (van Gemund, et al., 2004) (Abreu, 2009) 55
Figure II-1 – Exemple d'automates non déterministes 58
Figure II-2 – Composition synchrone de deux automates 60
Figure II-3 – Exemple de SED partiellement observable (Jéron, et al., 2006) 60
Figure II-4 – Automate ε(G) correspondant au système de la Figure II-3. 61

Figure II-5 – Automate déterminisé de l'exemple de la Figure II-3 62
Figure II-6 – Exemple (1) d'automate temporisé 63
Figure II-7 – Exemple (2) d'automate temporisé 65
Figure II-8 – Automate des régions correspondant à l'exemple de la Figure II-7 66
Figure II-9 – Franchissement d'une transition dans un RdP T-temporisé 68
Figure II-10 – Franchissement d'une transition dans un RdP P-temporisé (David, et al., 1992) 69
Figure II-11 – Exemple de RdP T-temporel (Diaz, 2001) 71
Figure II-12 – Graphe de classes de l'exemple de la Figure II-11 dans le cadre de la règle PSPT (Diaz, 2001) 72
Figure II-13 – Partie d'un système de production (Sava, 2001) 74
Figure II-14 – RdP P-temporel 75
Figure II-15 – Exemple d'automate temporisé (Tripakis, 2002) 77
Figure II-16 – Transformation en RdP T-temporel en sémantique forte de l'automate de la Figure II-15 78
Figure II-17 – Arbre d'accessibilité partiel 80
Figure II-18 – Observateur discret correspondant au modèle de la Figure II-2 81
Figure II-19 – Exemple illustratif de la notion de pseudo-diagnostiqueur 83
Figure II-20 – Vérificateur du système de la Figure II-19 84
Figure II-21 – Entités temporelles utilisées dans les contraintes 87
Figure II-22 – Graphe temporel d'une STC d'après les contraintes de l'utilisateur 88
Figure III-1 – Exemple de système diagnosticable 91
Figure III-2 – Cycles d'états Fi-indéterminés (Sampath, et al., 1995) 92
Figure III-3 – Illustration des états ambigus et de la non diagnosticabilité 93
Figure III-4 – Automate G1 correspondant à l'automate G de la Figure III-1 95
Figure III-5 – Automate G2 correspondant à l'exemple de la Figure III-1 96
Figure III-6 – Automate de Büchi correspondant au système de la Figure III-1 96
Figure III-7 – Architecture d'un système décentralisé 98
Figure III-8 – Système à diagnostiquer dans le cadre de l'approche décentralisée (Debouk, et al., 2000) 100
Figure III-9 – Diagnostiqueurs des deux sites surveillant le système de la Figure III-8 101

Figure III-10 – Architecture d'un système de surveillance basé sur la diagnosticabilité distribuée (Sengupta, 1998) 109

Figure III-11 – Composants d'un système 111

Figure III-12 – Diagnostiqueurs des deux composants de la Figure III-11 111

Figure III-13 – Modèle G du système composite et son générateur G' 112

Figure III-14 – Automate de test du système composite 113

Figure III-15 – Diagnostiqueur de l'exemple de la Figure III-13 114

Figure III-16 – Contre-exemple montrant que la diagnosticabilité modulaire n'entraine pas la diagnosticabilité locale (Contant, et al., 2006) 118

Figure III-17 – Architecture du diagnostic modulaire (Contant, et al., 2006) 119

Figure IV-1 – Automate temporisé modélisant un système fautif (Tripakis, 2002) 124

Figure IV-2 – Blocage par incompatibilité entre invariant de l'état source et les gardes des transitions de sorties 125

Figure IV-3 – Exemple d'automate zénon 126

Figure IV-4 – Copie $Gt\ i$ du système de la Figure IV-1 pour $\alpha = 3$ 128

Figure IV-5 – Produit synchrone $Gt1||sGt2$ correspondant à l'automate temporisé de la Figure IV-4 129

Figure IV-6 – Automate $(Gt1||sGt2) - f2$ correspondant à l'automate temporisé de la Figure IV-4. 130

Figure IV-7 – Automate Div(y) 132

Figure IV-8 – Automate *D* obtenu à partir de l'exemple de Figure IV-1 133

Figure IV-9 – Diagnostiqueur par estimation d'état 136

Figure IV-10 – Automate $Obs\,(\Delta)$ 138

Figure IV-11 – Automate temporisé non déterministe 139

Figure IV-12 – L'arbre temporisé infini G_t^{∞} associé à l'automate temporisé de la Figure IV-11 140

Figure IV-13 – Abstraction de l'arbre temporisé infini en graphe des régions 142

Figure IV-14 – Construction d'un arbre des régions 142

Figure IV-15 – Déterminisation symbolique de l'arbre des régions de la Figure IV-14 144

Figure IV-16 – Automate temporisé déterministe $BG_{t,2}$ obtenu par déterminisation de G_t ... 145

Figure IV-17 – Exemple d'automate EDG_t non déterministe (Alur, et al., 1999) 146

Figure IV-18 – Automate temporisé déterministe construit à partir de l'exemple de la Figure IV-17 147

Figure IV-19 – Exemple d'automate IEG_t non déterministe (Suman et al., 2008) 148

Figure IV-20 – Illustration des différents connecteurs temporels 152

Figure IV-21 – Futur réifié et quantificateur E 152

Figure IV-22 – Quantificateur E associé au connecteur G 153

Figure IV-23 – Quantificateur A associé au connecteur F 153

Figure IV-24 – Quantificateur A associé au connecteur G 154

Figure IV-25 – Codage en KRONOS de l'automate temporisé de la Figure IV-1 158

Figure IV-26 – Transcription du modèle graphique en description textuelle. 159

Figure IV-27 – Modélisation avec UPPAAL de l'automate temporisé de la Figure IV-1 161

Figure IV-28 – Construction d'un état sous UPPAAL 161

Figure IV-29 – Construction d'une transition sous UPPAAL 162

Figure IV-30 – Déclaration sous UPPAAL des horloges et des variables de l'automate de la Figure IV-27 162

Figure IV-31 – Message d'erreur en cas d'erreur de conception sous UPPAAL 163

Figure IV-32 – Simulation de l'exemple de la Figure IV-27 164

Figure IV-33 – Vérification avec UPPAAL 164

Figure IV-34 – Automate Af 165

Figure IV-35 – Model-cheking en utilisant UPPAA pour le test de diagnosticabilité 166

Figure V-1 – Décomposition d'un système en sous-systèmes et composants 171

Figure V-2 – Arbre matériel 171

Figure V-3 – Schématisation d'un composant programmable 172

Figure V-4 – Flux entre les différents constituants d'un système logique 173

Figure V-5 – Schématisation d'un composant logique de base 174

Figure V-6 – Exemple de machine de Mealy (b) pour un composant logique à deux entrées (a) 176

Figure V-7 – Machine de Moore à « événements gardés » équivalente à l'exemple de la Figure V-6 178

Figure V-8 – Application des quatre premières règles de réduction sur l'exemple de la Figure V-7 180

Figure V-9 – Application des règles 5 et 6 sur l'exemple de la Figure V-7 181

Figure V-10 – Schéma simplifié de la carte FerroCOTS 183
Figure V-11 – Spécification des entrées/sorties de la carte FerroCOTS 184
Figure V-12 – Décomposition structurelle de la carte FerroCOTS 184
Figure V-13 – Schéma générique d'un port d'E/S 185
Figure V-14 – Abstraction temporelle du comportement d'un port d'E/S 185
Figure V-15 – Modèle comportemental générique d'un port d'entrée ou de sortie 186
Figure V-16 – Modèle générique du comportement complet (normal et défaillant) d'un port d'entrée ou de sortie 186
Figure V-17 – Etape 1 de la transformation de modèles 187
Figure V-18 – Etape 2 de la transformation de modèles 188
Figure V-19 – Automate temporisé résultant de la transformation de modèles 189
Figure V-20 – Modèle de comportement complet d'un composant logique à deux entrées .. 190
Figure V-21 – Modèle comportemental complet d'une ligne fonctionnelle d'une carte fille 194
Figure V-22 – Test de diagnosticabilité du modèle de la Figure V-21 avec UPPAAL 195
Figure V-23 – Modèle du voteur V_k par rapport à la carte fille i et la sortie k 196
Figure V-24 – Modèle comportemental complet d'une ligne fonctionnelle intégrant la carte fille et le voteur 197
Figure V-25 – Diagnostiqueur du modèle de la Figure V-23 197
Figure VI-1 – Illustration de l'algorithme basé sur la technique de l'automate dupliqué 202
Figure VI-2 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une période 204
Figure VI-3 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une date 204
Figure VI-4 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une durée 205
Figure VI-5 – Exemple de fusion de 2 occurrences d'un événement référent 206
Figure VI-6 – Exemple de fusion de l'occurrence d'un événement référent et d'un événement contraint 207
Figure VI-7 – Application des règles de construction sur la STC de l'équation (VI-5) 208
Figure VI-8 – convoyeur à bande 209
Figure VI-9 – Abstraction du comportement normal d'une partie du convoyeur 209
Figure VI-10 – Structure du système de surveillance pour le diagnostic des défaillances 210

Figure VI-11 – Evaluation avec ROMEO de l'ensemble de STC1, STC3 et STC5 212

Figure VI-12 – Réécriture des STC en AT 214

Figure VI-13 – Composition parallèle des AT de STC1, STC3 et STC5 215

Figure VI-14 – Arbre de résolution de l'algorithme des mondes 217

Figure VI-15 – Modèle comportemental d'une ligne fonctionnelle d'une carte fille 218

Figure VI-16 – Arbre de résolution de l'algorithme de reconnaissance sur l'exemple d'une carte fille de la carte FerroCOTS 219

Figure A-1 : Graphe d'implication de la Table A-4 240

Figure C-1 - Ligne fonctionnelle réduite aux éléments d'une carte fille 245

Figure C-2 : Modèle Comportemental du port d'entrée $ei1$ 245

Figure C-3 : Modèle Comportemental du port d'entrée $ei2$ 246

Figure C-4 : Modèle Comportemental du port d'entrée $si1$ 246

Figure C-5 : Modèle Comportemental de la fonction logique $Fi1$ 247

Liste des tableaux

Table II-1 – Synthèse de la comparaison entre RdP P-temporels et RdP T-temporels 76

Table II-2 – Synthèse de la comparaison entre AT et RdP T-temporels 78

Table III-1 – Exemple de diagnostic décentralisé dans le cas du protocole naïf 101

Table III-2 – Règles de décision du coordonateur dans le cadre du protocole n°2 de (Debouk, et al., 2000) 102

Table III-3 – Exemple de diagnostic d'un système à l'aide du protocole n°2 103

Table III-4 – Fusion des décisions locales 107

Table III-5 – Synthèse des différentes approches de diagnosticabilité 120

Table V-1 – Nombre d'états et de transitions pour les différents modèles d'une ligne fonctionnelle 191

Table V-2 – Observabilité des différents événements 194

Table A-1 - Exemple de réduction avec la méthode de ligne correspondante 238

Table A-2 - Transformation de la Table A-2 238

Table A-3 - Transformation de la Table A-2 239

Table A-4 - Exemple de réduction avec la méthode d'implication graphique 239

.

Glossaire

AT Automate Temporisé. *57,62-63 65, 67-79,89, 215*

CL Composant Logique. *170, 172, 174, 176, 177, 179, 182, 187-190, 193, 221*

CTL Computation Tree Logic. *149,153, 155-156*

CPLD Complex Programmable Logic Device. *36*

EPLD Erasable Programmable Logic Device. *36*

FPGA Field Programmable Gate Array. *36*

MDT Mean Down Time. 30

MTTF Mean Time To Failure. *30*

MUT Mean Up Time. *30*

PLD Programmable Logic Device. *36*

PAL Programmable Array Logic. *36*

PLA Programmable Logic Array. *36*

RdP Réseaux de Petri. 23, *24, 57, 67-79, 88, 199, 203-205, 211, 220, 222-223*

SdF Sûreté de Fonctionnement. *23, 25-26, 28, 30-31, 33, 39*

SED Système à Evénements Discrets. 21, *23, 28, 48, 53, 56-57, 60, 77, 84, 89, 93-94, 123, 125, 131, 134, 191-192, 225*

SEL Système Embarqué Logique. 21, *57, 89, 119-121, 123, 165*

STC Signatures Temporelles Causales. 22-24, 57, 84-88, 199-203, 205-208, 210-223

TCTL Timed Computation Tree Logic. *24, 123, 149-150, 155-157, 159, 161, 163, 168*

.

Introduction générale

1. Motivations et objectifs

Aujourd'hui, les systèmes embarqués sont de plus en plus utilisés pour contrôler les systèmes complexes. Dans ce travail de thèse, nous nous intéressons aux systèmes embarqués critiques qui peuvent être utilisés pour la commande de systèmes de transport comme les systèmes ferroviaires ou les véhicules autonomes intelligents. Ces systèmes sont des systèmes critiques car ils permettent de contrôler des fonctions critiques de ces systèmes (Par exemple le système de freinage d'un train). Aussi, en cas de dysfonctionnement, il est nécessaire de réagir très vite afin de les maintenir dans un mode de fonctionnement sécurisé.

Le but de ce travail est de permettre la conception de systèmes tolérants aux fautes pour le contrôle-commande des systèmes de transport. Ce travail est une contribution au développement de systèmes embarqués logiques tolérants aux fautes. L'idée que nous développons est de rendre les systèmes embarqués tolérants aux fautes en concevant des architectures redondantes. Ces architectures devront être surveillées en ligne afin de gérer la commutation des modes en fonction de l'état du système.

Ces travaux de thèse sont inspirés d'une problématique qui nous a été proposée dans le cadre du projet FerroCOTS. Ce projet a été réalisé du 1er juin 2009 au 31 décembre 2012 à l'initiative d'un fabricant de trains. Il a eu pour objectif l'utilisation des composants numériques et la conception de systèmes embarqués pour le contrôle-commande des systèmes roulants ferroviaires. En effet, la fabrication de plus en plus importante des circuits intégrés à forte intégration en transistors et la baisse de leurs coûts permettent de les utiliser pour la mise en œuvre des systèmes embarqués. Malgré leurs nombreux avantages, les systèmes embarqués restent peu utilisés pour le contrôle des systèmes critiques. Cela est dû aux normes de sécurité industrielle qui contraignent les industriels à utiliser des technologies éprouvées comme celle basée sur les commandes à relais.

2. Contributions

Notre étude concerne notamment les Système Embarqués Logiques (SEL) qui sont une classe des Systèmes à Evénements Discrets (SED). Nous nous intéressons plus particulièrement au diagnostic de leurs défaillances. Dans le contexte du diagnostic des SED, nous distinguons deux types d'événements : les événements observables, issus des signaux émis par les capteurs et/ou les ordres de la partie commande, et les événements inobservables tels que les défaillances. Ainsi, l'objectif du diagnostic est de détecter et/ou d'identifier (si possible) les défaillances à partir de l'occurrence d'événements observables. Dans ce cadre, nous pouvons

distinguer deux approches de base pour le diagnostic de défaillances : une approche basée sur les diagnostiqueurs et une approche basée sur les Signatures Temporelles Causales (STC).

La principale limite de l'approche diagnostiqueur réside dans la gestion de l'explosion combinatoire liée au formalisme des automates. En effet, l'approche classique consiste à décomposer le système en composants. Ensuite, un modèle automate est associé à chaque composant. Le diagnostiqueur global du système dans une approche centralisée est alors obtenu par le produit synchrone des automates de ses composants. C'est ce produit synchrone qui engendre souvent une explosion combinatoire quand le système est complexe. Pour faire face à cette combinatoire, des travaux sont en cours dans deux directions : d'une part la conception de diagnostiqueurs décentralisés ou distribués, d'autre part la construction d'observateurs basés sur les Réseaux de Petri. Dans ces travaux, notre verrou principal est de combattre l'explosion combinatoire. Nous proposons une nouvelle méthode basée sur l'ingénierie par les modèles pour le diagnostic des systèmes embarqués logique. Cette méthode est basée sur une approche structurelle du système. En effet, elle consiste à le décomposer, dans un premier temps, en un ensemble de composants. Ensuite, nous étudions chaque composant pour construire un modèle de son comportement. La construction du modèle de comportement d'un composant est basée sur une double abstraction : une abstraction du comportement séquentiel et une abstraction du comportement temporel. La disponibilité d'un modèle comportemental du système est une condition nécessaire pour l'application de la technique du diagnostiqueur que nous proposons d'utiliser ici. Mais les systèmes auxquels, nous nous intéressons étant caractérisés par de nombreuses entrées/sorties, la construction d'un modèle du système peut se heurter à un problème d'explosion combinatoire. Aussi, afin de faire face à ce risque, nous proposons une nouvelle approche de diagnostic distribué qui prend en compte la décision de diagnostic du diagnostiqueur de chaque composant du système surveillé afin de caractériser l'état du système (Saddem, et al., 2011a)(Saddem, et al., 2011b).

Le diagnostic à l'aide des STC consiste à interpréter en ligne l'occurrence d'événements afin d'instancier les motifs qui seront reconnus. Une STC est reconnue lorsque tous ses événements se produisent en respectant leurs contraintes. La limitation majeure de l'approche STC est la garantie de la cohérence de la connaissance anormale ainsi encodée. Un deuxième niveau de difficulté réside dans l'interprétation des événements en entrée du système de diagnostic dans le cadre de l'hypothèse de défaillances multiples. Le problème posé dans ce cadre, est la possibilité qu'une séquence d'événements produits par plusieurs défaillances entraîne un surdiagnostic, c'est à dire l'isolation de défaillances supplémentaires. En effet, la connaissance codée est généralement une connaissance experte. Dans le meilleur des cas, elle peut être dérivée à partir de connaissances comme les arbres de défaillance. Toutefois, le problème est que chaque STC ou chronique est construite séparément sans tenir compte des autres règles. Aussi, il demeure nécessaire de vérifier la cohérence d'un ensemble de règles pour assurer la complétude et le déterminisme dans l'hypothèse de défaillances multiples.

Dans ce mémoire, nous proposons trois contributions pour améliorer le diagnostic à l'aide des STC. Les deux premières contributions concernent la vérification de la cohérence (Saddem, et al., 2010) et (Saddem, et al., 2011c). La troisième contribution concerne la correction de l'algorithme de diagnostic interprétant les STC dans le contexte de défaillances multiples (Saddem, et al., 2012).

3. Organisation du mémoire

Ce mémoire comporte six chapitres. Il est organisé comme suit :

Dans le chapitre I, nous posons le contexte de travail et la problématique de cette thèse. Nous définissons dans une première partie, les principaux concepts de la sûreté de fonctionnement que nous utilisons dans ce travail. Nous essayons, dans une seconde partie, de caractériser les systèmes sur lesquels va porter cette étude. Ensuite, nous introduisons la notion de systèmes embarqués logiques. Dans la quatrième partie, nous présentons un état de l'art synthétique relatif au diagnostic des systèmes embarqués, notamment du point de vue des composants programmables de type FPGA.

Dans le chapitre II, nous nous intéressons aux outils et méthodes pour le diagnostic des systèmes discrets. Nous présentons dans un premier temps quelques outils formels utilisés pour modéliser le comportement des systèmes discrets. Nous commençons ainsi par présenter les automates à états finis et les automates temporisés. Ensuite, nous présentons les Réseaux de Petri (RdP) temporisés et les RdP temporels. Dans un second temps, nous présentons des outils pour le diagnostic des systèmes discrets. Nous introduisons d'abord le concept d'observateur discret à partir duquel nous présenterons les notions de diagnostiqueur et de vérificateur. Nous terminons le chapitre II par la présentation des Signatures Temporelles Causales (STC) ou chroniques avec une approche pour le diagnostic des SED.

Dans le chapitre III, nous proposons une synthèse sur la diagnosticabilité des SED. En effet, il existe dans la littérature de nombreuses définitions de la diagnosticabilité. Nous avons donc cherché à montrer les relations entre les différentes définitions afin de proposer une vision unifiée dans le cadre d'une approche de diagnostic distribué. Nous commençons donc par présenter l'approche centralisée qui est l'approche originelle. Nous présentons les différentes méthodes de vérification de la diagnosticabilité, celle de (Sampath, et al., 1995), de (Yoo, et al., 2002) et de (Cassez, et al., 2008). Ensuite, nous passons à l'approche décentralisée. Nous présentons les travaux de (Debouk, et al., 2000). Après, nous nous intéressons à la codiagnosticabilité et la codiagnosticabilité conditionnelle. Enfin, nous présentons les approches modulaires (Sengupta, 1998) (Debouk, 2003) (Contant, et al., 2006).

Un moyen pour nous de combattre l'explosion combinatoire a été le choix d'un formalisme plus expressif que ceux utilisés pour modéliser les SED. Ainsi, nous avons choisi de développer nos travaux en nous appuyant sur les automates temporisés. Aussi, dans le chapitre IV, dans un premier temps, nous étudions le cas général des systèmes temporisés

modélisés par des automates temporisés. Nous nous intéresserons d'une part à leur diagnosticabilité, et d'autre part à la construction de diagnostiqueurs pour ce type de système. Nous verrons notamment que dans le cas général, à partir d'un modèle d'automate temporisé, un diagnostiqueur n'est pas forcément un automate temporisé, mais tout simplement une machine de Türing (Tripakis, 2002). C'est la raison pour laquelle, dans un second temps, nous nous intéressons au cas particulier des automates temporisés déterministes. Nous verrons que dans ce cadre, certains résultats des automates à états fini s'appliquent (Bouyer, et al., 2005a). Nous présentons ensuite la technique du model-checking et la logique temporelle TCTL pour vérifier le test de diagnosticabilité des modèles de nos systèmes. Nous montrerons une brève présentation de deux outils de model-checking, KRONOS et UPPAAL que nous exploitons dans le chapitre V.

Dans le chapitre V, nous proposons une nouvelle méthode basée sur l'ingénierie par les modèles pour le diagnostic des systèmes embarqués logiques. Cette méthode est basée sur une décomposition structurelle d'un système en composants. Nous montrons alors la dualité entre modélisation fine pour l'identification des composants en faute et la problématique de l'explosion combinatoire. Dans ce chapitre, nous proposons une extension de la diagnosticabilité modulaire au contexte des systèmes temporisés. Au travers de l'application de nos propositions au cas d'étude de la carte FerroCOTS, nous montrons toutefois les limites de l'approche diagnostiqueur.

Dans le chapitre VI, nous présentons trois contributions pour garantir la correction de l'interprétation d'une base de STC. Nous expliquons à travers un exemple, la problématique de la non cohérence d'une base de STC et les risques de surdiagnostic qui en découle lorsque nous utilisons une technique d'interprétation comme celle des automates dupliqués (Bertrand, 2009). Ensuite, notre première contribution consiste en l'exploitation des RdP T-temporels pour vérifier la cohérence d'une base de STC (Saddem, et al., 2010). Par la suite, dans un souci de cohérence avec le formalisme utilisé dans l'approche diagnostiqueur, la deuxième contribution est la proposition d'une méthode basée sur la technique du model-checking sur des automates temporisés pour vérifier la cohérence d'une base de STC (Saddem, et al., 2011c). Nous terminons par notre troisième contribution où nous proposons un algorithme d'interprétation basé sur le concept de monde qui garantit la correction du diagnostic (Saddem, et al., 2012).

I Diagnostic des systèmes embarqués logiques : contexte et problématique

Introduction du chapitre I

L'objectif de ce chapitre est de définir le contexte de travail de cette thèse et sa problématique. Comme présenté dans l'introduction générale de ce mémoire, le but de ce travail est de permettre la conception de systèmes tolérants aux fautes pour le contrôle-commande des systèmes de transport. Considérons par exemple la problématique du freinage d'un train. Aujourd'hui le système de freinage d'un train est basé sur des technologies à relais. Le choix de cette technologie est lié aux normes de sécurité concernant le développement et l'exploitation de ces systèmes. Ces normes rendent en effet difficile l'utilisation de technologies récentes comme celles basées sur des composants numériques pouvant mettre en œuvre à la fois des calculs numériques et des fonctions logiques. Ce travail est donc une contribution au développement de systèmes embarqués logiques tolérants aux fautes. L'idée que nous développerons est de rendre des systèmes embarqués tolérants aux fautes en concevant des architectures redondantes. Ces architectures devront être surveillées en ligne afin de gérer la commutation des modes en fonction de l'état du système.

Dans ce chapitre nous commencerons par définir les principaux concepts de la sûreté de fonctionnement que nous utiliserons dans ce travail. Cela nous permettra donc de définir plus précisément les notions de tolérance aux fautes et de surveillance que nous avons utilisées dans cette introduction. Nous essayerons, dans une seconde partie, de caractériser les systèmes sur lesquels va porter cette étude. En effet la notion de système embarquée et une notion qui recouvre différentes réalités. Ensuite, nous introduirons la notion de systèmes embarqués logiques. Dans la quatrième partie, nous présenterons la problématique du diagnostic des systèmes embarqués logiques.

I.I Sûreté de fonctionnement et tolérance aux fautes

La sûreté de fonctionnement (dependability en anglais) est au sens large du terme, la science qui s'intéresse au dysfonctionnement des systèmes. Elle est souvent présentée comme la « science des défaillances » (Zwingelstein, 1995).

I.I.1 Terminologie

Nous allons donc commencer par définir ce qu'est une défaillance et d'une manière générale la terminologie utilisée.

Définition I-1: La défaillance (CEI, 1974) (Zwingelstein, 1995)

C'est la cessation de l'aptitude d'une entité à assurer une fonction requise dans des conditions données.

Dans la Définition I-1, l'entité est un terme générique pour indiquer que cette définition s'applique aussi bien à un système qu'à des composants. Cette définition fait référence à la notion de fonction. Cela signifie qu'on parlera de défaillance dès qu'une entité ne délivre plus une des fonctions pour laquelle elle a été conçue dans des conditions normales de fonctionnement. Par conditions normales, nous entendons les conditions physiques (température, pression, taux d'humidité …) ou opérationnelle d'exploitation de l'entité.

Considérons par exemple les portes d'un train. Lorsque le train est en mouvement à une vitesse dépassant une limite fixée par les normes de sécurité, elles ne doivent pas être ouvertes, ni s'ouvrir sur sollicitation d'un passager. En cas d'ouverture, il y a manifestation d'une défaillance du système porte.

Considérons l'exemple de l'éclatement d'un pneu de voiture. Peut-on parler de défaillance, lorsque l'éclatement du pneu est causé par une pression insuffisante, donc à des conditions d'utilisation inadéquate ? Pour l'usager le résultat s'apparente à une défaillance. Par contre d'un point de vue juridique, si le constructeur peut démontrer que le pneu n'a pas été utilisé dans des conditions normales, la défaillance pour faute interne ne sera pas retenue.

Dans la terminologie de la sûreté de fonctionnement, on utilise d'autres termes que celui de défaillance. Elle définit notamment les termes de panne, erreur, faute et mode de défaillance.

Définition I-2 : La panne (Villemeur, 1988)

C'est l'inaptitude d'une entité à assurer une fonction requise.

Cette définition est très proche de celle relative à la défaillance. Mais elle introduit toutefois une nuance. Le terme « inaptitude» définit un état, alors que l'expression « cessation de l'aptitude », s'apparente à un événement. Nous pouvons donc en déduire que la défaillance est un processus faisant passer une entité d'un état de fonctionnement normal à un état de fonctionnement anormal. Nous utilisons ici le terme fonctionnement anormal et non « panne » car nous allons voir ultérieurement que l'occurrence d'une défaillance n'entraine pas toujours immédiatement la panne.

Il existe plusieurs manières de classifier les défaillances. En général les classifications utilisent les critères suivants : le domaine, la perception par des utilisateurs ou la criticité des conséquences (cf. Figure I-1).

Du point de vue du domaine, on distingue deux types de défaillances : les défaillances cataleptiques et les défaillances par dégradation. En effet, on a l'habitude de classer les défaillances en fonction de leur valeur (complète vs partielle) et de leur caractérisation temporelle (progressive vs soudaine).

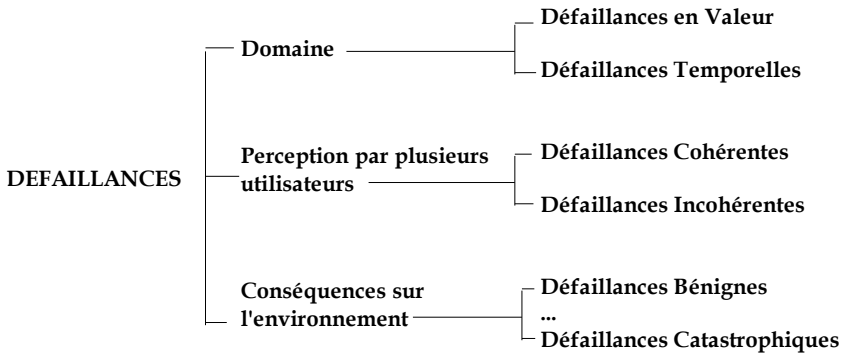


Figure I-1 – Critères de classification des défaillances

Du point de vue du domaine, on distingue deux types de défaillances : les défaillances cataleptiques et les défaillances par dégradation. En effet, on a l'habitude de classer les défaillances en fonction de leur valeur (complète vs partielle) et de leur caractérisation temporelle (progressive vs soudaine)

Définition I-3 : Défaillance cataleptique (CEI, 1974) (Villemeur, 1988)

C'est une défaillance qui est à la fois soudaine et complète.

Ce type de défaillance fait transiter directement l'entité considérée d'un état normal à un état de panne. Donc une défaillance cataleptique est assimilable à un événement. C'est par exemple le cas lorsqu'on essaie de démarrer une voiture et qu'elle refuse de démarrer sans avoir auparavant montré des signes de dysfonctionnement.

Définition I-4 : Défaillance par dégradation (CEI, 1974) (Villemeur, 1988)

C'est une défaillance qui est à la fois partielle et progressive.

Cette définition montre que toute défaillance n'est pas un événement, mais un processus. Ce processus a un début qui peut même ne pas être perceptible quand on n'a pas les moyens de perception adéquats. La progressivité signifie que son amplitude va augmenter jusqu'à atteindre une limite qui va faire basculer l'entité dans un état de panne. Considérons l'exemple de la défaillance du roulement d'une roue de voiture. En général, au bout d'un certain nombre d'utilisation, le roulement commence à émettre des bruits qui montrent qu'il arrive en fin de vie. Lorsqu'on entend ces bruits, cela signifie que la défaillance est en cours. Mais on n'est pas encore dans un état de panne. On peut encore utiliser le roulement pendant un certain temps.

La Figure I-2 illustre les différences comportementales entre défaillance cataleptique et défaillance par dégradation. Elle permet également de positionner l'état « en panne » comme l'état atteint par une entité quand la défaillance est complète.

Les exemples donnés pour illustrer les deux classes de défaillances précédentes, montrent qu'elles ne peuvent pas être traitées de la même manière. Les défaillances par dégradation, du

fait de leur progressivité, elles peuvent être traitées par des méthodes basées sur des modèles continus. Par contre les défaillances cataleptiques sont plutôt traitées par des méthodes discrètes (modèles et méthodes issus des Systèmes à Evénements Discrets ou SED).

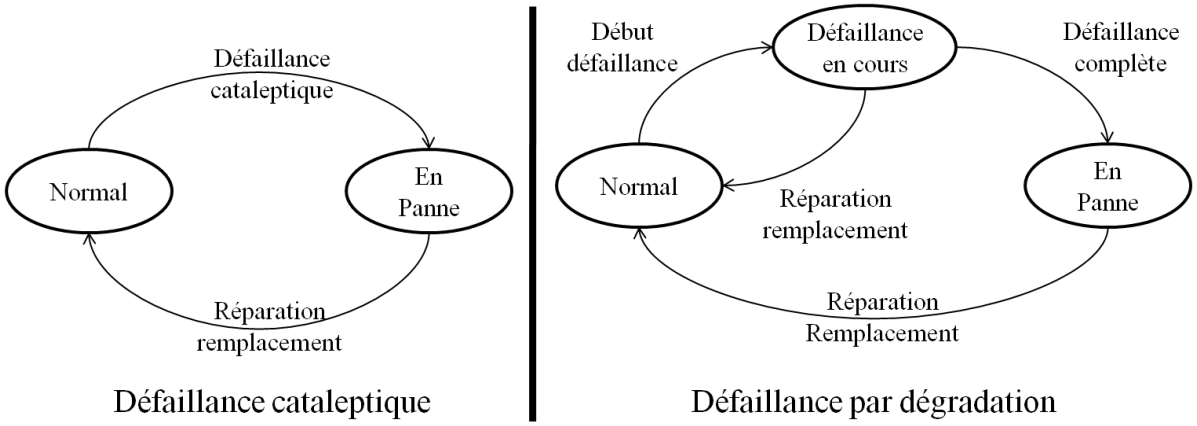


Figure I-2 – Défaillance vs panne

Dans la littérature, selon les communautés, on n'utilise pas les mêmes termes pour parler de dysfonctionnement. Dans la communauté automatique on utilise le terme de faute. D'ailleurs l'ensemble des méthodes développées pour le traitement des dysfonctionnements des systèmes industriels est connu sous la dénomination de « Fault Detection and Isolation (FDI) » dans la communauté internationale (IFAC …). Dans la communauté informatique, on utilise plutôt le terme de défaillance. Nous pouvons nous demander si ces termes sont interchangeables et définissent la même chose. Dans la réalité, il y a une causalité phénoménologique entre faute, erreur et défaillance. Une faute résulte en général d'une défaillance. L'activation d'une faute dans une entité engendre une erreur. La propagation de l'erreur aboutit à une nouvelle défaillance. Cette nouvelle défaillance pourra à son tour engendrer une faute et le processus est itéré. Fautes, erreurs et défaillances constituent les entraves à la sûreté de fonctionnement d'un système.

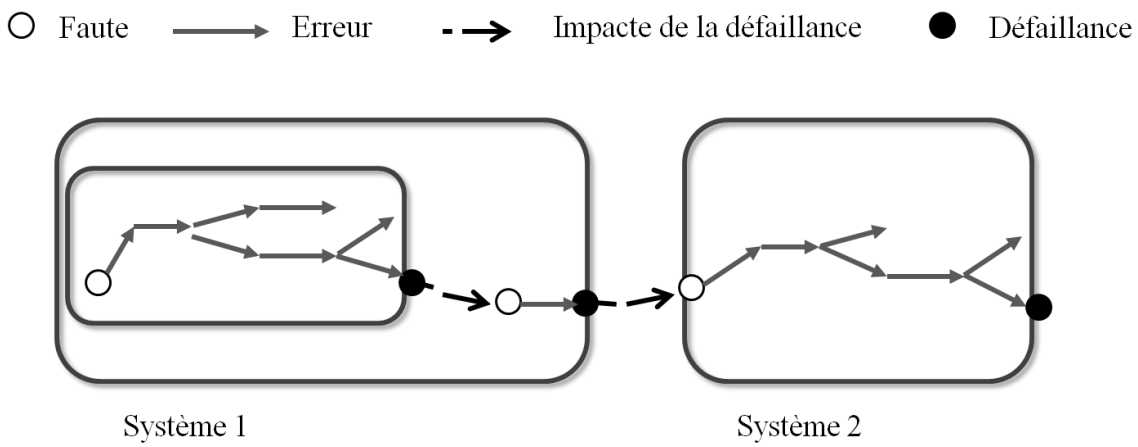


Figure I-3 – Relation faute-défaillance dans un système (Boulanger, 2010)

Nous voyons que les deux notions de « faute » et « défaillance » sont différentes mais très proches. Elles dépendent du point de vue et du niveau d'abstraction. Ainsi, la défaillance d'un composant est une faute pour le système qui le contient, ou au moins, pour les composants qui sont en interaction avec celui défaillant (Figure I-3). Dans cette étude, nous utiliserons indifféremment l'un ou l'autre des termes en supposant qu'ils sont interchangeables. Ce qu'il faudra retenir c'est qu'une défaillance se manifeste dans un système par ses conséquences. On parle alors de mode de défaillances.

Définition I-5 : Modes de défaillances (CEI, 1974) (Zwingelstein, 1995)

C'est l'effet par lequel est observée une défaillance.

En fait, il faut noter que cet effet doit être rapporté à une entité et une fonction bien précise. Cet effet peut comme on l'a vu dans le cadre de la relation faute-défaillance, se propager dans un système de composants en sous-ensemble, jusqu'au système lui même. Si on se réfère à la Figure I-4, l'objectif est donc de localiser la faute le plus près de son origine, pour la confiner rapidement dans une petite partie du système.

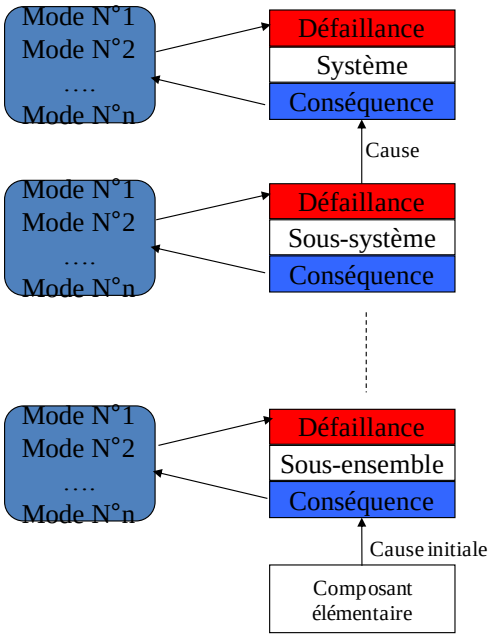


Figure I-4 – Modes de défaillance (Zwingelstein, 1995)

Une faute ou défaillance peut être perçue différemment par des observations distinctes. En effet, une faute peut apparaître, puis disparaitre selon l'instant où on l'observe. On parle alors de faute intermittente. Si une faute ne peut plus disparaître une fois qu'elle est apparue sans traitement de la cause de son apparition, on parle de faute permanente. Mais une faute peut être également perçue différemment par des capteurs répartis à différents endroits d'une entité surveillée. Lorsqu'elle est perçue différemment, on parle de faute incohérente. Dans le cas contraire, si la perception est la même quelque soit la position spatiale, on parle de faute cohérente.

I.I.2 Attributs de la sûreté de fonctionnement

Les attributs de la sûreté de fonctionnement (SdF) sont la fiabilité, la disponibilité, la maintenabilité et la sécurité-innocuité. A ces quatre attributs de base, pour certains systèmes (comme les systèmes informatiques), on peut ajouter la confidentialité et l'intégrité.

Définition I-6 : La fiabilité – reliability - (Zwingelstein, 1995)

C'est la probabilité qu'une entité E accomplisse une fonction requise sur [0, t] sachant qu'elle est non défaillante à t = 0.

$$R(t) = P(E\ non\ défaillant\ sur\ [0,t]) \quad \textbf{(I-1)}$$

A partir de cette définition, on distingue deux types de fiabilité : la fiabilité opérationnelle et la fiabilité prévisionnelle. La fiabilité opérationnelle est une fiabilité basée sur l'expérience et l'observation de systèmes comparables au système concerné. En effet, l'analyse de la fiabilité d'entités identiques dans les mêmes conditions opérationnelles permet d'estimer la fiabilité d'un système du même type. La fiabilité prévisionnelle consiste à prédire la fiabilité future d'un système à partir de la connaissance de la fiabilité de ses composants.

A partir de la fiabilité, on peut calculer le temps moyen de bon fonctionnement d'un système. C'est le temps moyen de fonctionnement avant l'occurrence de la première défaillance (Mean Time To Failure ou MTTF). Il est donné par l'équation (I-2).

$$MTTF = \int_0^{+\infty} R(t) \quad \textbf{(I-2)}$$

On peut également calculer le MTBF ou temps moyen entre deux défaillances.

Si la fiabilité permet a priori d'estimer la capacité d'une entité à être opérationnelle sur une période de temps, elle ne permet pas de dire si elle sera opérationnelle à une date donnée. C'est la disponibilité qui permet d'évaluer cette probabilité.

Définition I-7 : La disponibilité – Availability - (Zwingelstein, 1995)

C'est la probabilité que l'entité E soit en état d'accomplir une fonction requise dans des conditions données, à l'instant t.

$$A(t) = P(E\ non\ défaillant\ \text{à}\ l'instant\ t) \quad \textbf{(I-3)}$$

Comme pour la fiabilité, on définit des temps moyens caractérisant la disponibilité d'un système. Il s'agit du MUT ou Temps Moyen de Disponibilité (Mean Up Time) qui indique la durée moyenne de fonctionnement après réparation. On a également le MDT ou temps moyen d'indisponibilité (Mean Down Time). Il indique la durée moyenne d'indisponibilité d'une entité après une panne.

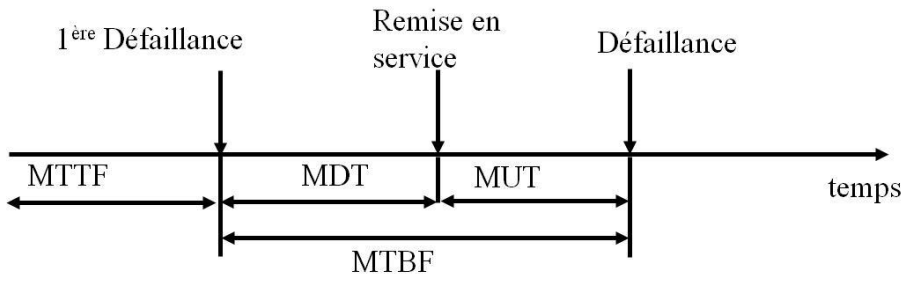


Figure I-5 – Synthèse des relations entre fiabilité et disponibilité

On montre que le MTBF est lié à ses deux temps moyens par la relation (I-4).

$$MTBF = MDT + MUT \tag{I-4}$$

En cas de panne, il est nécessaire de mesurer la capacité à réparer une entité. C'est la maintenabilité qui mesure cette capacité.

Définition I-8 : La maintenabilité – Maintenability –

C'est l'aptitude d'une entité à être réparée à la date t sachant qu'elle est en panne.

$$M(t) = P(la\ maintenance\ de\ E\ terminée\ à\ t) = 1 - P(E\ non\ réparé\ sur\ la\ durée\ [0,t]) \tag{I-5}$$

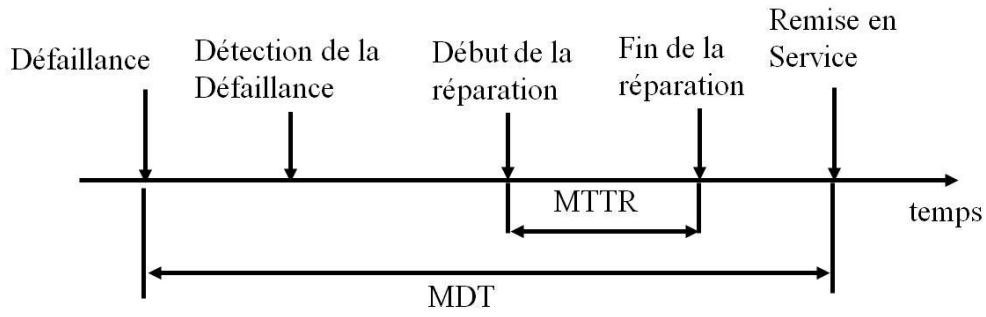


Figure I-6 – Synthèse des paramètres relatifs à la maintenabilité

A partir de la maintenabilité, on peut également calculer le MTTR ou temps moyen avant réparation.

$$MTTR = \int_0^{+\infty} \big(1 - M(t)\big)dt \tag{I-6}$$

I.I.3 Moyens de la sûreté de fonctionnement

L'objectif des moyens de la sûreté de fonctionnement est soit de prévenir les fautes soit de les traiter. On en distingue principalement quatre :

- La prévention des fautes,
- L'élimination des fautes,

- La prévision des fautes,
- La tolérance aux fautes.

La prévention de fautes a pour objectif d'empêcher l'occurrence de fautes ou au moins d'en diminuer la probabilité d'occurrence. Les outils utilisés résultent notamment de l'ingénierie des systèmes. On utilisera des méthodes de conception permettant a priori de prévenir les fautes humaines. Par exemple, dans le cadre de l'automatisation des systèmes de productique, la génération assistée par ordinateur de la commande est un moyen de prévention des fautes. Par contre, il est plus difficile de prévenir les fautes physiques. Un moyen consiste à utiliser des composants extrêmement fiables, ou du moins qui ne risquent pas d'être défaillants pendant la durée de vie du système. Mais cela a un coût qui rend cette approche limitée à des systèmes critiques comme par exemple ceux de l'aéronautique.

L'élimination des fautes consiste à rechercher les fautes pour les supprimer du système. On peut distinguer deux contextes : en phase de développement ou en exploitation. En développement, les outils utilisés sont la vérification, le diagnostic et la correction. La vérification permet a postériori de vérifier des propriétés intrinsèques du système. Elle peut être manuelle ou assistée dans le cas de systèmes complexes. La vérification assistée est basée sur une modélisation formelle du système et des propriétés à vérifier. Elle utilise des techniques comme le model-checking ou le theorem proving. Nous utiliserons le model-checking dans le chapitre V pour la vérification des propriétés pour la diagnosticabilité de systèmes embarqués logiques. Dans tous les cas, la vérification permet d'identifier des fautes latentes et des erreurs. Le diagnostic est lui couplé à des tests. Une fois l'entité réalisée, on lui applique des tests définis en phase de conception. L'analyse des résultats de ces tests permet de diagnostiquer la présence ou pas de fautes. Les étapes de vérification et de tests-diagnostic doivent être suivies d'une étape de correction pour l'élimination effective des fautes internes identifiées. En phase d'exploitation, l'élimination des fautes s'appuie sur une politique de maintenance : soit la maintenance préventive, soit la maintenance corrective. La maintenance préventive est appliquée avant l'occurrence de la panne. Elle est en général systématique. La maintenance corrective a pour objectif d'éliminer les fautes identifiées. Elle est en général précédée d'une étape de diagnostic.

La prévision des fautes n'est pas une technique ayant pour but la suppression des fautes. Elle a pour but d'identifier les fautes potentielles que peut connaître une entité pendant sa durée de vie. Deux grandes familles d'approches sont possibles : les approches qualitatives et les approches quantitatives. Ces approches sont basées sur des modèles qualitatifs ou stochastiques. Ces modèles peuvent être évalués par des tests ou par des simulations. La prévision des fautes peut être utilisée comme phase préliminaire avant l'élimination des fautes ou pour la conception d'architectures tolérantes aux fautes.

La tolérance aux fautes est un moyen complémentaire à ceux vu précédemment. En effet, il a pour objectif, non pas d'éliminer les fautes mais d'adapter le comportement du système à l'occurrence des fautes. Cela se traduit en pratique par la mise en œuvre d'architectures

redondantes qui permettent à une entité de poursuivre sa mission en dépit de l'occurrence de défaillances. La redondance est soit active, soit passive. Dans ce dernier cas, elle nécessite d'être activée. Dans tous les cas, il est nécessaire de surveiller en ligne le système pour détecter les fautes et alerter les opérateurs de conduite et de maintenance, voire même pour activer les fonctions de traitement des fautes.

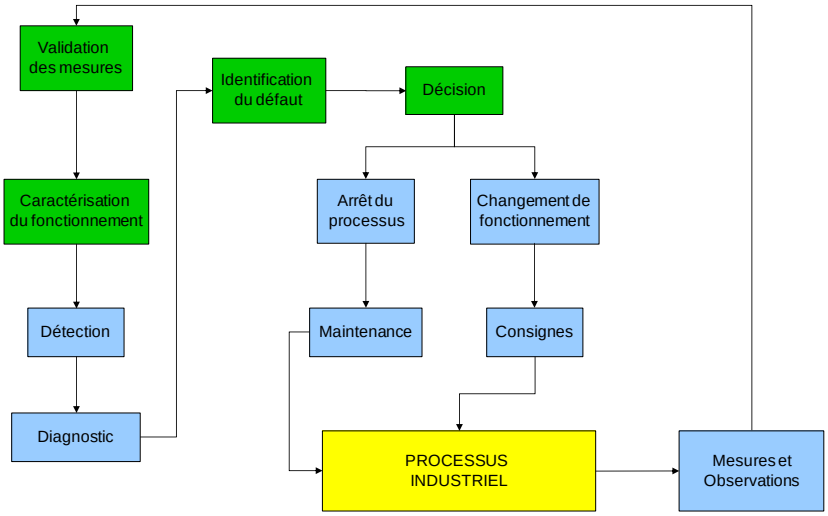


Figure I-7 – Schéma général de la surveillance en ligne des processus industriels (Zwingelstein, 1995)

La Figure I-7 propose un processus de surveillance des systèmes industriels. Ce processus dépend bien entendu de la nature du système et des objectifs de la surveillance. On peut néanmoins retenir que la surveillance en ligne passe par l'observation du système afin de détecter les défaillances de ses composants et de les diagnostiquer. La détection est une fonction qui permet d'identifier un fonctionnement anormal dans le comportement du système. Le diagnostic a pour objectif d'identifier la cause de la défaillance voire de mesurer son amplitude. Le traitement de la défaillance dépend souvent de la qualité du diagnostic. Dans les systèmes critiques, il faut toutefois envisager des traitements préliminaires comme la mise en repli dès l'étape de détection.

I.I.4 Conclusion

Dans cette partie nous avons introduit les principaux concepts liés à la sûreté de fonctionnement. Nous avons notamment vu que la tolérance aux fautes est l'un des moyens utilisé pour concevoir des systèmes critiques devant fonctionner en dépit de l'occurrence de fautes internes. L'objectif de ce travail est de proposer des nouveaux outils afin de contribuer à la conception de systèmes embarqués tolérants aux fautes. Ce travail s'intéressera plus particulièrement à la surveillance en ligne d'architectures redondantes. Notre objectif est d'accroitre la sûreté de fonctionnement de ces systèmes en permettant de détecter et diagnostiquer au plus tôt leurs défaillances.

I.II Les systèmes embarqués logiques

Ce travail concerne la conception d'architectures tolérantes aux fautes pour les systèmes embarqués. Mais nous ne prétendons pas proposer des solutions pour n'importe quel type de systèmes embarqués. Nous nous intéressons à des systèmes embarqués basés sur des composants sur étagères ou COTS (Commercial Off-The Shelf) numériques possédant de nombreuses entrées/sorties tout ou rien (micro-contrôleurs, FPGA, …). Ce sont des systèmes qui devront être capables de remplacer les fonctions d'automatismes à commande à relais comme celles que l'on trouve en aéronautique ou dans le ferroviaire. Nous avons choisi de définir ces systèmes par l'expression « systèmes embarqués logiques ». Dans cette partie, nous allons définir plus précisément cette classe de systèmes embarqués.

I.II.1 Systèmes embarqués

On parle de nos jours de plus en plus de systèmes embarqués. Mais ce terme recouvre deux réalités. Aujourd'hui on considère comme système embarqué soit des systèmes électroniques embarqués dans des applications mobiles, soit des systèmes électroniques programmables. Par exemple, un téléphone portable est considéré comme un système embarqué. De même, une « box » est considérée comme un système embarqué. Le concept de système embarqué est exprimé en anglais par « embedded system ». Une traduction mot à mot de cette expression anglaise nous donnerait en français « système enfoui ». Le terme enfoui est lié au fait que l'aspect électronique est enfoui sous des couches logicielles. En effet, de plus en plus de systèmes électroniques modernes sont conçus autour de puces numériques programmables (circuits intégrés) permettant d'implémenter des fonctions électroniques. L'aspect électronique et programmable n'est pas suffisant pour caractériser ce qu'est un système embarqué. Par exemple, les PC (Programmable Computers) répondent à cette définition mais ne sont pas des systèmes embarqués. En effet, les systèmes auxquels nous nous intéressons sont des systèmes autonomes dédiés à une tâche précise. Le concept d'autonomie introduit ici, signifie que ces systèmes ne disposent pas de périphériques comme ceux que l'on peut trouver habituellement autour d'un PC. Ils ne possèdent ni clavier, ni écran. Ils communiquent avec leurs environnements par l'intermédiaire d'entrées/sorties ou de bus de communication (bus CAN, SPI, RS232, …). L'autonomie se traduit également souvent par une faible consommation d'énergie.

Le succès des systèmes embarqués est fortement lié au développement des composants numériques. Depuis les années 70 et le développement des premiers microprocesseurs, le marché des microprocesseurs ne cesse de croitre. Cette croissance est en grande partie due à des évolutions technologiques permettant de mettre de plus en plus de transistors sur les surfaces réduites des microprocesseurs. La puissance des microprocesseurs double tous les 18 mois environs d'après la loi empirique de Moore (cf. Figure I-8). On est passé en 30 ans de quelques milliers de transistors à plusieurs milliards de transistors sur des puces de quelques cm^2 de surface. Cette évolution de la puissance, combinée à la capacité d'adressage d'espace mémoire de plus en plus important (passage de processeurs 4 bits à des processeurs 64 bits) a

entrainé une utilisation massive de ces composants numériques dans les applications. La conséquence étant des volumes croissants de composants fabriqués faisant baisser de manière sensible les coûts d'utilisation.

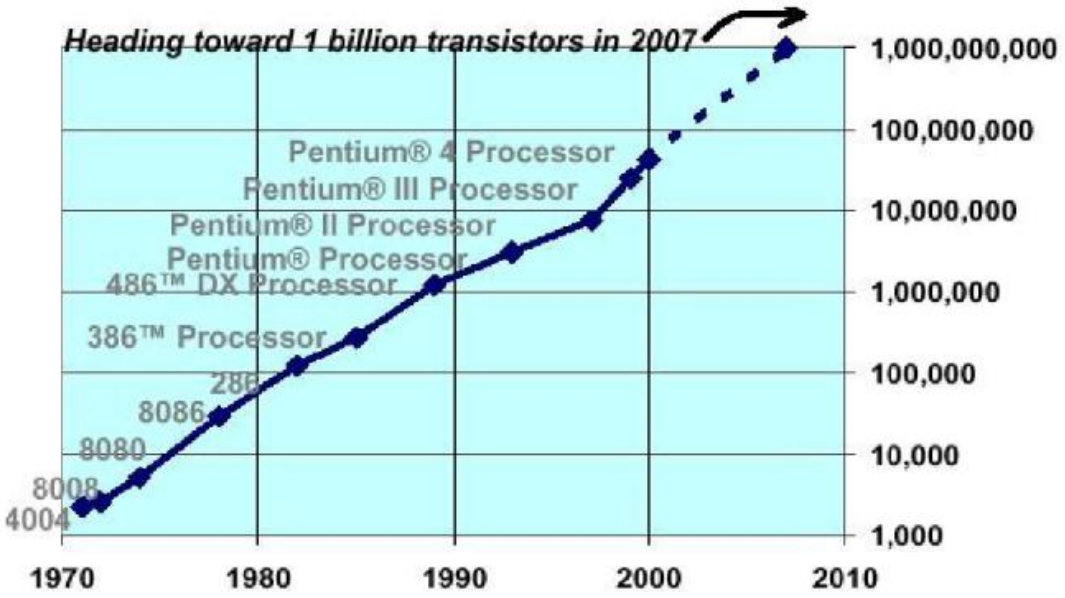


Figure I-8 – Evolution du nombre de transistors dans les processeurs Intel d'après la loi de Moore (Kadionik, 2005)

En 2004, il a été vendu 14 milliards de processeurs tout type confondu (microprocesseur, microcontrôleur, DSP) pour le marché de l'embarqué, contre seulement 260 millions de processeurs pour le marché du PC grand public (Kadionik, 2005). Ces chiffres montrent l'incroyable développement des systèmes embarqués dans les applications d'aujourd'hui.

Les composants numériques vendus appartiennent à différents types : microprocesseurs, microcontrôleurs, DSP, FPGA, ASIC.

Les microprocesseurs sont des composants numériques possédant une unité arithmétique et logique (UAL) permettant de faire des calculs numériques. Ils communiquent avec de nombreux composants périphériques (mémoires, entrées/sorties, …) par l'intermédiaire de bus (bus d'adressage, bus de données, bus de contrôle). Ils permettent de développer des applications en exploitant les fonctionnalités d'un système d'exploitation. Le système d'exploitation permet de simplifier le contrôle du matériel par des applications informatiques. Par contre, ils sont en général peu autonomes d'un point de vue énergétique.

Les microcontrôleurs sont des composants numériques plus intégrés que les microprocesseurs. En effet, ils sont dédiés à des applications pour lesquels la puissance de calcul est moins importante que les échanges entre le processeur et ses périphériques. Ils intègrent de ce fait de petites quantités de mémoire et des ports d'entrées/sorties leur permettant de communiquer avec leur environnement. Les bus de communication du processeur avec ses périphériques sont également intégrés dans la puce.

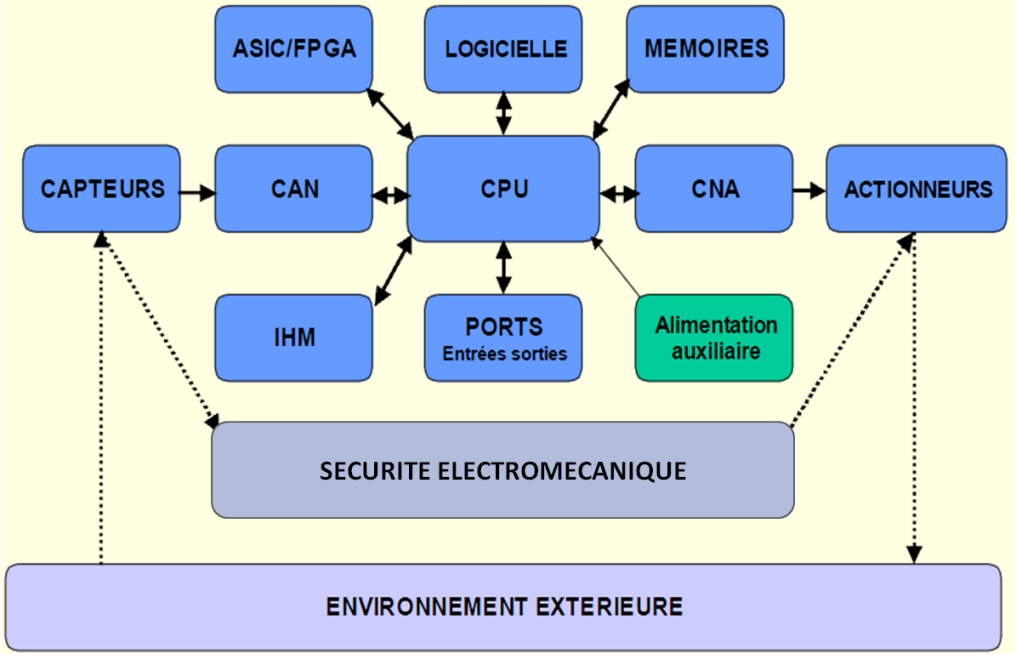


Figure I-9 – Structure typique d'un système embarqué (Grisel, et al., 2012)

Les DSP (Digital Signal Processor) sont des microprocesseurs spécialisés dans le traitement des signaux numériques. Ils ont un jeu d'instructions spécifiques qui tient compte des caractéristiques du calcul numérique. Par exemple, leur fonction principale permet de faire une multiplication suivie d'une addition et de stocker le résultat. Ce type de calcul est courant dans les algorithmes de filtrage développés pour le traitement du signal. Les DSP sont utilisés dans des systèmes comme les téléphones portables ou le GPS (Global Positionning System).

Le quatrième type de composants numériques utilisables dans les systèmes embarqués sont les composants logiques reprogrammables. Dans ce type, on distingue plusieurs sous-types comme les :

- FPGA (Field-programmable Gate Array, réseau de portes programmables in situ) ;
- PLD (programmable logic device, circuit logique programmable) ;
- EPLD (erasable programmable logic device, circuit logique programmable et effaçable) ;
- CPLD (complex programmable logic device, circuit logique programmable complexe) ;
- PAL (programmable array logic, réseau logique programmable) ;
- PLA (programmable logic array, réseau logique programmable).

Tous ces composants se caractérisent fonctionnellement par la possibilité de mettre en œuvre des fonctions logiques par programmation à l'aide d'outils de développement utilisant différents langages. Parmi les langages de programmation, il y a toujours les schémas blocs - diagrammes (Figure I-10). D'autre part, selon les fabricants, on dispose en général de langages proches de la programmation modulaire type VHDL (Figure I-11).

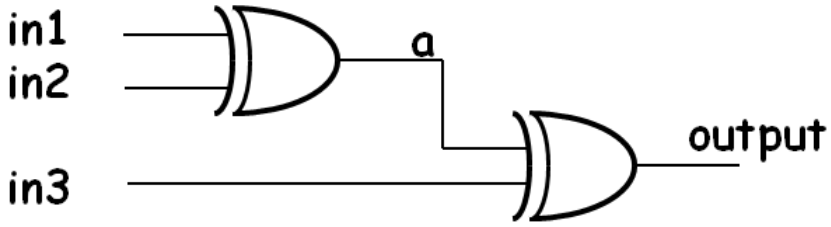


Figure I-10 – Exemple de schéma blocs-diagramme

```
ENTITY xor3 IS
        PORT ( in1, in2, in3 : IN BIT ;
                output : OUT BIT );
END xor3;

ARCHITECTURE archi OF xor3 IS
        SIGNAL a : BIT;
BEGIN
        a <= in1 XOR in2;
        out <= in3 XOR a ;
END archi;
```

Figure I-11 – Exemple de programmation d'une fonction logique en VHDL

Afin d'accélérer le développement d'application, pour chaque famille de composants, il existe des bibliothèques logicielles proposant des composants logiciels réutilisables que l'on peut intégrer dans son code. On les appelle des « Intellectual Property » ou IP, car ces COTS logiciels appartiennent en général à un fournisseur qui en possède les droits. Un développement de système embarqué est sensé les réutiliser sans aucune modification, car autrement, on perd deux qualités essentielles :

- La certification accompagnant l'IP qui garantit les fonctions pour lesquelles on en a fait l'acquisition,
- Le gain de temps apporté par l'IP par rapport à un développement spécifique voire son adaptation à un contexte applicatif particulier. Dans ce cas, on ne parle plus de COTS mais de MOTS (Modified Off-The-Shelf) (Clough, 2003).

Dans ce travail de thèse, nous considérons donc que les systèmes embarqués sont des systèmes électroniques programmables basés sur des composants numériques programmables.

I.II.2 Systèmes logiques et sécurité

Un système logique, est un système qui interagit avec son environnement par l'intermédiaire d'entrées/sorties tout ou rien. Dans la littérature scientifique on distingue deux types de systèmes logiques : les systèmes logiques combinatoires et les synchrones logiques séquentiels.

Les systèmes combinatoires sont des systèmes logiques qui étant donné une valeur du vecteur de leurs entrées calculent la même valeur pour leur vecteur de sorties. On a une relation combinatoire entre les sorties et les entrées. Formellement, chaque sortie est obtenue en réalisant une équation logique combinatoire avec des composantes du vecteur d'entrées. Toute équation logique peut être exprimée avec une famille d'opérateurs de base. La famille d'opérateurs de base la plus connue, est composée par les opérateurs $+$ (ou logique), . (et logique) et not (négation logique).

Les systèmes logiques séquentiels sont des systèmes, qui soumis à la même valeur de leur vecteur d'entrées, ne fournissent pas systématiquement la même valeur de sorties. On dit que ces systèmes ont une capacité de mémorisation de leur comportement passé. Pour formaliser cela, on introduit dans leur description le temps, et la notion d'état interne permettant cette mémorisation. Avec ces deux nouvelles variables on peut formaliser d'une manière générale le comportement des systèmes séquentiels asynchrones par les équations de Mealy (I-7) et (I-8).

Définition I-9 : Système séquentiel asynchrone

Soit $\sqcap = \{0,1\}$ *l'ensemble des valeurs logiques et* $t \in \mathbb{R}^+$*, la variable temps. Soit* $X \in \sqcap^n$*, le vecteur d'entrées d'un système,* $Y \in \sqcap^m$ *le vecteur des sorties, et* $Z \in \sqcap^q$ *son vecteur état interne. Le comportement du système est défini par les équations suivantes :*

$$Z_{t+\Delta t} = g(X_t, Z_t) \quad \textbf{(I-7)}$$

$$Y_t = f(X_t, Z_t) \quad \textbf{(I-8)}$$

La fonction g *est appelée équation de succession et l'équation* f *est appelée équation de sortie.*

Un système séquentiel asynchrone est un système qui recalcule son état interne et ses sorties à chaque modification d'une de ses entrées.

Un système séquentiel synchrone est un système logique dont le fonctionnement est synchronisé par une horloge. Il recalcule son état interne et ses sorties de manières synchrone avec la périodicité de son horloge. Cela amène à discrétiser le temps.

La synthèse d'un système séquentiel synchrone est en général réalisée à l'aide de portes logiques séquentielles qui sont des bascules (bascule JK, bascule D, bascule T). La synthèse d'un système asynchrone peut être réalisée avec des bascules SR (set/reset) ou par exemple de manière programmée par des calculateurs interprétant des langages comme le grafcet (David, et al., 1992). Pour les applications complexes à grand nombre d'entrées/sorties, on utilise souvent comme calculateur des automates programmables industriels (API) qui peuvent être programmés à l'aide des langages de la norme CEI 61131-3 (Sequential Functional Chart – SFC, langage à échelles de contacts, blocs fonctions diagrammes, listes d'instructions, texte structuré).

Malheureusement, malgré leur robustesse comparée par exemple à celle des PC, les automates programmables ne sont pas encore assez fiables pour des applications embarqués critiques exigeant un haut niveau de sécurité. Par exemple, dans les avions ou dans les trains, on préfère dans ce cas mettre en œuvre les automatismes de commande à l'aide de commande à relais qui ont prouvé leur fiabilité au fil des années.

Dans ce travail nous avons cherché à contribuer au transfert des fonctions d'automatismes des systèmes de transport critiques dans des composants numériques tout en garantissant le niveau de sécurité exigé par les normes en vigueurs. Le Comité Electronique International (CEI) a proposé différentes normes pour la sûreté de fonctionnement des systèmes industriels (Figure I-12).

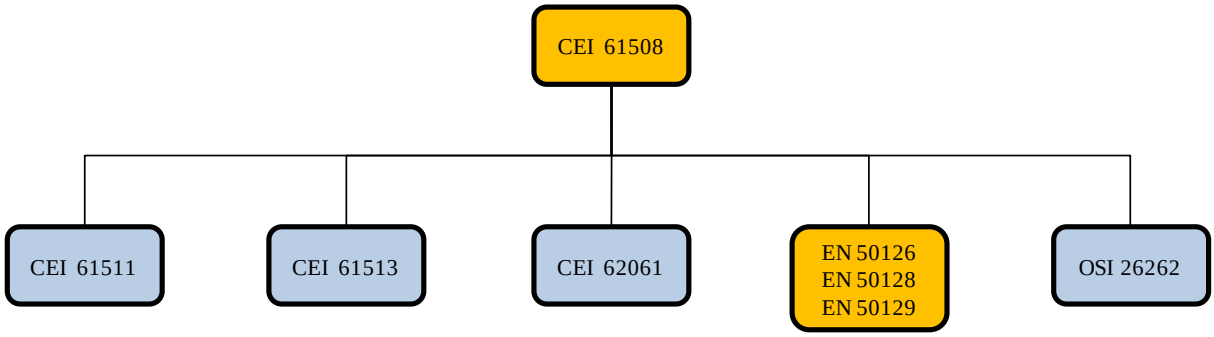


Figure I-12 – Normes de sûreté de fonctionnement pour les systèmes industriels

La norme CEI 61508 est une norme générale pour les systèmes industriels. Elle a été spécialisée en plusieurs normes filles en fonction des domaines d'applications. La norme CEI 62061 est une norme adaptée à la sûreté de fonctionnement des machines industrielles. La norme OSI 26262 est une adaptation de la CEI 61508, à la sûreté de fonctionnement dans l'automobile. Dans les systèmes ferroviaires, la CENELEC (organisme de normalisation Européen) a proposé trois normes filles :

- La EN 50126 qui est relative à la « Spécification et démonstration de la fiabilité, de la disponibilité, de la maintenabilité et de la sécurité » (CENELEC, 2000),
- La EN 50129 qui est relative aux « Systèmes de signalisation, de télécommunication et de traitement. Systèmes électroniques de sécurité pour la signalisation » (CENELEC, 2003),
- La EN 50128 qui est relative aux « Systèmes de Signalisation, de télécommunication et traitements - Logiciels pour les systèmes de commande et de protection ferroviaire » (CENELEC, 2001).

Pour être validé pour une utilisation par l'industrie automobile ou l'industrie ferroviaire, les systèmes logiques conçus doivent respecter ces normes. Or, les systèmes à base de composants numériques, ne sont pas considérés actuellement comme suffisamment sûrs pour commander des systèmes critiques comme un train d'atterrissage d'avion voire le système de freinage d'un train. En effet, les taux de défaillances des composants numériques standards sont en dessous des niveaux de sécurité exigés par ce type de système. L'objectif de ce travail, est de contribuer à rendre possible l'utilisation de ces composants standards malgré leur

qualité afin de pouvoir développer des systèmes embarqués permettant de faire un bon compromis entre coût et sécurité.

I.II.3 Systèmes temps réel

On fait souvent la confusion entre systèmes embarqués et systèmes temps réel. Tous les systèmes embarqués ne sont pas des systèmes temps réel. On peut effectivement comprendre que le système de calcul de la raideur à appliquer aux roues arrière d'une voiture pour amortir le choc subit par les roues avant à une vitesse de 130km/h doit être un système réagissant très vite en fonction de ses entrées. On est ici dans un cadre de temps réel dur. Par contre, une cafetière peut être considérée dans une certaine mesure comme un système embarqué sans pour autant être contraint à cette notion de temps réel dur. Les systèmes qui nous intéressent sont des systèmes embarqués temps réel dur.

Il n'y a pas de définition normalisée des systèmes temps réel. Nous allons donc prendre la définition suivante :

Définition I-10 : Système temps réel (Stankovic, 1988)

En informatique temps réel, le comportement correct d'un système dépend, non seulement des résultats logiques des traitements, mais aussi du temps auquel les résultats sont produits.

De cette définition, il vient qu'un système temps réel n'est pas forcément un système qui va vite, mais un système qui répond à des contraintes temporelles. Il doit pour cela garantir un déterminisme logique et temporel. Il doit également être fiable.

On distingue trois types de temps réel :

- Le temps réel dur : C'est un système pour lequel un retard dans la délivrance du résultat rend le système inutile. C'est par exemple le cas du contrôle d'un missile.
- Le temps réel mou : C'est un système pour lequel un retard dans la délivrance du résultat n'est pas catastrophique. C'est par exemple le cas du système d'ouverture d'une barrière automatique.
- Le temps réel ferme : C'est un système pour lequel on peut tolérer quelques retards (peu souvent) dans la délivrance du résultat. C'est par exemple le cas d'un système multimédia type téléphone mobile.

Dans ce travail, nous nous intéressons plus particulièrement à des systèmes temps réel dur.

I.II.4 Conclusion

Dans cette section, nous avons cherché à caractériser le type de systèmes concernés par ces travaux de recherche. Il s'agit de systèmes embarqués logiques caractérisés par un temps réel dur. Ces systèmes sont un type de systèmes électroniques programmables. Ils sont construits autour de composants de type microcontrôleur ou FPGA qui sont des COTS. L'intérêt de ces COTS est que ce sont des composants standards d'utilisation générale et donc à coût réduit. Ces composants doivent permettre le contrôle de systèmes critiques aussi bien d'un point de

vue sécurité que d'un point de vue contrainte temporelle à respecter. Les niveaux de sécurité exigés nécessitent de mettre en œuvre une ingénierie adéquate garante de l'obtention d'un code exempt de fautes internes. Leur caractère standard impose également d'être capable de construire des architectures matérielles tolérantes aux fautes. Cela passe par la conception d'architectures redondantes et par le diagnostic en ligne de leurs fautes.

I.III Problématique du diagnostic des systèmes embarqués logiques

Dans cette partie nous allons poser la problématique du diagnostic des systèmes embarqués logiques. Nous allons donc nous intéresser à l'existant en matière de diagnostic selon les trois points de vue que l'on peut avoir sur un tel système. Nous ferons une présentation synthétique du diagnostic des systèmes logiques, du diagnostic des composants programmables et du diagnostic des systèmes embarqués afin d'en cerner les limites. Cela nous permettra d'identifier les besoins pour les systèmes embarqués logiques.

I.III.1 Diagnostic des systèmes logiques

Le diagnostic des systèmes logiques a été initié dans les années 80 par un certain nombre d'auteurs pour créer la théorie logique. Nous en rappelons ici les bases qui ont été ensuite reprises pour le diagnostic des systèmes embarqués.

I.III.1.A Approche de R. Davis (Davis, 1984)

Cette approche propose d'abord un modèle de connaissance pour la modélisation des systèmes logiques. Dans le cadre de la description structurelle d'un système électronique, Davis a introduit trois concepts de base : les modules, les ports et les terminaux (Figure I-13).

Les modules représentent les composants du système. Selon le niveau d'abstraction considéré, ils modélisent des cartes électroniques ou des chips.

Les ports et les terminaux permettent de modéliser la connectivité des entités du système. Les ports modélisent les liaisons entre l'extérieur et l'intérieur d'un module. De ce fait, tout module comporte au moins deux ports. Les terminaux modélisent les liaisons intermodules d'un niveau donné. Récursivement, tout module peut être modélisé par la même approche. On crée donc une hiérarchie de modules.

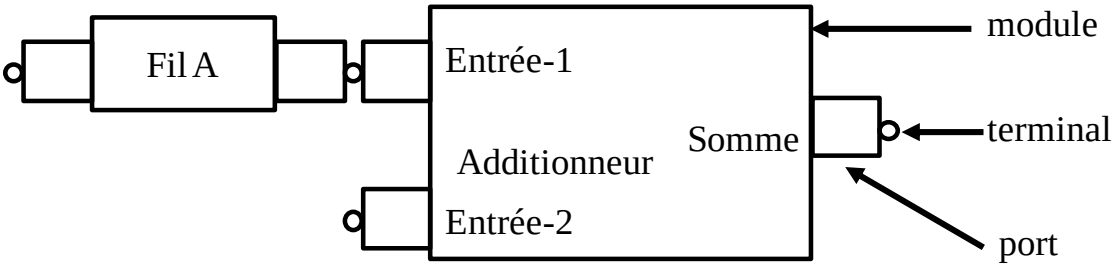


Figure I-13 – Principaux termes utilisés dans la description de la structure d'un système

Davis a développé en LISP, un langage permettant de générer une structure de données sur la base des composants et de la connectique spécifiée par un modèle graphique. Cette structure de données est utilisée pour le diagnostic.

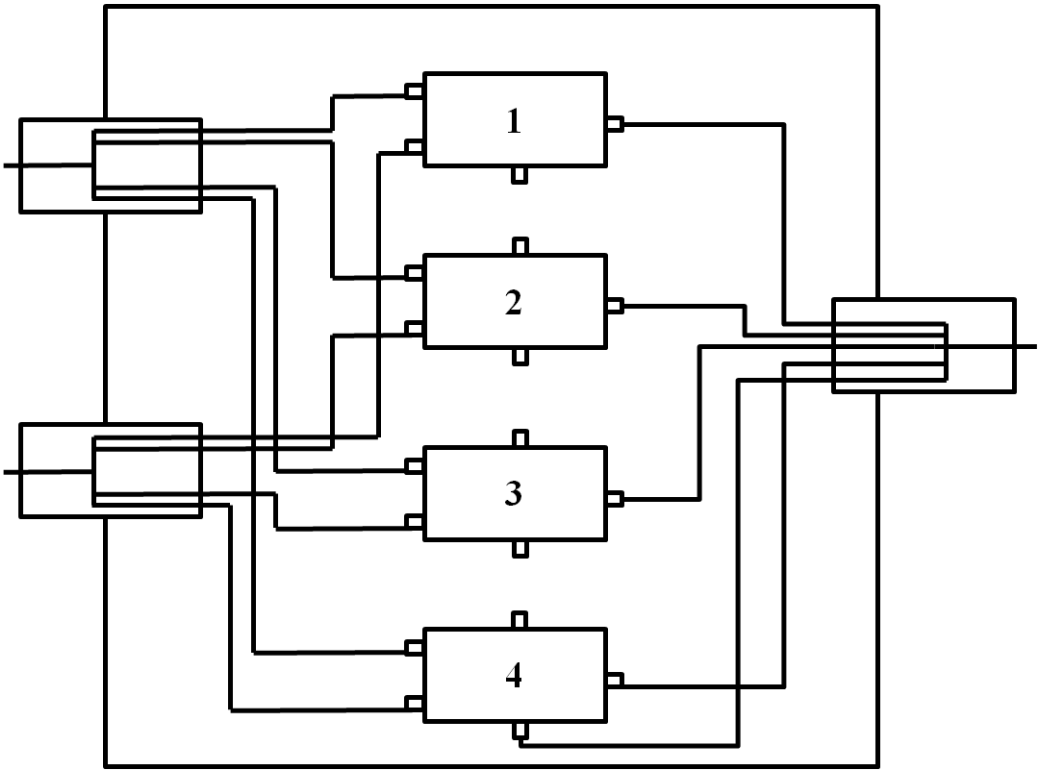


Figure I-14 – Modélisation structurelle d'un additionneur 4 bits

La seconde étape des travaux de Davis concerne la modélisation du comportement des constituants du modèle structurel. Le formalisme utilisé permet d'exprimer les relations entre les entrées et les sorties d'un module sous forme de règles ou contraintes. L'auteur distingue deux types de règles :

- Les règles exprimant le flux électrique au travers d'un module.

Si nous considérons l'exemple de la Figure I-13, nous aurons la contrainte suivante par rapport aux entrées $\text{Entrée} - 1$**,** $\text{Entrée} - 2$ et la sortie Somme **:**

$$\text{to get Somme from } (\text{Entrée} - 1\ \text{Entrée} - 2) \text{ do } (+\ \text{Entrée} - 1\ \text{Entrée} - 2)$$

- Les règles exprimant le flux inférentiel.

$$to\ get\ \text{Entrée} - 1\ from\ (Somme\ \text{Entrée} - 2)\ do\ (-Somme\ \text{Entrée} - 2)$$

$$to\ get\ \text{Entrée} - 2\ from\ (Somme\ \text{Entrée} - 1)\ do\ (-Somme\ \text{Entrée} - 1)$$

En s'appuyant sur le modèle de connaissance précédent, l'auteur a proposé une approche pour le diagnostic des défaillances des systèmes logiques. Le raisonnement est initié en comparant

les valeurs des sorties observées d'un module par rapport aux valeurs simulées, à partir de celles des entrées du système. Il a pour fondement le principe de violation de contraintes. Ce principe est le suivant : si le comportement observé d'un module ne correspond pas à celui attendu, alors soit le module est défaillant, soit il est plongé dans un environnement anormal (i.e. ses entrés sont erronées). Afin d'illustrer la démarche nous considérons la carte de la Figure I-15

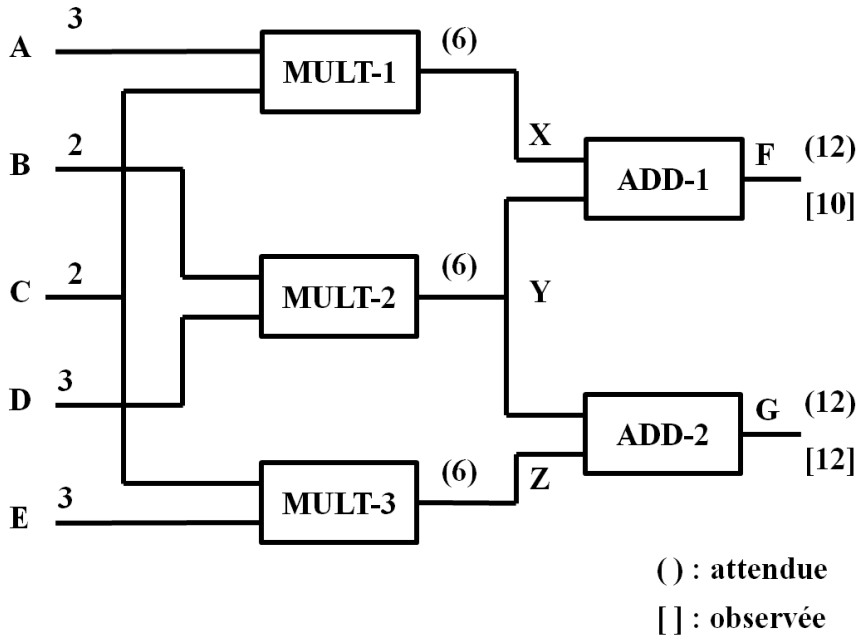


Figure I-15 – Exemple de système pour le diagnostic par la théorie logique

La carte électronique de cet exemple est constitué de trois multiplicateurs ($MULT-1, MULT-2, MULT-3$) et de deux additionneurs ($ADD-1$ et $ADD-2$).

L'observation des variables de sorties (F et G) nous montre, une différence entre la valeur réelle de F qui est 10 et sa valeur attendue 12. D'après le principe de violation des contraintes, trois hypothèses sont envisageables :

- $ADD-1$ est défaillant,
- La variable d'entrée de $ADD-1$, X est fausse,
- La variable d'entrée de $ADD-1$, Y est fausse.

Le principe de la résolution est de n'écarter aucune hypothèse. L'utilisation des règles du flux inférentiel permet de calculer les valeurs des variables internes dans un contexte d'hypothèses précises. Par exemple, si l'on suppose que $ADD-1$ n'est pas défaillant et que la valeur de X est bonne (X=6).

Les règles du flux inférentiel permettent alors d'estimer que Y=4. Cette valeur, injectée dans la règle exprimant la contrainte de fonctionnement de ADD-2 nous donnerait que G=10. Cela introduit une nouvelle violation de contrainte par rapport à G (puisque G=12 est la valeur

observée). On peut répéter le raisonnement tenu par rapport à F. Ainsi si on continue dans cette voie, on aboutit à la conclusion que les ensembles de composants défaillants possibles sont {MULT-2, ADD-2} et {MULT-2, MULT-3}.

Le mécanisme de résolution utilisée par Davis est donc l'analyse systématique de toutes les hypothèses issues de la violation d'une contrainte de fonctionnement. Aucune hypothèse n'est éliminée par des critères probabilistes ou statistiques. Dans l'exemple précédent, on aboutit ainsi à donner plusieurs solutions pour expliquer le comportement défaillant de la carte. Ces solutions sont les ensembles de composants suivants :

{ADD-1},{MULT-1},{MULT-2,ADD-2},{MULT-2, MULT-3}.

Nous utiliserons cet exemple pour présenter les autres approches de la théorie logique.

I.III.1.B Approche de M.R. Genesereth (Genesereth, 1984)

Elle est basée sur l'utilisation de données de conception (issues d'une CAO), pour diagnostiquer automatiquement les systèmes ou processus. Elle a conduit au développement du programme DART. Ce programme est sensé assurer le diagnostic en coopération avec un programme de test automatique. L'originalité des travaux de Geneserth se situe dans l'utilisation de la logique de premier ordre lors de la modélisation des systèmes.

On distingue deux phases dans la modélisation : la modélisation pour la conception du système et la modélisation pour le diagnostic du système.

En conception, la modélisation concerne deux axes : la description structurelle et la description du comportement.

Au niveau structurel, chaque composant est typé à l'aide d'un prédicat logique. Si on se réfère au circuit électronique de la Figure I-15, on peut distinguer les connaissances suivantes :

- **Connaissances structurelles relatives aux composants**

  SD1 : (MULT M1)
  SD2 : (MULT M2)
  SD3 : (MULT M3)
  SD4 : (ADD A1)
  SD5 : (ADD A2)

- **Connaissances structurelles relatives aux liens inter-composants.**

  SD6 : (CONN (IN 1 F1) (IN 1 M1))
  SD7 : (CONN (IN 3 F1) (IN 2 M1))
  SD8 : (CONN (IN 2 F1) (IN 1 M2))

SD9 : (CONN (IN 4 F1) (IN 2 M2))
SD10 : (CONN (IN 3 F1) (IN 1 M3))
SD11 : (CONN (IN 5 F1) (IN 2 M3))
SD12 : (CONN (OUT M1) (IN 1 A1))
SD13 : (CONN (OUT M2) (IN 2 A1))
SD14 : (CONN (OUT M2) (IN 1 A2))
SD15 : (CONN (OUT M3) (IN 2 A2))

F1 représente la carte électronique primaire. Une connaissance comme SD6 s'interprète de la manière suivante : l'entrée 1 de la carte F1 (IN 1 F1) est connectée (CONN) à l'entrée 1 du multiplicateur M1 (IN 1 M1).

Le comportement du système et de chacun de ses composants est également décrit par des règles utilisant des prédicats du langage. Là aussi, Genesereth distingue deux types de connaissances :

- Les connaissances modélisant le comportement du système tel que défini dans le cahier des charges (Behavioural Data).

Ces connaissances font abstraction de la structure interne du système.

Exemple : Circuit F1 avec A=3, B=2, C=2, D=3, E=3

BD1 : (IF (AND (VAL (IN 1 F1) t 3) (VAL (IN 2 F1) t 2) (VAL (IN 3 F1) t 2) (VAL (IN 4 F1) t 3) (VAL (IN 5 F1) t 3)) (AND (VAL (OUT 1 F1) (VAL (OUT 2 F1) t 12))

La règle BD1 donne les valeurs de sorties de F1 en fonction de celles de ses entrées.

- Les connaissances relatives à des composants de base vérifiant les lois de la physique (Theorical Data).

Par exemple le comportement théorique d'une porte logique AND pourrait être modélisé par la règle suivante :

TH1 : (IF (AND (ANDG d) (VAL (IN 1 d) t ON) (VAL (IN 2 d) t ON)) (VAL (OUT 1 d) t ON))

Le ANDG est une porte G de type AND Gate.

Pour le diagnostic, les données issues de la conception doivent être enrichies par des connaissances de deux nouveaux types :

- Les connaissances relatives à la description de l'environnement du système (Achieval and Observable Data).

ACi : (VAL (IN i F1) t x)

La variable "x" représente la valeur de l'entrée « i » de F1 à la date t.

OBj : (VAL (OUT j F1) t y)

La variable « y » représente la valeur de la sortie « j » de F1 à la date t.

- Les connaissances relatives aux hypothèses de défaillance (ou de non défaillance) introduites par l'utilisateur (Diagnostic Assumption).

DA1 : (IF (NOT (ADD A1)) (AND (ADD A2) (MULT M1) (MULT M2) (MULT M3)))

La règle précédente traduit l'hypothèse de défaillance de l'un des composants du système (NOT(ADD A1)).

Le diagnostic est activé par la donnée à une date d'observations ne correspondant pas à celles attendues (observations négatives). Ces observations sont alors interprétées comme des symptômes. A partir d'une inférence en chaînage arrière, l'ensemble des composants suspectés est établi.

Le fonctionnement de DART repose sur l'hypothèse de défaillance unique. Aussi, si l'ensemble des suspects est réduit à un composant, le diagnostic est terminé. Dans le cas contraire, DART prend en compte les observations positives fournies en même temps que les symptômes pour réduire l'ensemble de suspects. Ensuite, il tente de générer une série de tests permettant la discrimination du composant défaillant. Si la génération est réussie, la procédure de diagnostic est appliquée itérativement à partir des observations produites par les tests. Sinon, DART conclue en donnant l'ensemble minimal des composants suspectés.

L'intérêt de l'approche proposé par Genesereth se situe au niveau de l'utilisation de la logique pour modéliser les systèmes dans le cadre du diagnostic automatique. Toutefois, on peut douter de l'efficacité de DART notamment pour le diagnostic des systèmes complexes autre que les circuits électroniques. Cela est sensible, notamment dans la modélisation du comportement normal du système à diagnostiquer (Behavioural Data). Malgré l'utilisation de prédicats logiques, on a le sentiment qu'on peut être confronté à une combinatoire importante dans le cadre de la modélisation de comportements complexes. Par ailleurs, l'hypothèse de défaillance unique reste assez restrictive. En effet, que vaut le programme DART lorsqu'en cours de génération de tests discriminatoires, une nouvelle défaillance survient ?

Ces critiques trouvent leur réponse dans l'approche proposée par Reiter (Reiter, 1987)

I.III.1.C Approche de J. de Kleer (de Kleer, et al., 1980)(de Kleer, et al., 1984)(de Kleer, et al., 1987)(Froger, 1990)

Les premiers travaux de Johan de Kleer date de 1976 avec l'approche « Violated Expectation ». Cette approche concernait uniquement le diagnostic des systèmes digitaux. Elle avait pour fondement l'hypothèse de défaillance unique. En 1987, de Kleer proposa l'approche « General Diagnosis Engine » (GDE)(de Kleer, et al., 1987) qui généralise ses travaux antérieurs.

Comme son nom l'indique, GDE se pose comme approche universelle pour diagnostiquer tout type de systèmes ou processus. Elle conçoit le diagnostic comme une tâche itérative. Son but est d'identifier la défaillance à partir des mesures courantes, effectuées sur le processus. Si l'identification échoue la tâche de diagnostic doit déterminer les mesures additionnelles à effectuer.

Dans cette approche, le diagnostic utilise également un modèle explicite du procédé qui est définit par la donnée de :

- La description de sa structure physique,
- Le modèle de chacun de ses constituants.

Dans la conception de de Kleer, la notion de constituant est plus large que celle généralement admise. En fait, elle intègre à la fois des aspects structurels et fonctionnels du système. Ainsi, l'ensemble de constituants peut inclure les composants physiques de système, les processus et même les étapes d'une inférence logique. Cette notion est donc un concept central pour comprendre le formalisme GDE.

Exemple : Le système à diagnostiquer est de nouveau la carte électronique de la Figure I-15. Les constituants du circuit sont : M1, M2, M3, A1 et A2.

Le comportement de chaque composant est donné par son équation logique :

$M1 : X = A * C$

$M2 : Y = B * D$

$M3 : Z = C * E$

$A1 : F = X + Y$

$A2 : G = Y + Z$

Le diagnostic prend en compte l'hypothèse de défaillances multiples. Elle est initiée par les écarts détectés entre les variables mesurées (appelées Observations ou OBS) et leur prédiction. Il s'appuie sur trois concepts de base :

- L'environnement (ENV)

Il désigne un ensemble de composants (donné entre accolades {}) supposés non défaillants et utilisés pour prédire la valeur d'une variable.

{A1, M1, M2} est l'environnement utilisé pour calculer F.

- Les conflits (< >).

Un conflit est un ensemble de constituants dont l'un au moins est défaillant (disjonction).

Les conflits sont détectés en lançant une inférence C sur la base des observations OBS et de l'environnement associé ENV (C(OBS, ENV)).

Dans l'exemple de la carte électronique, C({F = 10}, {A1, M1, M2}), aboutit à une incohérence ($F_{mesuré}$ = 10 et $F_{prédit}$ = 12). Les éléments de l'environnement constituent donc un conflit noté < A1, M1, M2>.

- Candidats ([ ]).

Un candidat est un ensemble de constituants défaillants (conjonction).

Les candidats correspondent à la conjonction des conflits minimaux. Dans l'exemple précédent, le rejet des valeurs mesurées de F et G conduit respectivement aux conflits C1 = <A1, M1, M2> et C2 = <A1, A2, M1, M3>.

Le diagnostic est donné par l'ensemble des candidats minimaux : [A1], [M1], [A2, M2], [M2, M3].

Le mécanisme de résolution de GDE est basé sur une coopération avec un ATMS (de Kleer, et al., 1987) (Assumptiom based Truth Maintenance System). Ce système est chargé du maintient de la vérité des hypothèses émises par GDE dans le cadre de la génération de tests optimaux pour l'enrichissement de l'ensemble OBS.

I.III.1.D Conclusion

Ces approches sont souvent limitées à l'hypothèse de défaillance unique. Dans tous les cas, elles sont hors ligne.

I.III.2 Diagnostic des composants programmables

Le comportement des composants numériques peut être faussé par deux catégories de fautes : des fautes internes et des fautes externes.

Les fautes internes sont dues aux défaillances des constituants internes à ces composants : les ports d'entrées/sorties, les blocs des fonctions internes et les blocs de connexions internes permettant de relier les constituants précédents (Figure I-16).

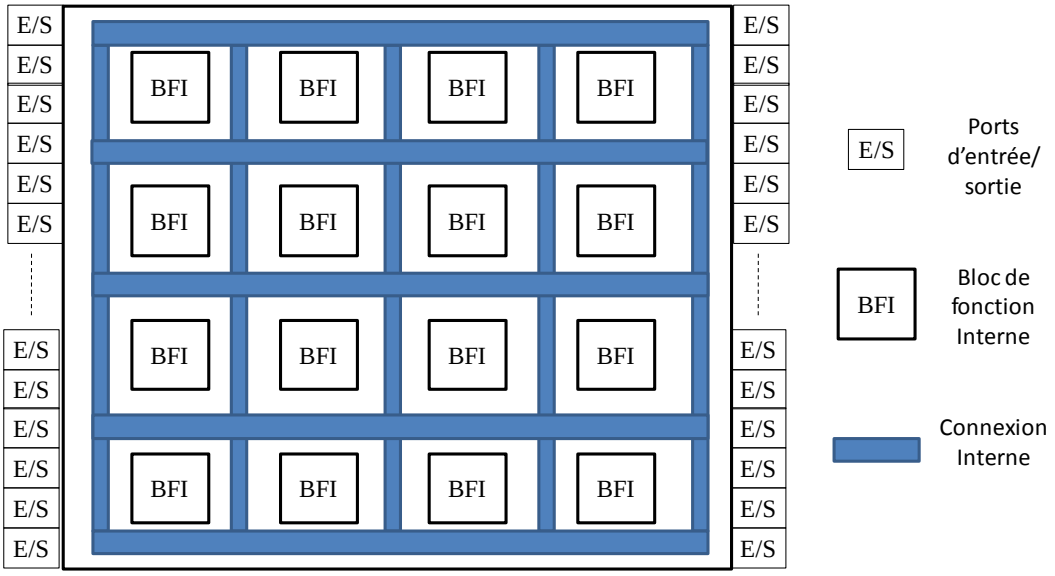


Figure I-16 – Schéma de principe de la structure d'un composant numérique

Les fautes externes sont surtout dues aux impacts de particules parasites émises dans l'atmosphère. En effet, l'augmentation du nombre de transistors sur des surfaces toujours aussi réduite s'est accompagnée d'une finesse de plus en plus importe de ces derniers, et d'une diminution du voltage de leur alimentation. Il y a quelques années, c'était surtout dans l'espace que les composants numériques étaient sensibles aux particules émises par le soleil, entrainant des reconfigurations parasites ou des fautes intermittentes. Avec la miniaturisation accrue des transistors, aujourd'hui leur comportement est perturbé même au sol.

Pour éviter les fautes internes permanentes, les fabricants de composants numériques procèdent à des tests systématiques basés sur le schéma de la Figure I-17. Le principe de ces tests est le suivant. Il consiste à définir différents vecteurs de tests qui sont injectés sur les ports d'entrées du composant. On mesure alors les valeurs des ports de sorties que l'on compare aux valeurs attendues compte-tenu de la configuration ou de la programmation du composant numérique.

L'analyse des résultats des différents tests permet de diagnostiquer les constituants des composants logiques qui sont défaillants.

Cette méthode de tests avec générateur de vecteurs de tests et analyseur des résultats peut également être utilisée en exploitation. Dans ce cas, il faut être capable de concevoir un système permettant d'intégrer les fonctions de génération des vecteurs de test et d'analyseur des résultats. Selon la catégorie du composant numérique, il faut embarquer soit uniquement de la mémoire additionnelle, soit des composants supplémentaires implémentant ces fonctions. Si le composant numérique considéré est reconfigurable, il est juste nécessaire de

stocker en mémoire, dans le cas de configurations différentes, celles pour les autotests et celle relative au fonctionnement normal.

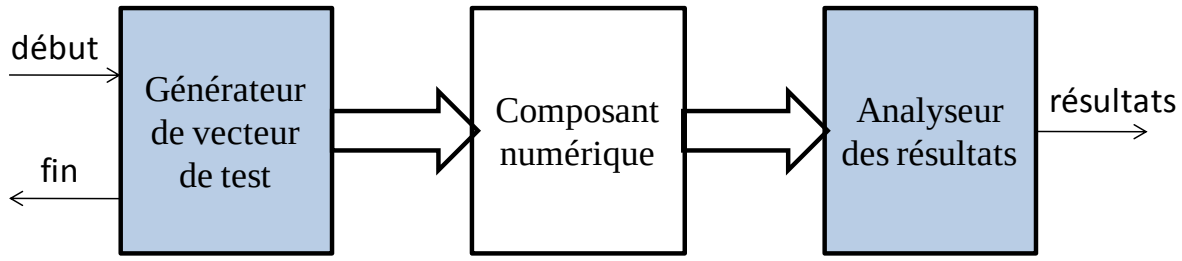


Figure I-17 – Schéma de principe du test des composants numériques

Dans ce travail, nous nous sommes surtout intéressés aux systèmes embarqués conçus à l'aide de composant FPGA. Nous allons donc illustrer cette méthode de diagnostic des fautes internes par tests du composant avant son exploitation. Il existe 3 types de FPGA :

- Les FPGA SRAM (Static RAM) : Ils sont volatiles (ce qui implique qu'après chaque coupure d'alimentation, la mémoire est perdue et à chaque démarrage du système équipé d'un FPGA SRAM, ce composant doit d'abord charger son bitstream avant de pouvoir fonctionner) et reconfigurables. Ils sont les moins chers et sur le plan fonctionnel leur principal avantage est qu'ils sont reconfigurables même en ligne. Leur inconvénient principal c'est leur volatilité qui pose problème pour les applications critiques.
- Les FPGA FLASH EEPROM : Ils sont non volatiles et reconfigurables. Plus chers que les FPGA SRAM, ils corrigent leur défaut du point de vue volatilité. Par contre, ils ne sont reconfigurables que hors ligne.
- Les FPGA Anti-Fusible : ils sont non volatiles et non reconfigurables. De ce fait, ils sont très intéressants pour les applications critiques.

Il existe différents types d'architectures chez les différents constructeurs. Elles peuvent être hiérarchique, matricielle, …

L'architecture d'un FPGA peut avoir une influence sur la technique de diagnostic utilisée pour l'identification de ses défaillances. L'architecture la plus classique est celle matricielle (Figure I-18).

De manière schématique, un FPGA SRAM est structuré en interne par trois types d'entités : les blocs logiques programmables (CLB), les boites de routage (SB) et les boite de connexion (CB). A sa périphérie, il possède des ports qui sont configurables en entrées ou en sorties pour que l'environnement puisse communiquer avec le FPGA. Les blocs logiques programmables sont à la base du fonctionnement du FPGA. C'est eux qui permettent de mettent en œuvre les

fonctions logiques et numériques implémentées dans un FPGA. Ils sont eux même constitués de 3 composants principaux (Figure I-19) : la LUT (LookUp Table), des multiplexeurs et des bascules D. La LUT est son principal composant. En effet, elle permet d'implémenter n'importe quelle fonction logique grâce d'une part à ses bits de configurations (entrées CPi) et aux valeurs données aux entrées k1. Quant aux entrées k2, elles permettent le contrôle direct des multiplexeurs. La fonction de sortie est donnée par les sorties k4.

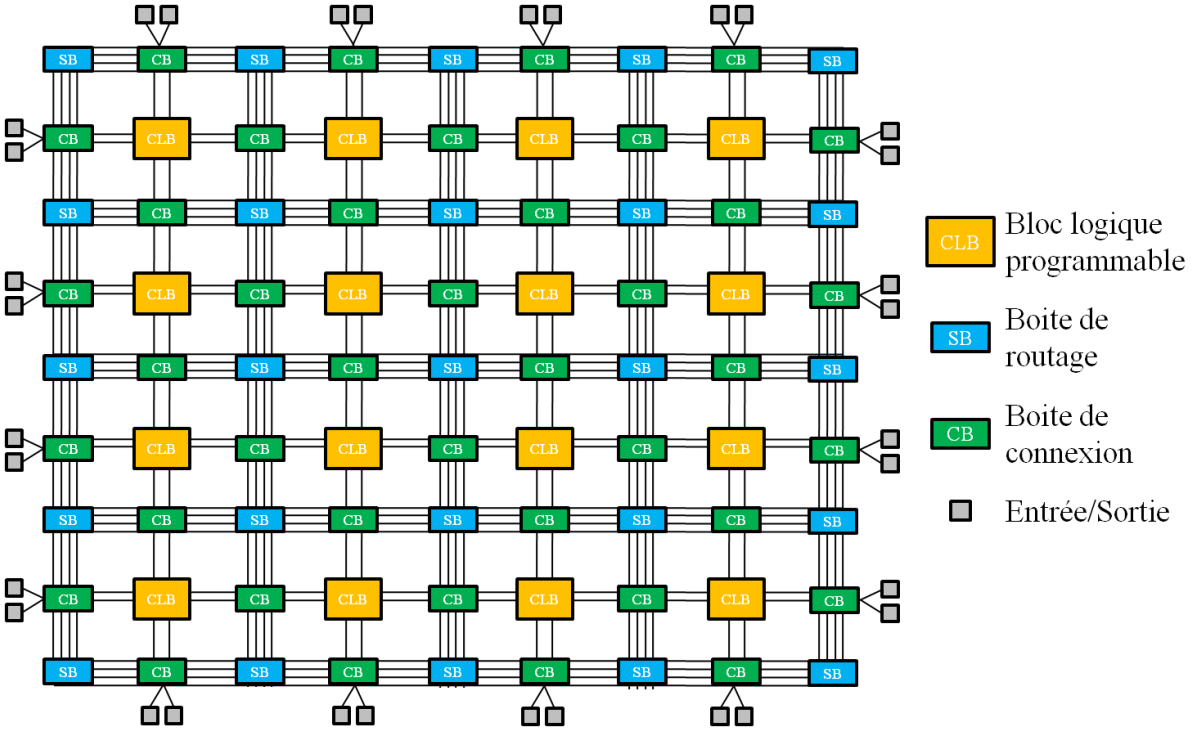


Figure I-18 – Architecture matricielle d'un FPGA

Les boites de connexion (CB) et les boites de routage (SB) constituent les éléments d'interconnexion de la structure interne du FPGA. Les CB permettent notamment de connecter les ports et les CLB. Les boites de routage permettent de relier les CB et donc indirectement les entités connectés par ces derniers. La configuration des CB et des SB s'effectue également à travers des bits de configuration.

C'est la reprogrammation des bits de configuration des éléments d'interconnexion et des CLB qui permet de reconfigurer le FPGA.

On distingue classiquement 3 catégories de méthodes pour le diagnostic des FPGA (Clermidy, 1999) (Manfoumbi, et al., 2003)(Figure I-20) :

- Les Built In Self Tests (BIST) : Ce sont des méthodes qui exploitent la reconfigurabilité des FPGA SRAM. Ces méthodes sont les seules à pouvoir être appliquées en ligne. Elles ne permettent pas de diagnostiquer toutes les pannes des composants d'un FPGA.
- Les Design For Testability (DFT) méthodes : Elles ont pour but de concevoir des

FPGA dont l'architecture est adaptée aux tests.

- Les tests IDqqt : Contrairement aux autres méthodes, ce n'est pas un test logique du composant. Cette méthode utilise la mesure du courant au niveau de la masse pour détecter des fautes du type collage permanent d'un transistor à l'état passant. L'inconvénient de la méthode est qu'elle nécessite des choix judicieux des vecteurs des tests qui dépendent de l'architecture du composant.

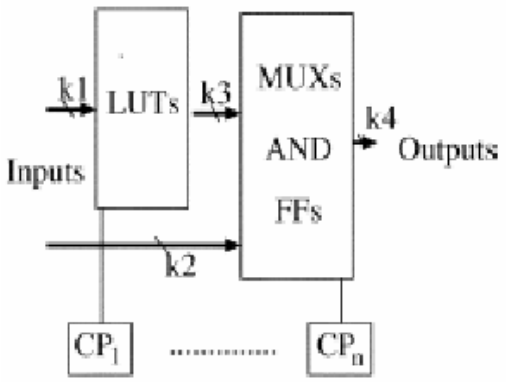


Figure I-19 – Structure d'un bloc logique programmable (Manfoumbi, et al., 2003)

Les méthodes de tests permettent en général d'identifier les composants défaillants par nature. Certaines méthodes sont plutôt adaptées au diagnostic des CLB d'un FPGA. C'est par exemple le cas des méthodes BIST ou Test And/OR. Dans ce cas on fait en général l'hypothèse que les autres composants sont fonctionnellement bons.

La majorité de ces méthodes s'appliquent hors ligne pour vérifier que le composant n'a pas de constituant fautif. Les méthodes BIST peuvent toutefois s'appliquer en ligne pour tester l'état du système avant une phase d'exploitation. Lorsque le FPGA est reconfigurable, elles ne nécessitent pas l'ajout de composants numériques pour le test. La reconfiguration permet de partitionner les CLB en trois groupes de constituants : les CLB utilisés pour la génération des vecteurs de tests, les CLB à tester et les CLB configurés pour analyser les résultats du test. Ainsi lors d'une session, un certain nombre de CLB sont testés. Ensuite, on intervertit les rôles des CLB testés et celui des CLB ayant servi soit à la génération des tests soit à l'analyse des résultats. Ainsi on peut en deux sessions d'autotests, tester l'intégralité des CLB d'un FPGA sous l'hypothèse du bon fonctionnement des interconnexions (Tomoo, et al., 1998). Certaines méthodes permettent même de détecter toute faute du CLB en une session et de localiser le CLB fautif en 4 sessions au plus (Stroud, et al., 1996)(Stroud, et al., 1996)(Stroud, et al., 1997).

Pour les composants numériques non-reconfigurables, il existe peu de travaux pour l'identification des constituants fautifs. Les approches proposées dans la littérature sont en général dépendantes du fonctionnel implémenté dans le composant. Elles sont basées sur l'association à chaque bloc séquentiels du programme d'une signature permettant de tracer le l'exécution du programme (concepts de Data Flow Graph et de Control Flow Graph dans

(Antola, et al., 2002). La signature caractéristique de chaque bloc est émise à l'entrée et à la sortie du bloc vers un composant tiers chargé de la vérification de la conformité. Cette approche permet de détecter voire de localiser le défaut sous réserve de pouvoir associer à chaque bloc sa localisation dans le composant.

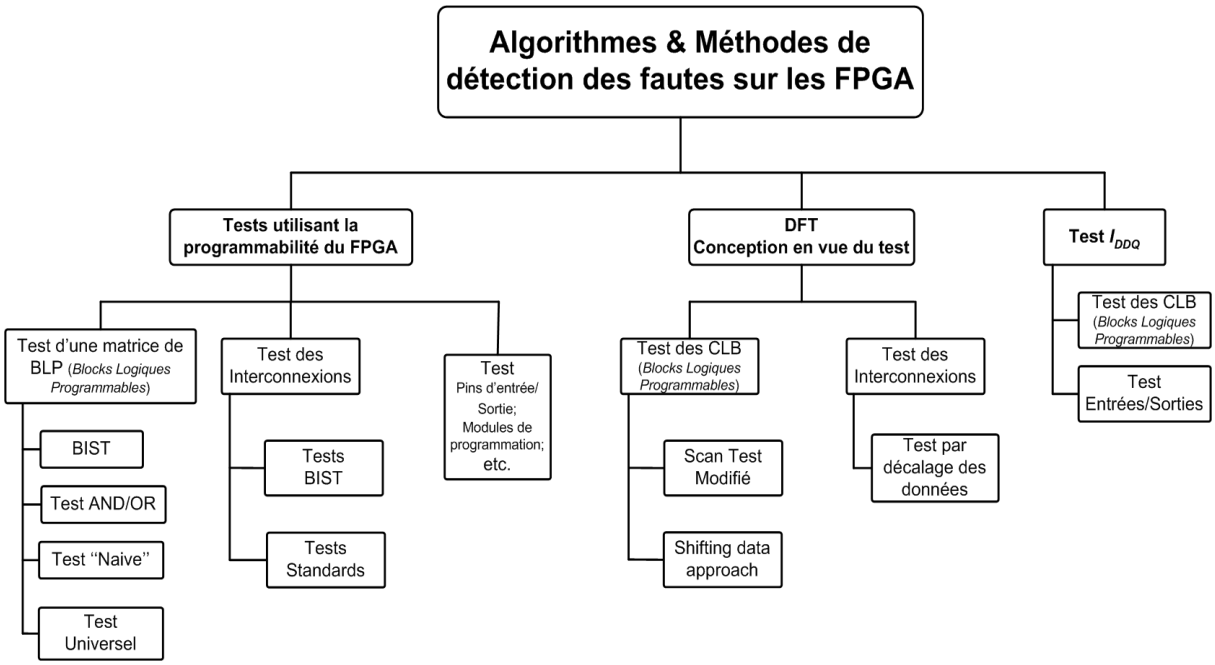


Figure I-20 – Méthodes de détection des fautes sur les FPGA (Manfoumbi, et al., 2003)

I.III.3 Diagnostic des systèmes embarqués

Une des problématiques majeures du diagnostic des systèmes embarqués est liée au fait que ces systèmes sont obtenus par intégration de différents sous-systèmes. Ainsi, en cas de faute, il n'est pas aisé d'identifier le sous-système responsable de la faute. Le diagnostic est d'autant plus complexe qu'un système embarqué n'est pas homogène, car il est constitué d'une partie matérielle et d'une partie logicielle. Parmi les approches développées, on peut citer le diagnostic à base de modèles et le diagnostic par analyse de trace - Spectrum-based Fault Localization – (van Gemund, et al., 2004) (Zoeteweij, et al., 2008).

I.III.3.A.a Approches de diagnostic à base de modèle

Le diagnostic à base des modèles a été initié par R. Reiter (Reiter, 1987) et J. de Kleer (de Kleer, et al., 1987) dans le cadre de la théorie logique (cf. § I.III.1). Il est adapté au diagnostic de la partie matérielle du système embarqué. De manière très schématique, il consiste à comparer les observations d'un modèle de référence avec celles du système réel (Figure I-21) pour identifier les ensembles de composants minimaux pouvant expliquer les différences. Bien entendu, le modèle de référence est supposé exempt d'erreurs et complet (Hypothèses très fortes). Donc toute différence dans les observations résulte forcément d'un dysfonctionnement dans le comportement du système réel.

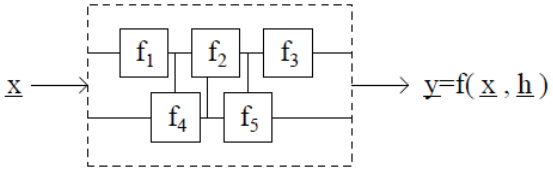


Figure I-21 – Vue conceptuelle d'un système (van Gemund, et al., 2004)

Dans le problème de base, on dispose du vecteur d'entrées et des vecteurs de sorties du système réel et du modèle du système. La problématique du diagnostic consiste à déterminer les sous-ensembles de composants dont les fautes peuvent expliquer le comportement défaillant du système réel. Dans ce cadre, l'identification est souvent incertaine car le diagnostic ne dispose que de la connaissance de la structure du système (relations d'interactions entre les composants) et de son comportement normal pour faire cette identification. Ce cadre de base extrêmement contraignant peut être relâché de différentes manières. La première méthode consiste à enrichir le modèle du système en y introduisant des modèles des fautes. La limite de cette méthode est la complétude. On n'est pas sûr d'introduire dans le système tous les modèles de fautes et donc il y aura des difficultés à diagnostiquer les fautes imprévues. La deuxième méthode consiste à accéder à des observations internes des entrées/sorties de certains composants. Si on a accès à l'observation de toutes les entrées/sorties de tous les composants, le problème devient trivial et s'apparente à du test. En pratique, dans les systèmes complexes, des contraintes techniques ne permettent pas en général d'avoir cette observabilité totale. Dans les chapitres ultérieurs nous verrons que ce problème d'observabilité suffisante est une problématique majeure qui se traduira notamment dans la notion de diagnosticabilité. La troisième méthode consiste à multiplier les comparaisons entre les observations de différents vecteurs d'entrées avec les différences entre les différents vecteurs de sorties du système et du modèle. Par analogie avec le diagnostic des composants numériques, pour chaque comparaison, nous parlerons de version de diagnostic. Avec l'hypothèse qu'il n'y a pas de nouvelles fautes durant l'établissement des différentes versions, le diagnostic peut être affiné en faisant l'intersection des ensembles des composants donnés par chaque version de diagnostic.

Dans le cas de systèmes logiques, le modèle du système peut être spécifié sous la forme d'équations logiques intégrant des variables modélisant l'état de chaque composant du système. Dans la Figure I-21, h est le vecteur modélisant l'état de chaque composant. Pour x et y donnés, le diagnostic revient donc à déterminer $h = f^{-1}(x, y)$. Si $h_i = 0$ alors le composant i est normal, sinon il est défaillant.

I.III.3.A.b Approche de diagnostic basée sur l'analyse de traces (méthode SFL)

Cette approche est comparable à l'analyse structurelle en automatique (Staroswiecki, et al., 1996)(Staroswiecki, et al., 2000). Elle est adaptée au diagnostic de la partie logicielle d'un système embarqué. Elle est utilisée pour la recherche automatique de bogue dans les logiciels complexes (Très nombreuses lignes de code). Elle est basée sur le constat qu'il est difficile

d'établir un modèle de comportement pour la composante logicielle. La difficulté est liée à la fois à l'hétérogénéité des composants logiciels mais surtout à l'explosion combinatoire que peut engendrer la construction d'un modèle de comportement pour l'ensemble du logiciel. La méthode consiste d'abord en la définition de différents blocs de codes qui constituent les composants dont on veut diagnostiquer l'état. Ensuite, l'injection de différents vecteurs de tests en entrée du système logiciel et l'analyse des résultats obtenus en sorties comparés aux résultats attendus permet de déterminer si chaque test est normal ou défaillant. On établit alors une matrice qui associe à chaque test les composants utilisés. On réalise également un vecteur d'erreur qui indique pour chaque test s'il est défaillant ou s'il est normal. Sur la base de ses données, il suffit de calculer pour chaque composant le ratio entre le nombre de tests défaillants rapportés aux nombre de tests totaux dans lequel il est utilisé. Les composants logiciels défaillants les plus probables sont ceux qui ont les ratios les plus importants.

Dans l'exemple de la Figure I-22, les blocs 1, 2, 3 et 4 sont les composants dont on veut diagnostiquer l'état. Le programme considéré dans l'exemple est l'application du tri à bulles à des nombres rationnels. La colonne « input », correspond aux différents tableaux de nombres rationnels utilisés en entrée du programme pour le tester. Le vecteur d'entrées $<\frac{3}{1},\frac{2}{2},\frac{4}{3},\frac{1}{4}>$ a donné en sortie $<\frac{1}{1},\frac{2}{2},\frac{3}{3},\frac{4}{4}>$ au lieu de la valeur attendue qui est $<\frac{1}{4},\frac{2}{2},\frac{4}{3},\frac{3}{1}>$. On note donc une erreur pour ce test dans le vecteur d'erreur. Le calcul du ratio de participation -ou coefficient de similarité- (Abreu, et al., 2006) de chaque composant à un test erroné noté S_j indique que le composant défaillant le plus probable est le bloc 4. On peut voir sur le programme qu'il correspond bien au composant fautif dans la mesure où le code ne trie en fait que les numérateurs.

```
void RationalSort(int n, int *num,
                  int *den){
  /* block 1 */
  int i,j,temp;
  for ( i=n-1; i>=0; i-- ) {
    /* block 2 */
    for ( j=0; j<i; j++ ) {
      /* block 3 */
      if (RationalGT(num[j], den[j],
            num[j+1], den[j+1])) {
        /* block 4 */
        temp = num[j];
        num[j] = num[j+1];
        num[j+1] = temp;
}}}}
```

input	block 1	2	3	4	error
$\langle\ \rangle$	1	0	0	0	0
$\langle\frac{1}{4}\rangle$	1	1	0	0	0
$\langle\frac{2}{1},\frac{1}{1}\rangle$	1	1	1	1	0
$\langle\frac{4}{1},\frac{2}{2},\frac{0}{1}\rangle$	1	1	1	1	0
$\langle\frac{3}{1},\frac{2}{2},\frac{4}{3},\frac{1}{4}\rangle$	1	1	1	1	1
$\langle\frac{1}{4},\frac{1}{3},\frac{1}{2},\frac{1}{1}\rangle$	1	1	1	0	0
$s_j =$	$\frac{1}{6}$	$\frac{1}{5}$	$\frac{1}{4}$	$\frac{1}{3}$	

Figure I-22 – Exemple illustratif de la méthode d'analyse des traces (van Gemund, et al., 2004) (Abreu, 2009)

Conclusion du chapitre I

Comme nous l'avons vu dans cette partie, il est nécessaire de diagnostiquer les fautes des systèmes embarqués logiques. En effet, ces systèmes sont souvent utilisés pour le contrôle de procédés critiques comme le système de freinage d'un train. Il faut donc pouvoir

diagnostiquer leur état, soit pour envisager une maintenance de l'équipement, soit pour poursuivre une mission dans le cadre d'une architecture tolérante aux fautes.

Les méthodes de diagnostic des composants numériques permettent d'identifier les parties internes à ces composants à l'origine d'une de leurs fautes. Ces méthodes ont surtout été développées pour le test des composants en sortie des chaines de fabrication. Les méthodes de diagnostic sont peu nombreuses et permettent souvent de ne diagnostiquer que certains constituants du composant. En général, elles nécessitent leur application hors-ligne. Sinon, en exploitation, elles doivent être intégrées lors de la réalisation du fonctionnel et donc être modifiée chaque fois que ce dernier est changé.

L'objectif de notre travail est de proposer une méthode de diagnostic des systèmes embarqués logiques (SEL) permettant l'utilisation d'outils de diagnostic fiables. Ce type de systèmes embarqués constitue une classe de système à événements discrets (SED) et de systèmes temporisés. Notre objectif dans ce travail est donc d'adapter les approches développées pour le diagnostic de ces classes de systèmes. L'une des difficultés posée par les SEL et le grand nombre d'entrées/sorties pouvant entrainer une explosion combinatoire lors de la création des modèles de diagnostic. Notre objectif dans la suite de ce mémoire, est de proposer une méthode permettant un diagnostic efficace tout en combattant cette explosion combinatoire.

Notre travail se situant dans le cadre des SED, la méthode que nous allons développer dans la suite de ce mémoire s'applique et concerne le diagnostic des défaillances cataleptiques. En effet, comme nous l'avons vu, les défaillances cataleptiques sont assimilables à des événements ce que ne sont pas les défaillances par dégradation. Celles-ci ne pourront être diagnostiquées que dans l'hypothèse où elles sont complètes. L'approche que nous voulons proposer ne pourra donc pas éviter que l'entité source de la défaillance n'atteigne un état de panne. Notre objectif est d'éviter que la panne se propage à l'ensemble du système qui la contient. Cela passe par un diagnostic rapide afin de permettre le confinement de la défaillance, voire la reconfiguration du système pour une poursuite de sa mission en attendant la réparation ou le remplacement de l'entité en panne. Dans ce travail, nous supposerons que les défaillances sont permanentes et cohérentes. Donc la disparition d'une défaillance ne pourra se faire qu'après traitement de cette défaillance.

II Outils et méthodes pour le diagnostic des SEL

Introduction du chapitre II

Dans le chapitre précédent, nous avons introduit la notion de système embarqué logique (SEL). Ce type de système est une classe de systèmes discrets (Systèmes à Evénements Discrets ou SED, et Systèmes Temporisés ou ST). Dans ce chapitre, nous nous intéressons aux outils et méthodes pour le diagnostic des SEL. Nous présenterons dans un premier temps quelques outils formels utilisés pour modéliser le comportement des systèmes discrets. Nous commençons ainsi par présenter les automates à états finis et les automates temporisés. Ensuite. Nous présentons les Réseaux de Petri (RdP) temporisés et les RdP temporels. Dans un second temps, nous présentons des outils pour le diagnostic des systèmes discrets. Nous introduisons d'abord le concept d'observateur discret à partir duquel nous présenterons les notions de diagnostiqueur et vérificateur. Nous terminons ce chapitre par la présentation des Signatures Temporelles Causales (STC) ou chroniques, une approche pour le diagnostic de systèmes discrets.

II.I Outils formels pour la modélisation du comportement des systèmes discrets

Dans cette partie, nous introduisons les outils nécessaires à la modélisation du comportement des systèmes discrets afin de pouvoir développer une approche de surveillance basée sur les modèles. Nous détaillons particulièrement les modèles automates à états finis et les réseaux de Petri et leurs extensions permettant une manipulation explicite du temps.

II.I.1 Automates à états finis et langages

Un automate à états finis est un quintuplé $G = (Q, q_0, \Sigma, \delta, Q_m)$ avec :

- *Q est un ensemble finis d'états ;*
- *q_0 est l'état initial avec $q_0 \in Q$;*
- *Σ est un ensemble d'événements encore appelé alphabet fini ;*
- *δ est la fonction de transition définie de $Q \otimes \Sigma \rightarrow Q$;*
- *Q_m est un ensemble d'états finaux également appelés états marqués ou accepteurs tels que $Q_m \subseteq Q$;*

On définit la fonction Γ: $Q \rightarrow 2^{\Sigma}$ qui fait correspondre à chaque état ses événements « actifs ». Si pour chaque état $q \in Q$, $\Gamma(q) = \Sigma$, alors G est dit complet.

On note $\varepsilon \in \Sigma$, l'événement vide qui est un événement non observable.

Dans la suite de ce paragraphe, nous donnons quelques définitions et nous présentons quelques opérations sur les automates.

Définition II-1 : Automate déterministe

Un automate G est dit déterministe s'il n'existe pas de transition sur l'événement vide (ε) et si $\forall q \in Q, \forall \sigma \in \Sigma\backslash\{\varepsilon\}, \delta(q,\sigma)=q1$ et $\delta(q,\sigma)=q2 \Rightarrow q1=q2$.

Remarque : Un automate qui contient des transitions sur l'événement ε est un automate non déterministe car on ne sait pas dans quel état on est (cf. G2 sur Figure II-1).

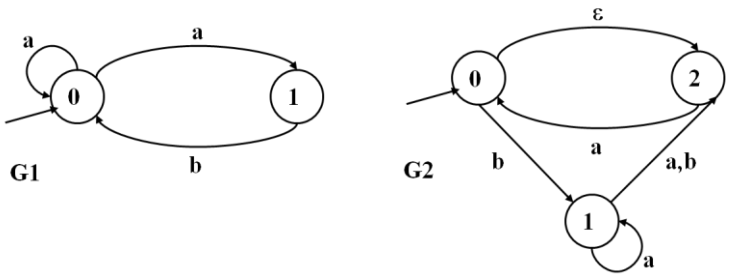


Figure II-1 – Exemple d'automates non déterministes

Soit $s \in \Sigma^*$ avec Σ^* l'ensemble des séquences que l'on peut construire sur l'alphabet Σ. Cela signifie qu'il existe $\sigma_i \in \Sigma$ avec $i \in [1..n]$ / $s = \sigma_1 \sigma_2 \ldots \sigma_{i \ldots} \sigma_n$.

On peut alors étendre la fonction de transition δ, aux séquences de Σ^*. En effet, on a la relation $\delta(q,s) = \delta(q, \sigma_1 \sigma_2 \ldots \sigma_n) = \delta(\delta(q, \sigma_1 \sigma_2 \ldots \sigma_{n-1}), \sigma_n)$.

Cela permet d'utiliser la théorie des langages pour décrire le comportement d'un automate (Ramadge, et al., 1987a)(Ramadge, et al., 1987b).

Définition II-2 : Langage généré et langage accepté

Le langage généré par l'automate $G = (Q, q_0, \Sigma, \delta, Q_m)$ est l'ensemble des séquences de Σ^* générées par G à partir de son état initial. $L(G) = \{s \in \Sigma^* / \exists q \in Q, \delta(q_0,s)=q\}$.

Le langage accepté par l'automate $G = (Q, q_0, \Sigma, \delta, Q_m)$ est l'ensemble des séquences de L(G) qui permettent d'atteindre un état marqué à partir de l'état initial. $L_m(G)=\{s \in L(G) / \delta(q_0,s) \in Q_m\}$.

Définition II-3 : Accessibilité

Un état $q \in Q$ est dit accessible s'il existe une séquence $s \in \Sigma^*$ tel que $\delta(q_0,s) = q$.

Un automate à états finis G est dit accessible si tous ses états sont accessibles.

On adoptera par la suite les notations suivantes :

L'ensemble des états accessibles à partir de q par la séquence s sera noté $\Delta_G(q,s) = \{q' \in Q / \delta(q,s) = q'\}$.

L'ensemble des suffixes de s dans L(G) sera noté $L(G)/s=\{t \in \Sigma^* / s.t \in L(G)\}$.

Définition II-4 : Vivacité

Le langage L(G) généré par un automate à états finis G est dit vivant si à partir de tout état accessible, on peut toujours franchir une transition ($\forall q \in Q / \exists s \in \Sigma^*, \delta(q_0,s) = q$ alors $\Gamma(q) \neq \phi$).

Exemple : L'automate G_1 de la Figure II-1 est tel que Γ(0)={a} et Γ(1)={b}. Donc G_1 est vivant.

Définition II-5 : Projection et projection inverse

Soit $\Sigma=\Sigma^1 \cup \Sigma^2$, on définit la projection P_i de $\Sigma^* \rightarrow (\Sigma^i)^*$ par :

$$\begin{cases} P_i(\varepsilon)=\varepsilon \\ P_i(\sigma)=\sigma \text{ si } \sigma \in \Sigma^i \text{ sinon } P_i(\sigma)=\varepsilon \\ P_i(s.\sigma)=P_i(s).P_i(\sigma) \text{ pour } s \in \Sigma^* \text{ et } \sigma \in \Sigma \end{cases}$$

De même on définit la projection inverse P_i^{-1} de $(\Sigma^i)^* \rightarrow \Sigma^*$ par :

$P_i^{-1}(t)=\{s \in \Sigma^* / P_i(s)=t\}$

Remarque : En pratique, la projection P_i consiste à effacer d'une séquence s tous les événements n'appartenant pas à Σ^i.

Définition II-6 : Composition d'automates

La composition d'automates consiste à créer un automate à partir de la composition de deux automates indépendants définis sur 2 alphabets Σ^1 et Σ^2.

Si $\Sigma^1 \cap \Sigma^2 = \emptyset$, alors le résultat de la composition correspond au produit cartésien des deux automates.

Si $\Sigma^1 \cap \Sigma^2 \neq \emptyset$, soit G la composition synchrone de G1 et G2 (on note $G = G1 ||_s G2$) tel que $Gi=(Q^i, q^i_0, \Sigma^i, \delta^i, Q^i_m)$ avec i=1,2.

$G = (Q^1 \otimes Q^2, (q^1_0, q^2_0), \Sigma^1 \cup \Sigma^2, \delta, Q^1_m \otimes Q^2_m)$ et $\Gamma^i(q^i_j)= \{\sigma \in \Sigma^i / \delta^i(q^i_j, \sigma) \in Q^i$ et $\delta^i(q^i_j, \sigma)!)$ avec :

- $\delta((q^1,q^2), \sigma) = (\delta^1(q^1,\sigma), \delta^2(q^2,\sigma))$ si $\sigma \in \Gamma^1(q^1) \cap \Gamma^2(q^2)$
- $\delta((q^1,q^2), \sigma) = (\delta^1(q^1,\sigma), q^2)$ si $\sigma \in \Gamma^1(q^1) \setminus \Sigma^2$
- $\delta((q^1,q^2), \sigma) = (q^1, \delta^2(q^2,\sigma))$ si $\sigma \in \Gamma^2(q^2) \setminus \Sigma^1$
- *indéfini sinon*

Le modèle de comportement associé aux deux composants est obtenu par composition synchrone sur les événements communs (cf. Figure II-2).

Remarque : $L(G)=L(G1)\cup L(G2)$ et $L_m(G)=L_m(G1)\cup L_m(G2)$

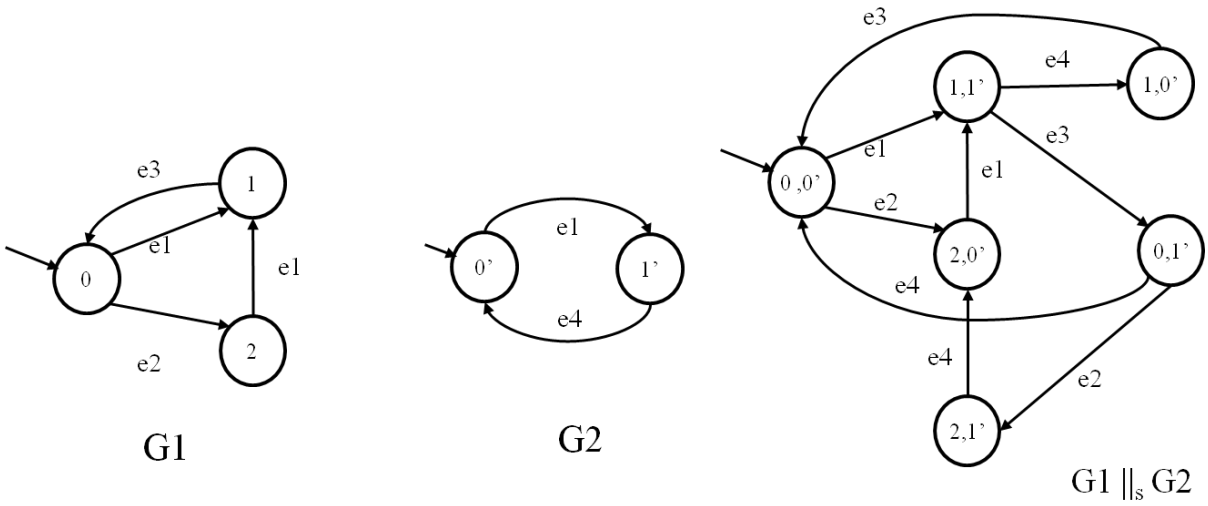


Figure II-2 – Composition synchrone de deux automates

On s'intéresse maintenant aux systèmes à événements discrets partiellement observables. Ces systèmes sont caractérisés par deux catégories d'événements (Figure II-3) : l'ensemble des événements observables Σ_o et l'ensemble des événements inobservables Σ_{uo}. On a bien entendu $\Sigma=\Sigma_o\cup\Sigma_{uo}$.

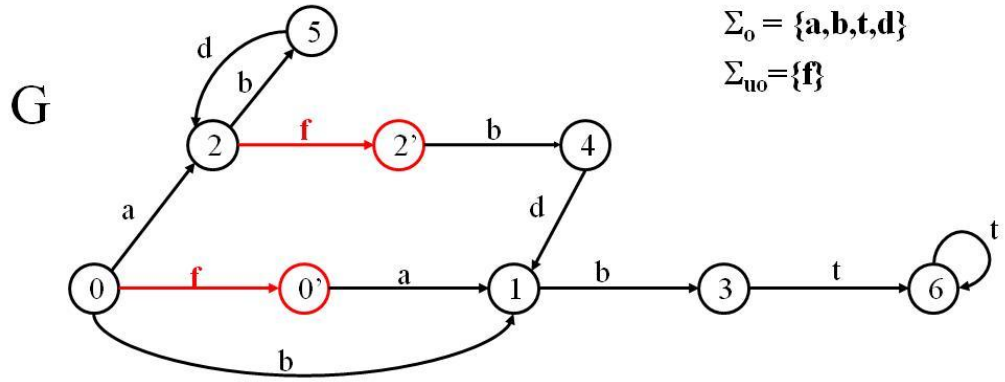


Figure II-3 – Exemple de SED partiellement observable (Jéron, et al., 2006)

On notera $\mathbf{P_o}$ la projection définit de $\Sigma^*\rightarrow\Sigma_o^*$ qui est telle que $\forall\ L\subseteq\Sigma^*$, $P_o(L)=\{P_o(s)\in\Sigma_o^*\ /\ s\in L\}$.

Définition II-7 : Trace ou langage observable

La trace d'un automate à états finis G est définie par trace(G)=$P_o(L(G))$. Elle représente l'ensemble des séquences observables de G.

On définit l'ε-clôture : une opération consistant à effacer d'un automate les événements inobservables.

Définition II-8 : ε-clôture

Etant donné un automate à états finis $G=(Q, q_0, \Sigma, \delta, Q_m)$ avec $\Sigma=\Sigma_o \cup \Sigma_{uo}$, l'ε-clôture de G est l'automate $\varepsilon(G) =(Q^\varepsilon, q_0, \Sigma_o, \delta^\varepsilon, Q^\varepsilon_m)$, où $Q^\varepsilon \subseteq Q$ et $\delta^\varepsilon \subseteq Q \otimes \Sigma_o \otimes Q$ sont définis par :

- $q_o \in Q^\varepsilon$
- *si* $q \in Q^\varepsilon$, $q' \in Q$, $s \in (\Sigma_{uo})^*$, $\sigma \in \Sigma_o$, $(q,s.\sigma)=q'$ *alors* $q' \in Q^\varepsilon$ *et* $\delta^\varepsilon(q, \sigma)=q'$

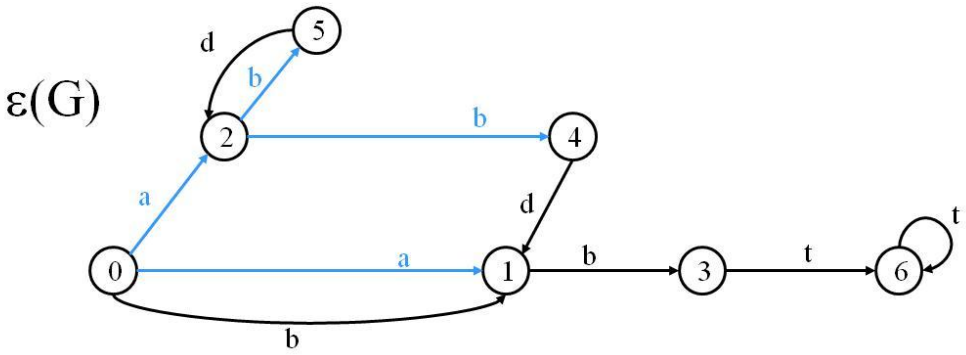


Figure II-4 – Automate ε(G) correspondant au système de la Figure II-3.

Remarque : L'automate ε(G) est en général non déterministe. On peut noter sur l'exemple de la Figure II-4 qu'au niveau de l'état '0' on a deux transitions dépendantes du même événement 'a' (δ(0,a)=1 et δ(0,a)=2). Donc ε(G) est non déterministe.

On définit à présent la déterminisation qui a pour objectif de construire un automate déterministe à partir d'un automate non déterministe. En effet, il a été montré que tout automate à états finis non déterministe peut être transformé en un automate à états finis déterministe (Cassandras, et al., 2008).

Définition II-9 : Déterminisation

Etant donné un automate à états finis $G = (Q, q_0, \Sigma, \delta, Q_m)$ avec $\Sigma=\Sigma_o \cup \Sigma_{uo}$, le déterminisé de G est l'automate $det(G) = (X, x_0, \Sigma_o, \delta^d, X_m)$, où $X \subseteq 2^Q$ et $\delta^d \subseteq Q \otimes \Sigma_o \otimes Q$ sont définis par :

- $x_0=\{q_o\} \in X$
- *Pour* $x \in X$ *et* $\sigma \in \Sigma_o$, $\delta^d(x,\sigma) = x' = \cup_{q \in x}\{\Delta_G(q,\mu.\sigma) / q \in x , \mu \in (\Sigma_{uo})^* \}$

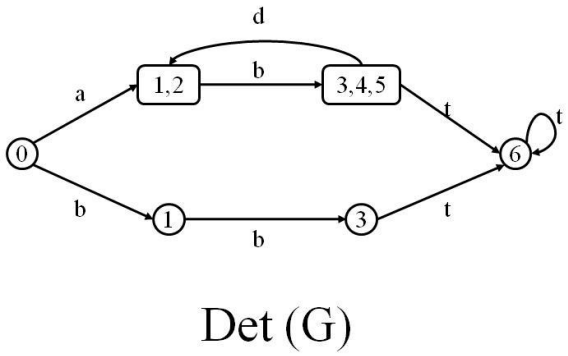


Figure II-5 – Automate déterminisé de l'exemple de la Figure II-3

L'opération de déterminisation consiste donc à construire un automate déterministe sans événements inobservables (Figure II-5). On a trace(G)=L(detG)=L(ε(G)).

Si ε(G) est déterministe, alors det(G)= ε(G).

II.I.2 Automates temporisés

Les automates temporisés (AT) étendent les automates finis classiques avec des horloges explicites. Ils ont été introduits par Allur et Dill en 1994 (Alur, et al., 1994). Le temps est ajouté au modèle classique des automates sous forme d'horloges et de prédicats sur ces horloges. Les prédicats sont de deux types :

- Les gardes : ce sont des horloges associées aux transitions discrètes et qui donnent des contraintes à respecter sur les horloges pour pouvoir franchir cette transition.
- Les invariants : il s'agit de contraintes temporelles exprimées à l'aide des horloges associées aux locations. Ces contraintes sont activées lorsque le système atteint une location. Elles expriment les conditions nécessaires à respecter pour demeurer dans la location.

Définition II-10 : Automate temporisé

Un automate temporisé est un tuple $G_t = (Q, q_0, X, \Sigma, E, Inv, F, R)$ avec :

- *Q est un ensemble fini de locations ;*
- *$q_0 \in Q$ est la location initiale ;*
- *X est un ensemble fini d'horloges à valeurs réelles positives ;*
- *Σ est un ensemble fini d'actions (ou événements) encore appelé alphabet fini ;*
- *$E \subseteq Q * C(X) * \Sigma * 2^X * Q$ est un ensemble fini de transitions ;*
    - *$C(X)$ représente l'ensemble des contraintes sur les variables de X, constitué de la conjonction de contraintes atomiques. Etant donné x une horloge de X, une contrainte temporelle atomique à partir de x est de la forme 'x ~ α' avec $\alpha \in T$ et $\sim \in \{<, \leq, >, \geq, =\}$. Nous pouvons également avoir des contraintes du type '$x - y \sim \alpha$' avec $x, y \in T$;*

- *2^X représente des sous-ensembles de l'ensemble des variables d'horloges. Le sous-ensemble associé à une transition représente les horloges à réinitialiser.*

- *Inv $\in$ C(X) associe un invariant à chaque location q ;*
- *F $\subseteq$ Q et R $\subseteq$ Q sont respectivement les locations finales et répétées de G_t.*

Remarque : Quand cela n'est pas nécessaire, nous omettrons les ensembles *F* et *R*.

Dans l'illustration de la Figure II-6, nous avons deux locations, 'q_0' et 'q_1'. A chacune de ces locations est associé un invariant. Un invariant est un ensemble de contraintes conjonctives que doivent vérifier les horloges de l'automate dans l'état considéré. Soit x et y deux horloges de *X*. Un invariant à partir de x et y est de la forme 'x < α et y ≤β'. Nous noterons que les seuls opérateurs que nous pouvons utiliser dans ce cadre sont ceux indiqués. Les valeurs limites des horloges ne sont pas forcément les mêmes car avec la réinitialisation, elles n'ont pas forcément la même date de départ. On ne peut rester dans une location que tant que l'invariant est vérifié. Ainsi dans notre exemple, on ne peut rester dans la location 'q_0' que tant que l'horloge x est strictement inférieure à la limite 5.

Une transition est franchie lorsque l'action survient alors que la garde est vérifiée. Donc, dans l'exemple de la Figure II-6, la transition est franchie si 'e' survient alors qu'il s'est écoulé plus que 2 unités de temps (ut ci-après) depuis que le système est en 'q_0'. A ce moment, on réinitialise l'horloge $x := 0$.

Il est très important de noter que les automates temporisés, du fait de la possibilité de réinitialisation des horloges, offrent la possibilité de construire des modèles de systèmes avec un temps absolu ou un temps relatif. Dans cet exemple, l'horloge x modélise un temps relatif.

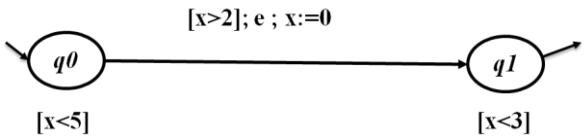


Figure II-6 – Exemple (1) d'automate temporisé

Définition II-11: Chemin fini

Soit G_t un automate temporisé. Un chemin fini de G_t est une séquence finie de transitions successives notée P :

$$q_0 \xrightarrow{g_1 ; a_1 ; r_1} q_1 \xrightarrow{g_2 ; a_2 ; r_2} q_2 \cdots q_{n-1} \xrightarrow{g_n ; a_n ; r_n} q_n$$

où $(q_{i-1}, g_i, a_i, r_i, q_i) \in E$ pour tout $i \in [1..n]$.

Le chemin est dit acceptant si la location finale q_n appartient à F.

Soit Σ un alphabet fini. Nous notons Σ^* l'ensemble des suites finies sur Σ et Σ^ω l'ensemble des suites infinies sur Σ ; $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

Définition II-12 : Mots temporisés

Un mot temporisé $w = (a_i, d_i)_{1 \le i \le n}$ *est un élément de* $(\Sigma \otimes T)^*$, *pour tout* $i \geq 0, a_i \in \Sigma, d_i \in T$.

d_i *représente le délai écoulé avant que survient l'événement* a_i. *Il peut aussi être défini par la paire* $w = (s, \tau)$ *avec* $s = (a_i)_{1 \le i \le n}$ *un mot de* Σ^* *et* $\tau = (d_i)_{1 \le i \le n}$ *une séquence temporelle sur* T^*.

A tout mot temporisé, on peut associer une durée que nous noterons $duree(w) = \sum_{i=1}^{n} d_i$.

Définition II-13 : Valuation temporelle

Une valuation temporelle sur l'ensemble d'horloge X est une application $\nu : X \to T$ *qui associe à chaque horloge x une valeur dans T. L'ensemble de toutes les valuations de X est noté* T^X. *Soit* $t \in T$, *la valuation v + t est définie par* $(\nu + t)(x) = \nu(x) + t, \forall x \in X$. *Pour un sous-ensemble r de X, nous notons* $[r := 0]\nu$ *la valuation telle que pour tout* $x \in r, ([r := 0]\nu)(x) = 0$ *et pour tout* $x \in X \setminus r, ([r := 0]\nu)(x) = \nu(x)$.

Définition II-14 : Une exécution

Une exécution de l'automate sur le chemin P est une suite de la forme :

$$\rho = < q_0, \nu_0 > \xrightarrow[d_1]{g_1 ; a_1 ; r_1} < q_1, \nu_1 > \cdots < q_{n-1}, \nu_{n-1} > \xrightarrow[d_n]{g_n ; a_n ; r_n} < q_n, \nu_n >$$

où q_0 *est la location initiale,* $\tau = (d_i)_{1 \le i \le n}$ *est une séquence temporelle,* $\nu_0(x) = 0, \forall x \in X, et\ (\nu_i)_{1 \le i \le n}$ *sont les valuations temporelles.*

Dans une exécution, un état $< q_i, \nu_i >$ est une configuration de l'automate à un instant donné. En reprenant l'exemple de l'exécution ρ, une transition représente un écoulement de temps de durée d_i puis un passage de la location q_{i-1} à la location q_i. Ceci se décrit par les deux transitions successives suivantes :

- $< q_{i-1}, \nu_{i-1} > \xrightarrow{d_i} < q_{i-1}, \nu_{i-1} + d_i >$ qui correspond à l'écoulement du temps dans l'état q_{i-1},
- $< q_{i-1}, \nu_{i-1} + d_i > \xrightarrow{a_i} < q_i, \nu_i >$ qui correspond au passage de la location q_{i-1} à la location q_i tel que $(\nu_{i-1} + d_i) \vDash g_i\ et\ \nu i = [r_i := 0](\nu_{i-1} + d_i)$.

La trace de ρ, $(tr(\rho))$ *est le mot temporisé* $tr(\rho) = (a_i, d_i)_{1 \le i \le n}$.

$tr(\rho)$ *est accepté par* G_t *s'il est la trace d'une exécution de* G_t *se terminant dans un état de F.*

Définition II-15 : Langage temporisé

L'ensemble des traces d'exécutions d'un automate temporisé A constitue un langage L(A) avec $L^*(A)$, l'ensemble de ses mots temporisés finis et $L^\omega(A)$, l'ensemble de ses mots temporisés infinis tels que $L(A)= L^*(A) \cup L^\omega(A)$.

Le graphe des régions RG(A) d'un automate temporisé A, est un automate fini sur l'alphabet $\Sigma' = C(X) * \sum * 2^X \cup \{\varepsilon\}$. Ses états sont constitués de couples (q, R), où $q \in Q$ est une location de A et R une région de $R^X_{\geq 0}$. Il permet d'étudier les propriétés d'un automate temporisé car il y a une bisimulation temporelle entre un automate temporisé et son graphe des régions. Il est formellement défini par :

Définition II-16 : Graphe des régions

Le graphe de région d'un automate temporisé G est un automate à états fini $RG(G)=\{Q \otimes R^X_{\geq 0}, (q_0,0), \Sigma' = C(X) * \sum * 2^X \cup \{\varepsilon\}, \delta^{RG}\}$ avec :

- *$\delta^{RG}((q, R),e)=(q', R')$ s'il existe une transition $\delta(q,g,e,r)=q'$ dans G avec R satisfaisant g et $R'=R[r->0]$ (i.e. l'ensemble des variables de R avec celle de r remise à zéro) ;*
- *$\delta^{RG}((q, R), \preceq)=(q', R')$ si $R \preceq R'$.*

Considérons l'exemple d'automate temporisé de la Figure II-7. La valeur constante maximale des contraintes atomiques est 1. On peut en déduire l'automate des régions correspondant (Figure II-8). Cet exemple montre que le modèle d'un système sous forme d'automate temporisé est plus concis que sous forme d'automate à états finis. Par contre, on peut noter que le graphe des régions fait clairement apparaître un état bloquant qui n'était pas apparent sur la Figure II-7. Le graphe des régions permet d'analyser le comportement d'un système temporisé et d'en vérifier les propriétés.

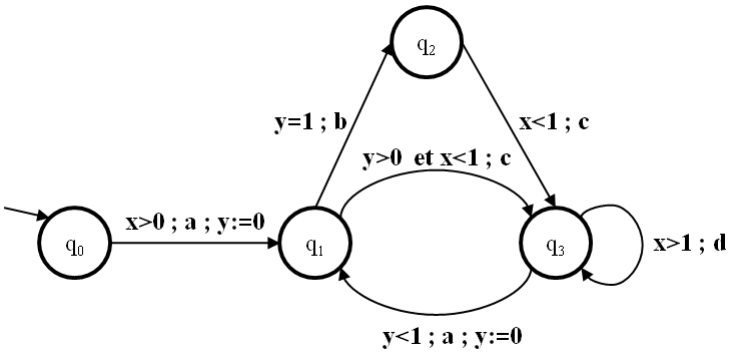


Figure II-7 – Exemple (2) d'automate temporisé

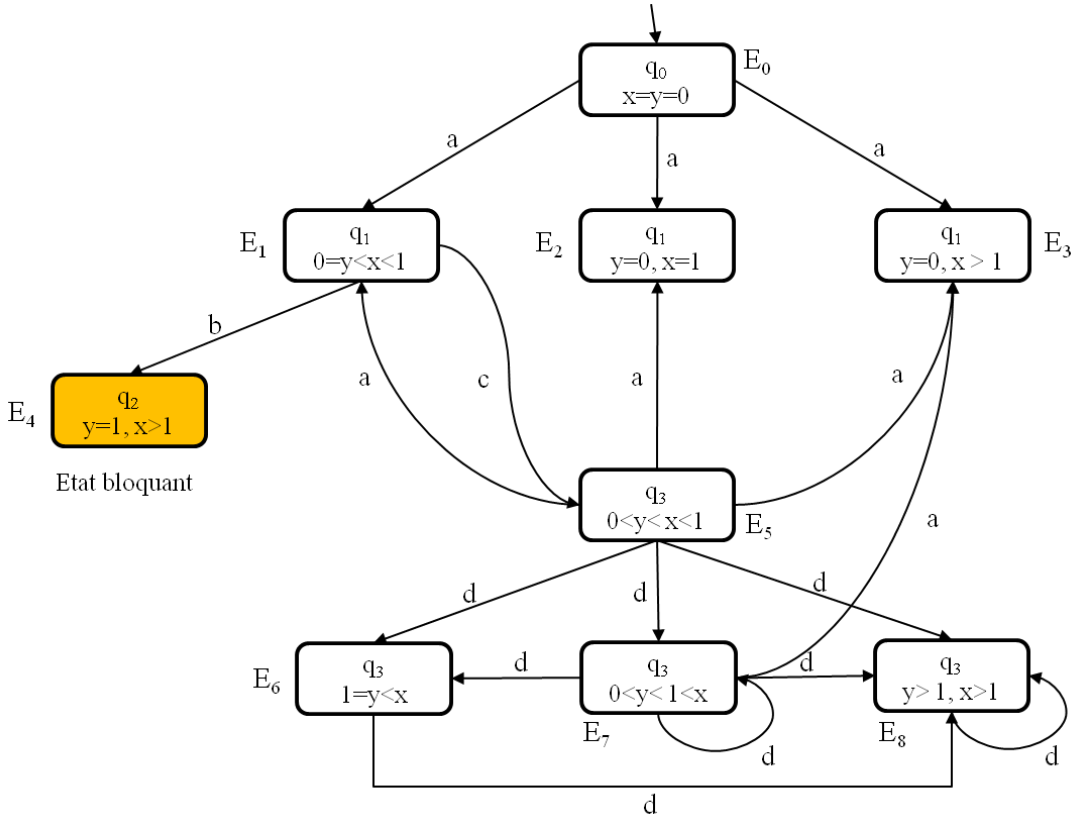


Figure II-8 – Automate des régions correspondant à l'exemple de la Figure II-7

Définition II-17 : Produit synchrone d'automates temporisés

Le produit des automates temporisés est défini de manière standard : les automates se synchronisent sur les actions communes excepté l'action invisible ε.

Soit $G_i = (Q_i, q_0^i, X_i, \Sigma^i, E_i, Inv_i)$, $i \in \{1, 2\}$, *deux automates temporisés tels que* $X_1 \cap X_2 = \varnothing$. *Le produit synchronisé de* G_1 *et* G_2 *est l'automate temporisé* $G_1 \times G_2 = (Q, q_0, X, \Sigma, E, Inv)$ *défini par :*

– $Q = Q_1 \times Q_2$;

– $q_0 = (q_0^1, q_0^2)$;

– $\Sigma = \Sigma^1 \cup \Sigma^2$;

– $X = X_1 \cup X_2$;

– $E \subseteq Q \times C(X) \times \Sigma \times 2^X \times Q$ et $((q_1,q_2), g_{1,2}, \sigma, R, (q'_1,q'_2)) \in E$ si :

- soit $\sigma \in \Sigma^1 \cap \Sigma^2$ et

(i) $(q_k, g_k, \sigma, r_k, q'_k) \in E$ pour $k = 1$ et $k = 2$;

(ii) $g_{1,2} = g_1 \wedge g_2$;

(iii) $R = r_1 \cup r_2$;

- *soit pour* $k = 1$ *ou* $k = 2$, $\sigma \in (\Sigma^k \setminus \Sigma^{3-k}) \cup \{\varepsilon\}$, *et*

(i) $(q_k, g_k, \sigma, r_k, q'_k) \in E_k$;

(ii) $g_{1,2} = g_k$;

(iii) $R = r_k$;

– $Inv(q_1, q_2) = Inv(q_1) \wedge Inv(q_2)$.

II.I.3 Réseaux de Petri temporisés

Les réseaux de Petri sont un formalisme proposé en 1962 par Carl Adam Petri (Brams, 1982) (David, et al., 1992). Il a proposé un outil permettant notamment de simplifier la modélisation des systèmes discrets caractérisés par du parallélisme et de la synchronisation. Il existe aujourd'hui de nombreuses abréviations (pouvant se ramener au modèle d'origine) et extensions (expressivité plus grande) au formalisme d'origine que l'on qualifie de Réseaux de Petri ordinaires (RdP). Nous nous intéressons dans cette étude aux extensions manipulant un temps explicite.

Les réseaux de Petri Temporisés sont obtenus à partir des réseaux de Petri ordinaires en associant des durées aux transitions ou aux places ou aux arcs du RdP. On parle de RdP T-temporisés, P-temporisés ou A-temporisés. Dans la suite, nous présentons les RdP T-temporisés et P-temporisés.

II.I.3.A Les Réseaux de Petri T-temporisé

Ce type de RdP a été développé par Ramchandani (Ramchandani, 1974). Dans ce modèle, une durée est associée à chaque transition, et le marquage est décomposé en deux sous-ensembles : M^{nr} l'ensemble des jetons non réservés et, M^r l'ensemble des jetons réservés qui ne peuvent donc pas participer à la sensibilisation d'autres transitions. La réservation se fait de la façon suivante : quand dans les places amont d'une transition T_i il y a assez de jetons pour que T_i soit franchie, T_i réserve les jetons qui vont la franchir pendant la durée qui lui est associée, ces jetons deviennent donc non-disponibles, et au bout de cette durée seulement, elle est franchie en ajoutant des jetons non réservés (disponibles) aux places avales.

D'un point de vue sémantique, les RdP T-temporisés permettent de modéliser des durées associées à des actions.

Définition II-18 : RdP T-temporisé

Un RdP T-temporisé est un 5-uplet <P, T, Pré, Post, δ >, avec : <P, T, Pré, Post > un RdP marqué, δ : T-->Q est une application de l'ensemble des transitions T dans l'ensemble des nombres rationnels positifs ou nuls / δ(T_i) = d_i : temporisation associée à la transition T_i.

Fonctionnement

Une marque peut avoir deux états. Elle peut être réservée pour le franchissement d'une transition T_i ou non réservée. Lorsqu'une marque est réservée, elle devient inaccessible pour les autres transitions en conflit avec la transition qui l'a réservée.

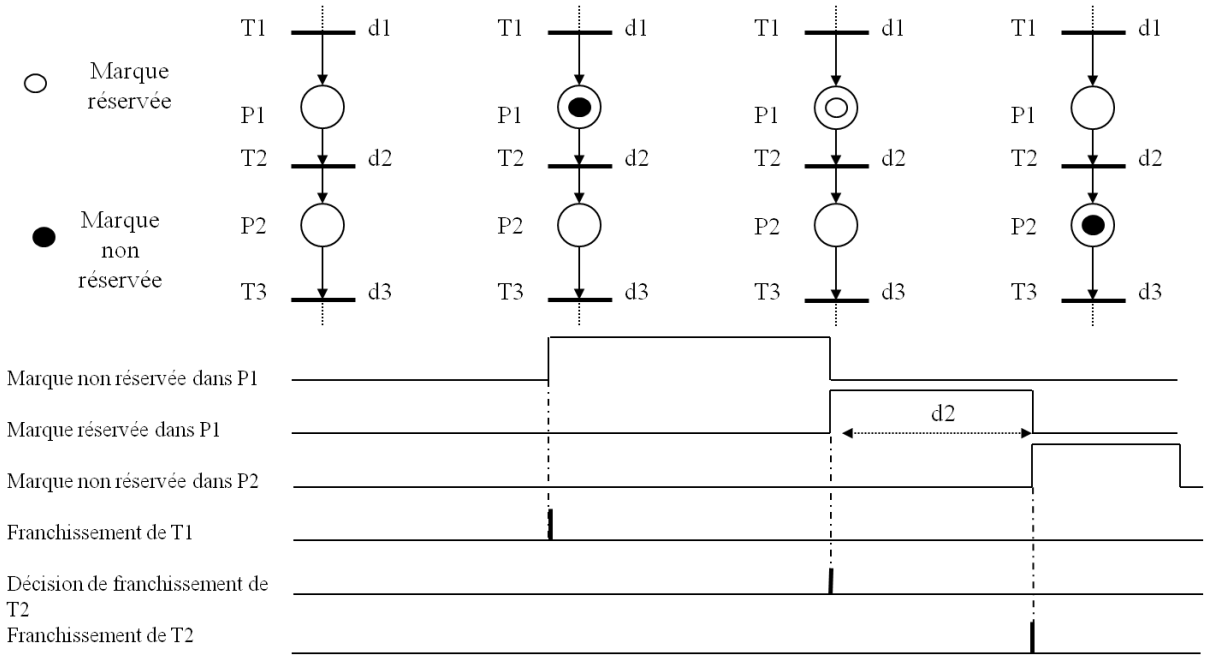


Figure II-9 – Franchissement d'une transition dans un RdP T-temporisé (David, et al., 1992)

Franchissement d'une transition

A t=0, le marquage initial M_0 ne comprend que des marques non réservées. Cependant à t quelconque, le marquage M du RdP est la somme (l'union) de marques non réservés M^{nr} et des marques réservés M^r : $M=M^{nr} + M^r$.

Une transition est validée pour un marquage si elle est validée pour les marques non réservées. Le franchissement s'effectue pendant une durée égale à d_i en ne retirant des places d'entrée que les marques réservées.

II.I.3.B Les Réseaux de Petri P-temporisé

Contrairement aux RdP T-temporisés, les durées dans ce type de modèle sont associées aux places. Chaque jeton arrivant dans une place n'est disponible qu'après y avoir séjourné pendant la durée associée.

De la même façon que dans les RdP T-temporisés, le marquage d'un RdP P-temporisé peut être décomposé en deux sous-ensembles : J_d, l'ensemble des jetons disponibles et J_{nd} l'ensemble des jetons non disponibles. Un jeton déposé à la date t dans une place P à laquelle est associée une durée θ reste indisponible entre les dates t et t + θ. Il devient ensuite disponible pour un tir éventuel d'une transition sensibilisée en aval de P. Néanmoins, cela n'indique pas quand une transition peut être tirée. C'est pour cette raison qu'une caractérisation du fonctionnement de ce modèle doit être ajoutée. Il existe en effet deux types de fonctionnements pour un RdP P-temporisé : un fonctionnement au plus tôt ou à vitesse maximale, où une transition est tirée dès qu'elle est sensibilisée et un fonctionnement sans obligation (Sifakis, 1977).

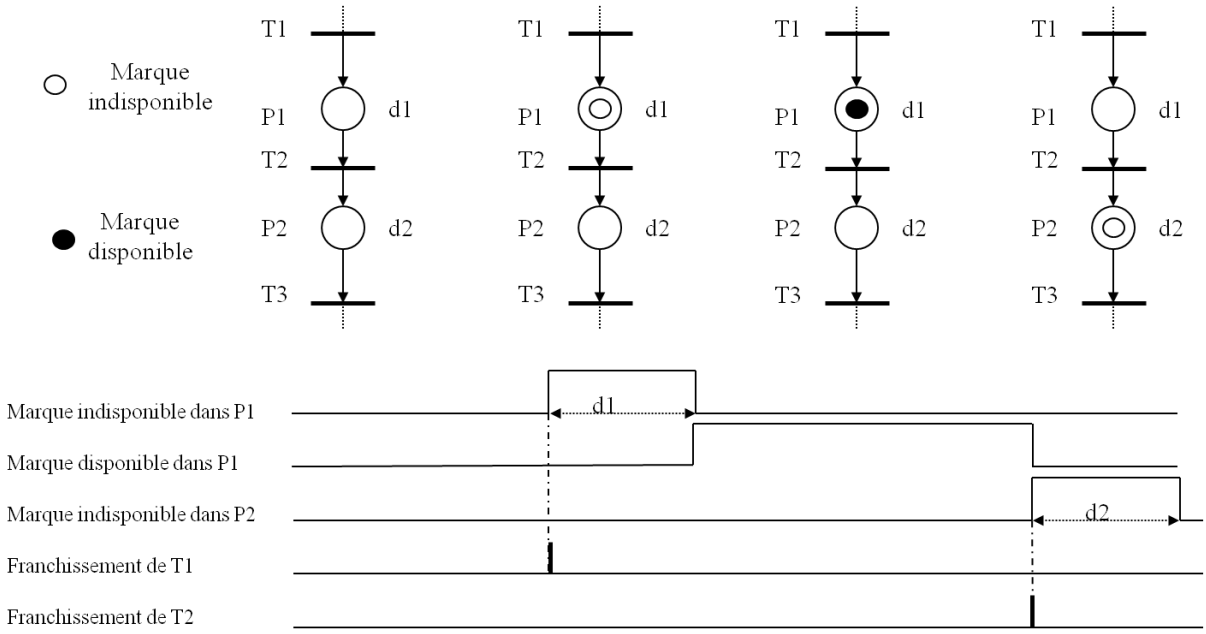


Figure II-10 – Franchissement d'une transition dans un RdP P-temporisé (David, et al., 1992)

D'un point de vue sémantique, les RdP P-temporisés permettent de modéliser des durées minimales d'être dans un état (marquage : station du jeton dans une place).

Définition II-19 : RdP P-temporisé

Un RdP P-temporisé est un 5-uplet <P, T, Pré, Post, δ >, avec : <P, T, Pré, Post > un RdP marqué, δ : P-> Q est une application de l'ensemble des places P dans l'ensemble des nombres rationnels positifs ou nuls / δ(P_i) = d_i : temporisation associée à la place P_i.

Fonctionnement

Lorsqu'une marque est déposée dans une place P_i, cette marque doit y rester pendant une durée minimale de d_i. Pendant cette durée, la marque est dite indisponible**.**

Franchissement d'une transition

A t = 0, le marquage initial M_0 ne comprend que des marques disponibles. Cependant, à t quelconque, le marquage M du RdP est la somme (l'union) de marques disponibles M^d et de marques indisponibles M^i : $M=M^d + M^i$

Une transition est validée pour un marquage M si elle est validée pour les marques disponibles. Le franchissement s'effectue à durée nulle. Le franchissement s'effectue comme dans un RdP autonome en ne retirant des places d'entrée que les marques disponibles.

II.I.3.C Remarques sur les RdP P-temporisé et T-temporisé

L'interprétation des deux modèles RdP T-temporisés et P-temporisés dépend de la sémantique associée. Dans le cas d'un fonctionnement à vitesse maximale, les deux aboutissent à un régime permanent après un régime transitoire. Les deux modèles sont équivalents. On peut passer de l'un à l'autre par des transformations simples (David, et al., 1992).

Le fonctionnement au plus tôt n'est pas forcément satisfaisant. En effet, il est parfois plus optimal, au niveau du fonctionnement d'un RdP, de ne pas tirer toutes les transitions dès qu'elles sont sensibilisées. Par ailleurs, la contrainte temporelle associée à un RdP T-temporisé est assez limitée puisqu'il s'agit d'une simple constante contrairement à des modèles qui permettent de prendre en compte des incertitudes ou indéterminismes sur le fonctionnement sous forme d'intervalles, par exemple les RdP temporels.

II.I.4 Réseaux de Petri temporels

Les RdP temporels sont des extensions temporelles des RdP. On associe des intervalles de temps (deux dates min et max) soit aux places (RdP P-temporels), soit aux transitions (T-temporels), soit aux arcs (RdP A-temporels). On distingue deux sémantiques : la sémantique forte (strong semantics) et la sémantique faible (weak semantics). Les modèles courants de RdP temporels sont : les RdP T-temporels en sémantique forte (Merlin, 1974), les RdP P-temporels en sémantique forte (Khansa, 1997) et les RdP A-temporels en sémantique faible (de Frutos Escrig, et al., 2000).

II.I.4.A Les Réseaux de Petri T-temporels

Définition II-20 : RdP T-temporel

Un réseau de Petri T-temporel est un 6-uplet < P, T, Pre, Post, M0, Is>, tel que :

- *P= {P1, P2,..,Pn} est un ensemble fini des places ;*
- *T = {T1,T2,…,Tn} est un ensemble fini des transitions, $P \cap T = \varnothing$;*
- *Pré est l'application d'incidence avant définie de $P \otimes T \rightarrow \{0,1\}$;*
- *Post est l'application d'incidence arrière définie de $P \otimes T \rightarrow \{0,1\}$;*

- *M_0 est le vecteur marquage initial ;*
- *Is : $T \rightarrow Q \otimes (Q \cup \infty)$ est une fonction qui associe un intervalle de temps à chaque transition. Q représente l'ensemble des nombres positifs.*
- *Is (Ti) = $[a_i^s, b_i^s]$; $a_i^s \leq b_i^s$; $0 \leq b < +\infty$; $0 \leq b_i^s \leq +\infty$; a_i^s et b_i^s sont des nombres rationnels positifs.*

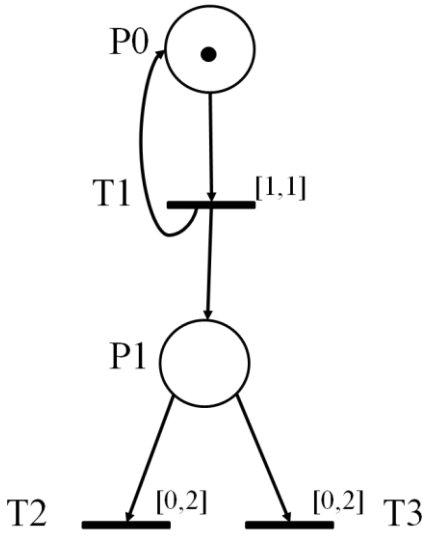


Figure II-11 – Exemple de RdP T-temporel (Diaz, 2001)

La Figure II-11 illustre un exemple de RdP T-temporel.

Fonctionnement

L'intervalle $[a_i^s, b_i^s]$ est appelé intervalle statique de franchissement. Il modélise la contrainte temporelle imposée pour le franchissement de la transition Ti. L'origine de temps considérée pour la définition de cet intervalle est l'instant de validation de la transition Ti. a_i^s est l'instant de franchissement au plus tôt et b_i^s est l'instant de franchissement au plus tard.

L'état d'un RdP T-temporel est défini par le couple *S : (M, I)* où *M* représente le marquage du RdP autonome sous-jacent et *I* est le vecteur des intervalles de franchissement. Chaque élément de *I* mémorise les dates possibles pour le franchissement d'une transition validée par le marquage.

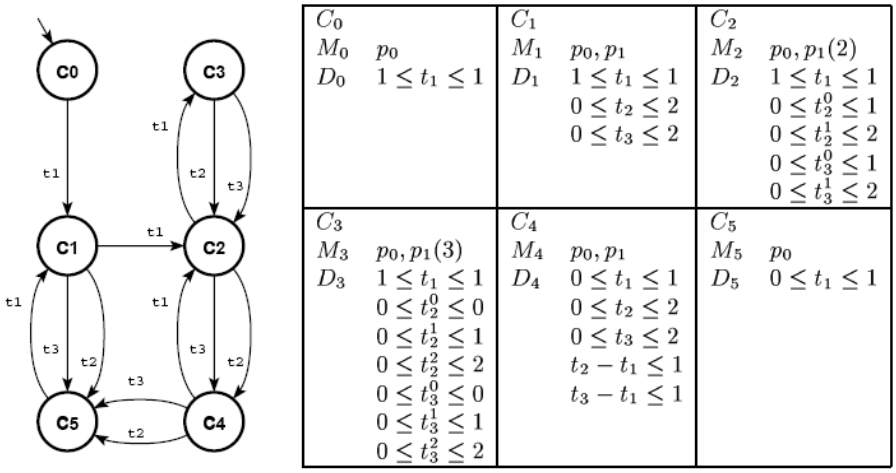
Franchissement d'une transition

Une transition peut être franchie si deux conditions sont vérifiées. La première condition c'est la condition de *marquage* : la transition doit être validée par le marquage du RdP autonome sous-jacent. La deuxième condition c'est la *condition de temps* : Une transition validée ne peut pas être franchie avant l'écoulement d'une période égale à son instant de franchissement au plus tôt depuis l'instant de sa validation. Par contre, elle doit être franchie avant une période de temps égale à son instant de franchissement au plus tard.

Concernant le tir des transitions, on trouve deux sémantiques distinctes : la sémantique faible (Weak Time Semantics : WTS) et la sémantique forte (Strong Time Semantics : STS). En sémantique faible, une transition *peut* être tirée dans son intervalle de tir. Après la date de tir au plus tard, elle n'est plus franchissable. Cependant, en sémantique forte, une transition *doit* être tirée dans son intervalle de tir à moins qu'elle ne soit désensibilisée par le tir d'une autre transition. A la date de tir au plus tard, la transition est franchie si cela n'a pas été le cas auparavant.

Une transition t est *k-sensibilisée* par un marquage M si k > 0 et k est le plus grand entier tel que M ≥ k. Pré(t). Une transition t est *multi-sensibilisée* par un marquage M si elle est k-sensibilisée par M pour un certain k > 1.

Différentes interprétations peuvent être associées au contexte des transitions multi-sensibilisées. Dans l'interprétation standard de la sensibilisation, chaque transition sensibilisée est associée à une et une seule variable temporelle du domaine de tir, qu'elle soit ou non multi-sensibilisée. Dans l'interprétation étendue, k variables temporelles sont associées à une transition multi-sensibilisée. Du coup cela complexifie les règles de franchissement dans la mesure où il est nécessaire de savoir quelle instance de jeton a franchi la transition afin de maintenir dans les bons intervalles les variables associées aux jetons restants. Une manière de résoudre ce problème consiste à faire l'hypothèse que l'instance la plus ancienne est celle qui franchie en premier la transition (règle PSPT : premier sensibilisée premier tirée).



C_0 M_0 p_0 D_0 $1 \leq t_1 \leq 1$	C_1 M_1 p_0, p_1 D_1 $1 \leq t_1 \leq 1$ $0 \leq t_2 \leq 2$ $0 \leq t_3 \leq 2$	C_2 M_2 $p_0, p_1(2)$ D_2 $1 \leq t_1 \leq 1$ $0 \leq t_2^0 \leq 1$ $0 \leq t_2^1 \leq 2$ $0 \leq t_3^0 \leq 1$ $0 \leq t_3^1 \leq 2$
C_3 M_3 $p_0, p_1(3)$ D_3 $1 \leq t_1 \leq 1$ $0 \leq t_2^0 \leq 0$ $0 \leq t_2^1 \leq 1$ $0 \leq t_2^2 \leq 2$ $0 \leq t_3^0 \leq 0$ $0 \leq t_3^1 \leq 1$ $0 \leq t_3^2 \leq 2$	C_4 M_4 p_0, p_1 D_4 $0 \leq t_1 \leq 1$ $0 \leq t_2 \leq 2$ $0 \leq t_3 \leq 2$ $t_2 - t_1 \leq 1$ $t_3 - t_1 \leq 1$	C_5 M_5 p_0 D_5 $0 \leq t_1 \leq 1$

Figure II-12 – Graphe de classes de l'exemple de la Figure II-11 dans le cadre de la règle PSPT (Diaz, 2001)

Afin d'étudier les propriétés d'un RdP T-temporels, on peut générer son graphe de classes. Une classe d'états C, d'un RdP temporel, est un couple (M, D) dans lequel M est un marquage, et D un ensemble d'inéquations appelé domaine de tir $\{\vec{t}\ t.q.\ A\vec{t} \leq \vec{b}\}$. Les vecteurs ont une composante pour chaque transition sensibilisée par M. Il a été démontré que dans le cadre d'un RdP borné, pour l'interprétation standard où la règle PSPT, le graphe de classes est fini (cf. Figure II-12).

Le graphe de classes peut être obtenu par la méthode énumérative (Berthomieu, et al., 1983) (Berthomieu, et al., 2003) qui est implémenté dans l'outil TINA développé au LAAS (Berthomieu, et al., 2004). Pour la construction et l'analyse des RdP T-Temporels, une alternative importante est constituée par l'outil ROMEO développé à l'IRCCYN (David, et al., 2008). Un des intérêts de ROMEO réside dans sa capacité à traduire un modèle RdP T-temporel en automate temporisé, offrant ainsi la possibilité d'utiliser des outils dédiés à ce formalisme comme KRONOS ou UPPAAL pour l'analyse des propriétés du modèle du système.

II.I.4.B Les Réseaux de Petri P-temporels

Le modèle RdP P-temporel a été introduit dans (Khansa, 1997) pour modéliser le comportement des systèmes de production dont le fonctionnement est soumis à des contraintes de synchronisation très fortes. Cet outil est dérivé du modèle RdP autonome en lui associant un intervalle de temps $[a_i, b_i]$ à chaque place P_i. Généralement, lorsqu'une marque arrive dans une place, elle ne peut pas participer tout de suite à la validation d'une transition. Dans ce cas, on dit que la marque est indisponible. Une marque dans la place P_i participe à la validation de ses transitions de sortie si elle est restée au moins durant le temps a_i dans cette place. Elle doit quitter la place P_i par le franchissement d'une de ses transitions de sortie au plus tard lorsque sa durée de séjour atteint la borne maximale b_i de l'intervalle de temps associé. Une fois cette valeur dépassée, la marque perd sa capacité à valider des transitions. On dit qu'elle devient morte et elle ne peut plus jamais participer à la validation d'une transition.

Définition II-21 : RdP P-temporel

Un réseau de Petri P-temporel est un 6-uplet < P, T, Pre, Post, M0, Is>, tel que :

- *P= {P1, P2,…,Pn} est un ensemble fini des places ;*
- *T = {T1,T2,…,Tn} est un ensemble fini des transitions, $P \cap T = \varnothing$;*
- *Pré est l'application d'incidence avant définie de $P \otimes T \rightarrow \{0,1\}$;*
- *Post est l'application d'incidence arrière définie de $P \otimes T \rightarrow \{0,1\}$;*
- *M_0 est le vecteur marquage initial ;*
- *Is : $P \rightarrow Q \otimes (Q \cup \infty)$ est une fonction qui associe un intervalle de temps à chaque place. Q représente l'ensemble des nombres rationnels positifs.*

  Is (Pi) = $[a_i^s, b_i^s]$; $a_i^s \leq b_i^s$; $0 \leq a_i^s < \infty$; $0 \leq b_i^s \leq \infty$

  a_i^s et b_i^s sont des nombres rationnels positifs.

Fonctionnement

Une marque dans la place P_i participe à la validation des transitions de sortie de P_i si elle est restée au moins une durée a^s_i. Par contre, elle doit quitter P_i au plus tard lorsque la durée de son séjour atteint la valeur b^s_i.

L'état d'un RdP P-temporel est défini par le couple *S : (M, I)* où *M* représente le marquage du RdP autonome sous-jacent et *I* est une application qui associe un intervalle $[a^s_i, b^s_i]$ à chaque marque j dans la place P_i. Cet intervalle est défini par rapport à la date d'arrivée de la marque j dans P_i. Il mémorise les dates de disponibilité de la marque j pour participer à la validation des transitions de sortie de P_i.

Franchissement d'une transition

Une transition T_j d'un RdP P-temporel est validée et peut être franchise si chacune de ses places d'entrée P_i contient au moins Pré(P_i, T_j) marques disponibles.

L'évolution d'un RdP P-temporel est déterminée par deux types d'événements : la mort des marques dans des places et le franchissement de transitions.

Exemple

L'exemple suivant illustre le principe de fonctionnement d'un RdP P-temporel. C'est l'exemple d'une cuve de traitement chimique et un système de transport représenté par un robot (Figure II-13).

La durée de traitement d'une pièce dans la cuve doit être de 1 à 4 ut. Ainsi, on considère qu'avant 1 ut, le traitement n'est pas encore achevé. Par contre, lorsque la durée d'immersion dans la cuve dépasse 4 ut, la pièce est considérée comme un rebut. Le rôle du robot est de transporter les pièces brutes dans la cuve et de retirer les pièces traitées.

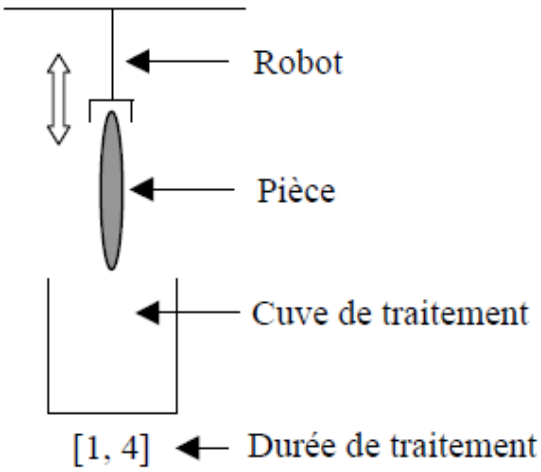


Figure II-13 – Partie d'un système de production (Sava, 2001)

Pour simplifier, on considère le traitement d'une seule pièce. Le fonctionnement de ce procédé est modélisé par le RdP P-temporel de la Figure II-14 :

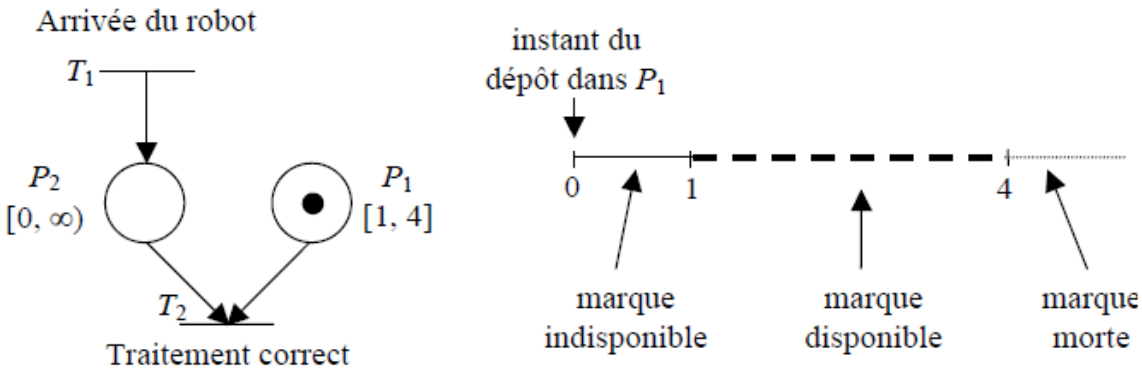


Figure II-14 – RdP P-temporel

La transition T1 modélise l'arrivée du robot au dessus de la cuve de traitement. La présence d'une marque dans la place P2 modélise l'état de disponibilité du robot au dessus de la cuve de traitement. On suppose qu'il n'y a aucune contrainte sur la durée d'attente du robot. Par conséquent, l'intervalle de temps associé à la place P2 est [0, +∞[. Le traitement d'une pièce dans la cuve est modélisé par la présence d'une marque dans la place P1. La contrainte temporelle imposée sur la durée de traitement est modélisée par l'intervalle de temps [1, 4] associé à cette place.

Le traitement correct d'une pièce est modélisé par le franchissement de la transition T2. Une marque dans la place P1 devient donc disponible pour la validation de T2 après y avoir séjourné 1 ut. Elle doit quitter cette place par le franchissement de T2 au plus tard lorsque la durée de son séjour devient 4 ut. S'il y a une marque dans P2 au plus tard à 4 ut depuis le dépôt d'une marque dans P1, la transition T2 peut être franchie. Cette évolution correspond à un traitement correct de la pièce. Par contre, s'il n'y a pas de marques dans P2 avant un délai maximal de 4 ut, la marque dans P1 devient morte ce qui modélise en pratique une perte du produit. On modélise ainsi une perte du produit qui a subit un traitement anormalement long. Le problème est alors : que faire du jeton devenu mort étant donné qu'il ne peut plus participer au franchissement des transitions ?

II.I.4.C Comparaison RdP T-temporel et RdP P-temporel

Le tableau suivant récapitule notre comparaison entre les RdP T-temporels et les RdP P-temporels.

RdP P-temporel	RdP T-temporel
• Plus approprié pour la représentation des synchronisations temporelles. • Mettent l'accent sur la synchronisation temporelle, ceci ne correspond pas au principe de fonctionnement des RdP qui considèrent d'abord une synchronisation logique. • Concept de jetons morts en sémantique faible auquel il est difficile de donner un sens pratique.	• Possibilité de résolution des conflits effectifs par la temporisation. • Respecte la philosophie des RdP. • L'interprétation est plus intuitive par rapport au RdP P-temporel parce que son évolution est déterminée par un seul type d'événement (franchissement des transactions). • Capacité à modéliser des situations d'urgence. • Bien adapté à la modélisation des systèmes temps réels critiques.

Table II-1 – Synthèse de la comparaison entre RdP P-temporels et RdP T-temporels

II.I.5 Conclusion

Nous avons présenté quatre catégories d'outils formels de modélisation du temps pour les systèmes à événements discrets. Nous avons commencé par les automates à états finis, puis les Automates Temporisés (AT) qui en sont une extension avec des horloges explicites. Ensuite, nous avons présenté les Réseaux de Petri temporisés qui sont des extensions temporelles des Réseaux de Petri (RdP) en associant des durées aux arcs, aux transitions ou aux places. On les utilise surtout pour l'évaluation des performances. Enfin, nous avons présenté les Réseaux de Petri temporels qui sont des extensions temporelles des réseaux de Petri ordinaires en associant des intervalles de temps aux arcs, aux transitions et aux places. Il existe des outils pour les RdP temporels. Nous citons TINA et ROMEO. Après cette étude, nous avons dans un premier temps retenu les AT et les RdP T-temporels. Comme nous l'avons vu dans l'exemple de la Figure II-7, les AT permettent des représentations plus concises que les automates à états finis. Comme nous l'avons déjà souligné, l'un des problèmes que nous avons à traiter est de combattre l'explosion combinatoire. Il nous semble donc préférable de choisir un modèle plus concis. D'autre part, étant donné que nous nous intéressons à des systèmes embarqués, il semble cohérent de choisir un outil permettant une représentation explicite du temps. C'est la raison pour laquelle nous comparons à présent les AT et les RdP T-temporels.

II.I.6 Comparaison de l'expressivité des AT et des RdP T-temporels

Dans ce paragraphe, nous allons comparer l'expressivité des AT et des RdP T-temporels pour justifier notre choix des AT que nous avons adopté . Soit l'AT de la Figure II-15 constitué de six états : q0,.., q5, avec q0 l'état initial et quatre événements : 'a', 'b', 'f' et 'ε'. 'f' modélise

une défaillance et ‘ε’ un événement inobservable. Cet AT modélise donc le comportement complet du système considéré : comportement normal et comportement défaillant. Ce système est diagnosticable seulement si le paramètre α est supérieur ou égal à 3. Dans ce cas, si on a ‘b’ qui survient plus de 3 ut après ‘a’, on sait qu’on a forcément eu la défaillance ‘f’. S’il y a moins de 3 ut entre les deux évènements on a un comportement normal. Cet exemple illustre le fait que l’introduction du temps dans les modèles est un facteur permettant d’augmenter la capacité à diagnostiquer sans pour autant avoir une observabilité supérieure.

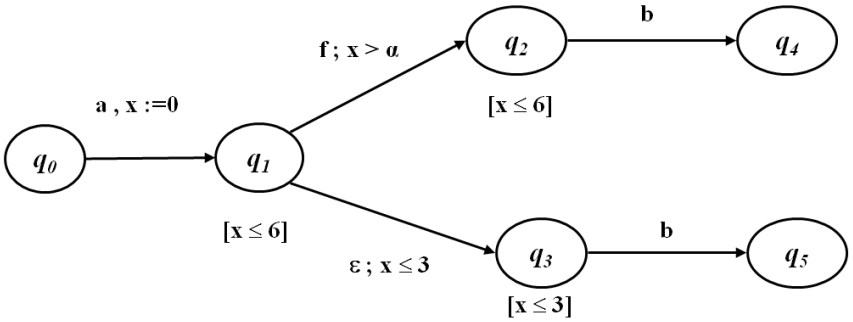


Figure II-15 – Exemple d’automate temporisé (Tripakis, 2002)

Nous avons transformé cet AT en un RdP T-temporel afin de comparer l’expressivité des deux formalismes. Aujourd’hui de nombreux auteurs travaillent sur la construction d’outils de diagnostic basés sur l’AT. Dans cette étude, nous cherchons à vérifier si les RdP T-temporels sont une alternative sachant que les modèles de commande des SED sont plus naturellement construit à l’aide du formalisme RdP. En effet, ils modélisent plus facilement le parallélisme, les synchronisations et le partage des ressources inhérents à ce type de système.

Dans le modèle RdP de la Figure II-16, les places nommées ‘hi’ (i ∈ {1,2,3}) ont été introduites pour modéliser le concept d’horloge globale des AT. En effet, dans un AT les horloges sont globales dans le sens où si elles ne sont pas réinitialisées à 0 lors du franchissement d’une transition, les contraintes sont évaluées en fonction de la valeur qu’elles ont prise dans les états antérieurs. C’est par exemple le cas, dans l’exemple de la Figure II-15 quand le système évolue de l’état ‘q1’ à l’état ‘q5’ par l’état ‘q3’. L’horloge ‘x’ n’est pas remise à zéro dans la transition de ‘q1’ à ‘q3’. Du coup, cela entraine que pour avoir cette évolution, ‘b’ doit survenir moins de 3 ut après être arrivé dans ‘q1’. La place ‘h1’ sert à définir l’horloge liée à la vérification que l’évènement τ symbolisé par la transition T2, doit survenir moins de 3 ut après être arrivé dans l’état ‘q1’. La transition T8 permet de modéliser la condition de garde liée à la transition de ‘q1’ à ‘q3’. La transition T4 associée à la place ‘h2’ permet de modéliser l’invariant de l’état ‘q3’ qui dans le modèle AT est associé à la même horloge globale ‘x’. Comme dans le RdP, le franchissement de T2 supprime le jeton de ‘h1’, on est obligé de créer la place ‘h2’ qui va modéliser la même horloge mais pour valider une autre contrainte.

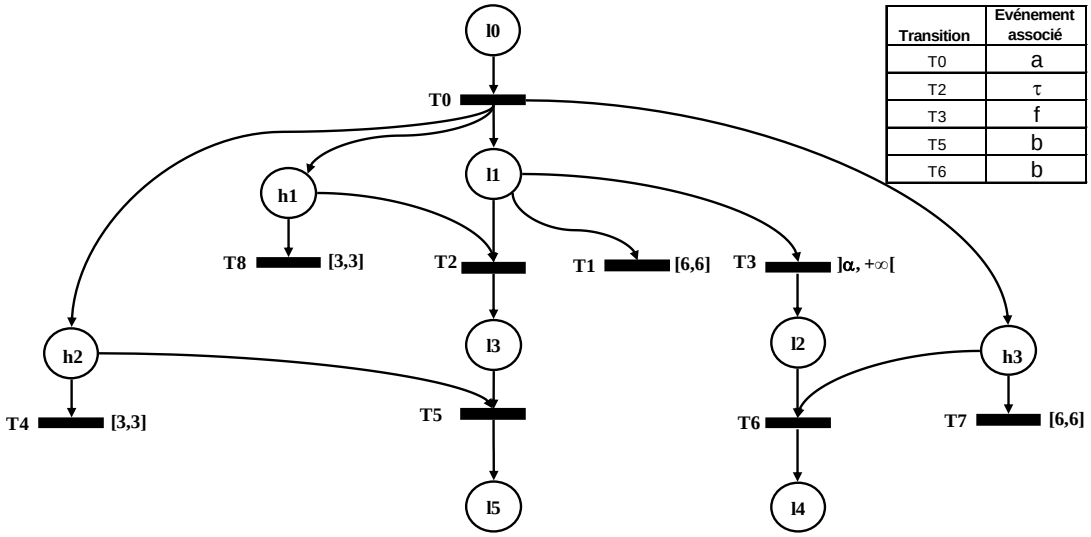


Figure II-16 – Transformation en RdP T-temporel en sémantique forte de l'automate de la Figure II-15

En comparant les deux modèles, nous remarquons que :

Le modèle AT	Le modèle RdP T-temporel
• Deux types d'expressions associées à une transition : les gardes (délais) et les actions, • Utilise des horloges globales, • Possibilité d'associer des contraintes aux états, • Plus simple graphiquement par rapport aux contraintes temporelles globales, • Différentes compositions possibles mais très combinatoire (produits cartésiens, synchrones, …), • Existence d'outils de vérification de propriétés (par exemple CMC, UPPAAL et KRONOS, …)	• Les évolutions entre états correspondent soit à des événements (actions en AT) soit à des contraintes temporelles (gardes des AT), • Utilise des horloges locales, • Modélise de manière plus naturelle le parallélisme et la synchronisation, • Possibilité sous certaines hypothèses de transformation en automates temporisés (ROMEO), • Possibilités de construction d'un graphe de classes permettant l'analyse de propriétés (accessibilité, vivacité, …), • Existence d'outils (par exemple TINA, Romeo, …)

Table II-2 – Synthèse de la comparaison entre AT et RdP T-temporels

Cet exemple montre donc qu'il peut y avoir transformation des AT en RdP T-temporel de même qu'il a été proposé des transformations de RdP T-temporels vers les AT. Cette transformation est possible avec l'outil ROMEO.

Nous avons finalement choisi d'utiliser dans ce travail le formalisme des automates temporisés. En effet, même si les RdP T-temporels en sémantique forte ont une expressivité comparable, les AT offrent des outils puissants pour le diagnostic des systèmes temporisés.

II.II Outils pour le diagnostic des systèmes discrets

De nombreux outils ont été développés ces dernières années pour le diagnostic des systèmes discrets. Nous pensons notamment aux chroniques et au concept de diagnostiqueur. Ce dernier a initialement été défini par Sampath (Sampath, et al., 1995). Dans ce qui suit, nous présentons le concept de diagnostiqueur. Pour cela, nous commençons par introduire la notion d'observateur discret à partir de laquelle il a été défini. Ensuite, nous présentons le diagnostiqueur de Sampath et le concept de vérificateur. Enfin, nous détaillons les Signatures Temporelles Causales, une sorte de chronique.

II.II.1 Observateur discret

Un automate observateur modélise l'ensemble des comportements observables d'un système. Formellement un observateur est un automate dont l'alphabet peut être partitionné en deux types d'événements : l'ensemble des événements observables Σ_o et l'ensemble des événements non-observables Σ_{uo} (également appelés événements inobservables). L'objectif d'un observateur est d'arriver à identifier l'état courant du système compte tenu des événements observables générés par le système et que peut percevoir un observateur externe. Dans la bibliographie, on trouve de nombreuses études sur les observateurs discrets. Le lecteur pourra trouver des synthèses plus ou moins complètes dans (Ozveren, et al., 1990) (Ozveren, et al., 1992) et dans (Sampath, et al., 1995). Classiquement on peut distinguer deux grandes classes d'observateurs : les observateurs pour la reconnaissance d'états et les observateurs pour la reconnaissance d'événements ou de séquences d'événements.

Dans les observateurs pour la reconnaissance d'états on peut également distinguer deux catégories d'approches. La première catégorie inspirée par les travaux de Lin et Wonham sur la synthèse de contrôleur, nécessite l'identification de l'état courant après chaque réception d'événements produit par le procédé (Lin, et al., 1988) (Lin, 1994). La seconde catégorie consiste à garantir qu'au bout d'une séquence finie d'événements on est capable d'identifier l'état atteint par le système (Ozveren, et al., 1990).

Certains travaux se sont également intéressés à la reconnaissance de séquence d'événements conduisant d'un état à un autre état du système considéré. C'est par exemple le cas de ceux relatifs au concept d'inversibilité (Ozveren, et al., 1992). Un langage est dit inversible si à tout instant, en utilisant la connaissance issue des séquences d'événements observés, on peut reconstruire l'ensemble de la séquence (y compris les événements inobservables) par rapport à un nombre borné d'événements du passé. Ce concept d'inversibilité peut donc être très utile pour faire de l'identification de défaillance. Il n'est pas à confondre avec le concept de

diagnosticabilité de Sampath (Sampath, et al., 1995) qui lui permet de faire de la détection de fautes.

La Figure II-2 illustre le modèle comportemental d'un système défini à partir du modèle de ses composants. A partir de ce modèle, dans la suite de ce chapitre, nous allons illustrer comment construire un observateur d'états. Dans le modèle de la Figure II-2 nous supposons que les deux événements 'e1' et 'e4' sont des événements observables et les deux événements 'e2' et 'e3' sont des événements inobservables. Quel est l'état courant de notre système sachant que l'état initial du système est (0,0') et que nous avons observé 'e1' puis 'e4' ?

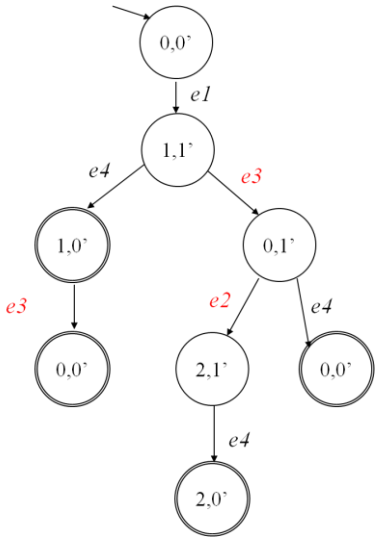


Figure II-17 – Arbre d'accessibilité partiel

Nous construisons l'arbre d'accessibilité (Figure II-17). A partir de l'état (0, 0'), nous observons 'e1'. Nous ne pouvons passer qu'à l'état (1, 1') de l'état (0, 0') avec l'observation 'e1'. Ensuite, nous observons 'e4'. Nous pouvons passer de l'état (1, 1') à l'état (1, 0'). L'événement 'e3' étant inobservable, il peut avoir lieu avant l'événement 'e4' sans qu'on ait pu l'observer. Donc de l'état (1, 1'), le système peut passer à l'état (0, 1') sur occurrence de l'événement 'e3', avant l'occurrence de l'événement 'e4'. Aussi quand l'événement 'e4', survient le système commute vers l'état (0, 0'). De l'état (0, 1'), l'événement non observable 'e2' pourrait également nous faire transiter vers l'état (2,1'), si cet événement survenait avant l'évènement 'e4'. Dans un tel cas, l'occurrence de 'e4' nous mènerait vers l'état (2,0'). Cet exemple, illustre bien la complexité de reconstruction de l'état du système compte tenu de l'observabilité courante du système. Sur la base de la séquence 'e1' suivi de 'e4', à partir de l'état (0,0'), le système peut se retrouver dans l'un des 3 états suivants : (1,0'), (2,0') ou (0,0'). La résolution de cet indéterminisme dépend de l'utilisation que nous voulons faire de l'observateur. Dans le contexte du diagnostic, différentes solutions sont envisageables comme l'attente de nouveaux événements observables permettant de supprimer des hypothèses d'évolutions (concept de chroniques) ou par exemple l'approche diagnostiqueur.

La Figure II-18 présente un exemple d'observateur construit à partir du modèle global du système de la Figure II-2. Cet observateur est obtenu en supprimant les transitions

correspondant aux événements inobservables et en fusionnant les états auxquels conduisaient ces transitions avec les états dans lesquels arrive l'événement observable terminant ainsi une séquence d'événements inobservables à partir d'un état antérieur observable. On pourra trouver un algorithme de construction d'observateurs discrets dans (Cassandras, et al., 2008).

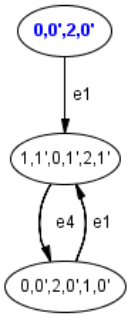


Figure II-18 – Observateur discret correspondant au modèle de la Figure II-2

II.II.2 Diagnostiqueur de Sampath

Le diagnostiqueur de Sampath est un automate observateur renseigné. Il permet de diagnostiquer l'état défaillant ou pas d'un système discret partiellement observable. Dans les systèmes discrets, les fautes sont assimilées à des événements inobservables. Mais leurs modes de défaillance peuvent être observés à l'aide de capteurs placés dans le système. Donc, l'objectif d'un diagnostiqueur est d'arriver à diagnostiquer l'occurrence de fautes par l'observation d'événements observables, et cela dans un délai fini.

Nous notons Σ l'ensemble des événements du système considéré. Nous ne pouvons observer qu'un sous-ensemble des événements du système (ensemble des événements observables Σ_o). D'autres événements ont lieu mais ne sont pas observables. On notera Σ_{uo} l'ensemble des événements non observables caractérisant le fonctionnement d'un système. Parmi les événements inobservables nous distinguons les fautes (ou défaillances). Elles appartiennent à Σ_f.

Remarque : Afin de faciliter la compréhension de la construction d'un diagnostiqueur, nous utiliserons le terme état pour désigner les états du modèle du système et le terme macro-état pour désigner les états d'un diagnostiqueur.

Dans un diagnostiqueur, le passage d'un macro-état à un autre se fait de la même manière que pour l'automate observateur correspondant. Afin de permettre le diagnostic, à chaque état du système regroupé au sein d'un macro-état, nous ajoutons son statut respectif du point de vue de l'occurrence ou non d'une faute. Les statuts possibles sont 'normal' pour un état du modèle du système atteint par un chemin ne comportant pas d'événements de faute, et l'étiquette 'Fi' du type de la faute dans le cas contraire. Ainsi les macro-états du diagnostiqueur sont des ensembles de paires (état du modèle du système, étiquette). Un macro-état d'un diagnostiqueur peut être qualifié de 'Normal' lorsque tous les candidats sont étiquetés par N,

'Fi-certain' lorsque tous les candidats ont Fi dans l'étiquette, sinon le macro-état du diagnostiqueur est qualifié de 'Fi-incertain'.

Afin d'illustrer la construction d'un diagnostiqueur, considérons de nouveau l'exemple de la Figure II-2. Nous supposons que les évènements inobservables 'e2' et 'e3' sont des événements de fautes que nous cherchons à diagnostiquer. 'e2' est une faute de type F1 et 'e3' est une faute de type 'F2'.

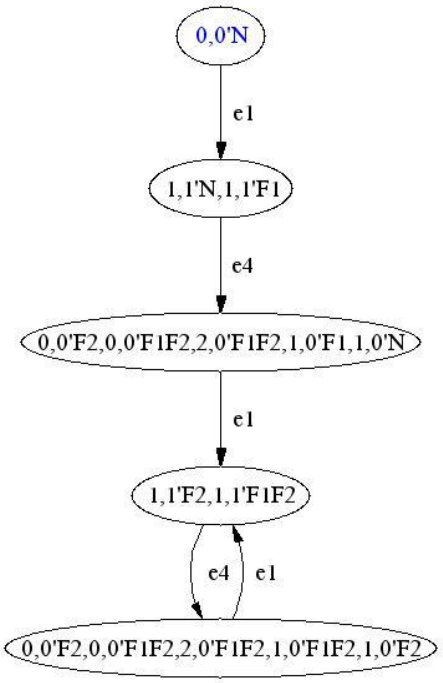


Figure II. 1 – Diagnostiqueur correspondant à l'exemple de la Figure II-2 avec F1 : e2 et F2 : e3

Après observation de 'e1' puis 'e4', l'état du système est du point de vue du diagnostic F1-F2-incertain. Cela signifie qu'il est possible que dans son évolution, le système ait eu une faute de type F1 ou une faute de type F2. Les états possibles du système sont (0,0'), (1,0'), ou (2,0'). Nous ne pouvons pas décider dans ce cas là si notre système est en état normal ou en état de faute. Une solution pourrait consister à ajouter des capteurs supplémentaires afin d'avoir plus d'informations sur l'état du système et pouvoir le diagnostiquer. En effet, avoir seulement 'e1' et 'e4' comme événements observables ne nous permet pas de diagnostiquer le système. Mais l'ajout des capteurs augmente le coût du système et peut être difficile d'un point de vue technique. En plus, ces ajouts ont surtout tendance à réduire la disponibilité du système. En effet, cela consiste à ajouter des composants qui sont souvent plus sujets aux pannes que les actionneurs eux-mêmes. Ainsi, notre idée est d'exploiter le facteur temps, pour augmenter les capacités de discrimination du diagnostiqueur.

II.II.3 Vérificateur de Jiang (Jiang, et al., 2001)

Le vérificateur est un outil qui a été conçu pour la vérification de la diagnosticabilité d'un système discret modélisé par un automate à états finis. Il est basé sur la construction d'un pseudo-diagnostiqueur G_o. C'est un automate non déterministe dont le principe de construction est similaire à la construction de ε(G) avec l'étiquetage des états selon le principe du diagnostiqueur.

Définition II-22 : Pseudo-diagnostiqueur

Etant donné $G=(Q, q_0, \Sigma, \delta)$ le modèle du comportement complet d'un système, $S=\{Fi, i \in [1..n]\} \cup N$ l'ensemble des statuts que peut prendre un de ses états et, $\Pi_f=\{1,2, ..,n\}$ l'ensemble des indices des types de fautes / $\Sigma_f = \bigcup_{i=1}^{n} \Sigma_{fi}$. 'fi' représente un type de faute. $\psi : \Sigma \rightarrow \Sigma_f \cup \varnothing$ est la fonction d'affectation d'une faute à chaque événement de Σ. On définit $G_o=(Q_o \otimes S , q^o_0 , \Sigma_o, \delta_o)$ le pseudo-diagnostiqueur de G avec :

- *$Q_o \subseteq Q$, l'ensemble des états observables de Q ;*
- *$q^o_0=(q_0,N)$ l'état initial qui est supposé normal et $q_0 \in Q_o$;*
- *Σ_o l'ensemble des événements observables de Σ ;*
- *δ_o la fonction de transition entre états observables défini par :*
  - *$\forall (q,l) \in Q_o \otimes S$, $\forall i \in \Pi_f$, si $\exists q' \in Q$ et $s.u \in \Sigma^*_{uo}$ avec $s \in \Psi(\Sigma_{fi})$ et $\sigma \in \Sigma_o$ / $\delta(q,s.u.\sigma)=q'$, alors $\delta_o((q,l), \sigma)=(q',F_i) \in Q_o \otimes S$*
  - *$\forall (q,l) \in Q_o \otimes S$, $\forall i \in \Pi_f$, si $\exists q' \in Q$ et $s \in \Sigma^*_{uo}$ avec $s \notin \Psi(\Sigma_{fi})$ et $\sigma \in \Sigma_o$ / $\delta(q,s.\sigma)=q'$, alors $\delta_o((q,l), \sigma)=(q',l) \in Q_o \otimes S$*

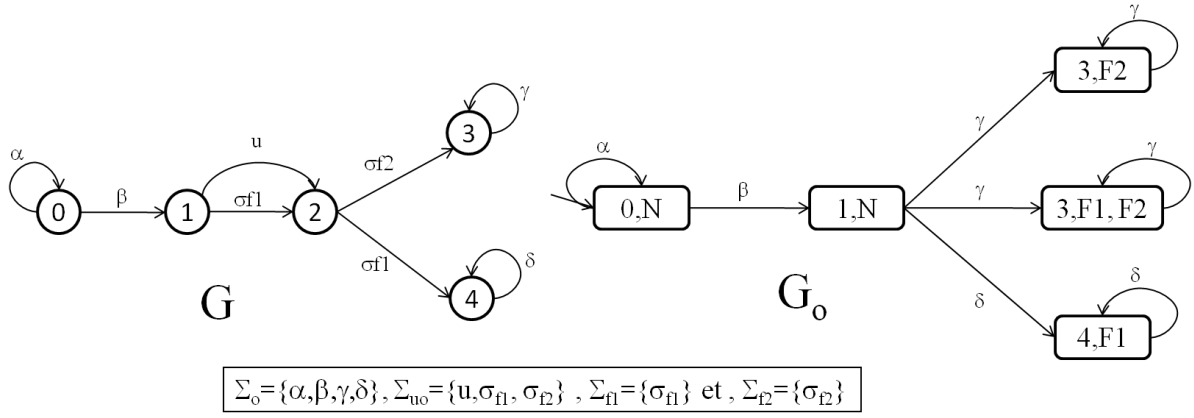


Figure II-19 – Exemple illustratif de la notion de pseudo-diagnostiqueur

Considérons l'exemple du système G défini par la Figure II-19. Pour ce système, nous supposons que $\Sigma_o=\{\alpha,\beta,\gamma,\delta\}$, $\Sigma_{uo}=\{u,\sigma_{f1}, \sigma_{f2}\}$, $\Sigma_{f1}=\{\sigma_{f1}\}$ et $\Sigma_{f2}=\{\sigma_{f2}\}$. D'après la Définition II-22, nous obtenons alors le pseudo-diagnostiqueur G_o. Nous notons dans cet exemple que le modèle obtenu est non déterministe car nous avons deux transitions à partir de (1,N) qui dépendent du même événement 'γ'. Cela est dû au fait que dans G, de l'état 1 à l'état 3, nous

avons deux chemins possibles correspondant respectivement à la séquence 'u. σ_{f2}' et '$\sigma_{f1}.\sigma_{f2}$'. La première séquence conduit à l'état '3,F1' et la deuxième à l'état '3,F1,F2'.

Il a été montré que les pseudo-diagnostiqueurs vérifient deux propriétés :

1. $L(G_o)=P_o(L(G))$;
2. Tous les nœuds d'un cycle de G_o ont la même étiquette.

Le vérificateur G_v est alors obtenu par le produit synchrone du pseudo-diagnostiqueur avec lui-même ($G_v=G_o||_sG_o$).

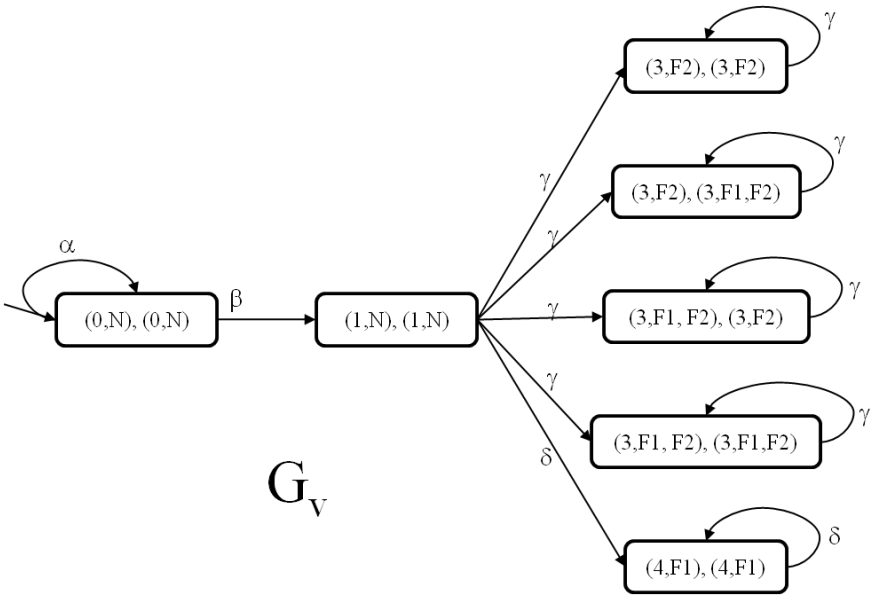


Figure II-20 – Vérificateur du système de la Figure II-19

On montre que dans le vérificateur pour tout cycle d'états (q_i, l_i^1) (q_i,l_i^2) tel que $l_i^1 \neq l_i^2$, il existe dans le pseudo-diagnostiqueur un cycle correspondant aux états de type (q_i,l_i^1) dont tous les états sont étiquetés par 'l_i^1' et un cycle correspondant aux états de type (q_i,l_i^2) dont tous les états sont étiquetés par 'l_i^2'.

Nous verrons dans le chapitre suivant que la recherche de cycles dans un vérificateur est une méthode efficace pour vérifier la diagnosticabilité d'un système.

II.II.4 Les Signatures Temporelles Causales

Les Signatures Temporelles Causales ou STC ont été définies dans (Toguyeni, et al., 1990)(Toguyeni, et al., 1997). Il s'agit d'un formalisme permettant d'interpréter des événements pour diagnostiquer des SED comme les systèmes manufacturiers.

Les STC sont des règles construites à partir de deux catégories d'événements : des événements caractérisant le contexte courant (ordre de la partie commande ou compte-rendu de capteurs), et des événements appelés symptômes et caractérisant des défaillances

(Toguyeni, et al., 1990). Chaque STC consiste en un ordonnancement partiel d'un ensemble d'événements avec l'expression de contraintes temporelles spécifiques à chaque défaillance.

Ces dernières années, plusieurs auteurs ont proposé des méthodes basées sur le codage hors ligne de modèles d'événements partiellement ordonnés. Ainsi, un concept assez proche a été proposé par le Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS), au début des années 90, les chroniques. Comme les STC, une chronique peut être définie comme un ensemble d'événements partiellement ordonnés et contraints par des relations temporelles (Dousson, et al., 1993). Au cours des dix dernières années, ce formalisme a été utilisé et/ou étendu par différents auteurs, particulièrement pour le diagnostic des défaillances (Boufaied *et al.*, 2002). Il a également été exploité par France Télécom pour le diagnostic de pannes dans les réseaux de télécommunications (Cordier, et al., 2000)(Guerraz, et al., 2004) ou pour le diagnostic des défaillances d'un service web (Cordier, et al., 2007). Une récente étude propose leur interprétation à l'aide de la technique des automates dupliqués, ou l'écriture des chroniques en réseau de Petri colorés (Bertrand, 2009).

Dans (Holloway, et al., 2000), les auteurs proposent un formalisme similaire, appelé « template language ». Cependant, ce langage de « templates » est utilisé pour indiquer le bon fonctionnement d'un système.

Le problème commun des approches orientées diagnostic est que la connaissance codée est généralement une connaissance experte. Dans le meilleur des cas, elle peut être dérivée à partir de connaissances comme les arbres de défaillance (Guerraz, et al., 2004). Toutefois, le problème est que chaque STC ou chronique est construite séparément sans tenir compte des autres règles. Aussi, il demeure nécessaire de vérifier la cohérence d'un ensemble de règles pour assurer complétude et déterminisme dans l'hypothèse de défaillances multiples (Saddem, et al., 2010) (Saddem, et al., 2011c).

Définition II-23 : Signature Temporelle Causale

Une Signature Temporelle Causale (STC) est un sous-ensemble d'événements observables partiellement ordonnés qui caractérise un comportement défaillant d'un système. Ces événements sont contraints par un ensemble de contraintes temporelles portant sur leurs occurrences. En pratique, une STC est la description d'un motif temporel définissant un ordre partiel, sur des événements décrits par leur type et leur date d'occurrence. Les relations entre les événements peuvent être logiques (conjonction) ou temporelles (séquence, absence, ...).

Ainsi chaque situation anormale peut être décrite par une ou plusieurs STC, les événements pouvant être normaux (ordre de commande, compte-rendu d'un capteur) ou pouvant correspondre à des symptômes (Toguyeni, et al., 1990). Les contraintes portent sur leur date d'occurrence. Le diagnostic à l'aide des STC consiste donc à interpréter en ligne l'occurrence d'événements afin d'instancier les motifs qui seront reconnus. Une STC est reconnue lorsque tous ses événements se produisent en respectant leurs contraintes. Ceci détermine si le système est normal ou défaillant.

II.II.4.A Syntaxe générale

Afin de faciliter l'usage des STC, nous avons proposé une syntaxe permettant une écriture simplifiée. Cependant, nous avons montré dans (Saddem, et al., 2011c) que nous pouvons utiliser des formalismes comme les automates temporisés voire les réseaux de Petri T-temporels (Saddem, et al., 2010).

Une STC est une règle composée d'une partie « conditions » et d'une partie « conséquences ». La partie « conditions » est composée de un ou plusieurs triplets. Soit (X, Y, ct) un triplet. 'X' et 'Y' sont deux événements avec 'X' l'événement de référence et 'Y' l'événement contraint, attendu par rapport à 'X'. 'ct' est une contrainte temporelle qui permet de modéliser une date, un délai ou une durée. S'il n'y a pas de contrainte temporelle entre 'X' et 'Y', 'ct' devient 'nct', ce qui signifie aucune contrainte de temps. Soit la STC décrite par l'équation (II-1). Cette STC se compose de six triplets.

$$(In, A, nct) * (In, C, nct) * (A, B, ct1) * (A, D, ct2) * (C, D, ct3) * !(D, E, ct4) => G \quad \textbf{(II-1)}$$

'A', 'B', 'C', 'D', 'E' sont des événements ; 'In' est l'événement toujours occurrent, que nous introduisons de manière théorique pour définir la référence temporelle des événements qui ne sont pas contraints ; 'G' est une défaillance ou un événement inobservable qui est déduit ; 'cti' est la i^{éme} contrainte temporelle ; 'nct' signifie pas de contrainte temporelle ; (In, A, nct) signifie que 'A' est un événement sans contrainte. Le '!' est un opérateur de négation. !(D, E, ct4) signifie que 'E' ne doit pas survenir après 'D' tant que la contrainte temporelle 'ct4' est vraie. '*' est un opérateur qui sépare chaque triplet et qui signifie « et à un autre moment ». Ainsi l'opérateur '*' doit être utilisé comme un 'ET' qui lie temporellement la reconnaissance de chaque triplet.

Dans cette écriture, nous supposons que chaque événement est pris en compte qu'une seule fois dans une STC. Cela implique que dans l'équation (II-1), l'événement 'D' dans le triplet !(D, E, ct4) est le même événement que dans les triplets (A, D, ct2) et (C, D, ct3). Dans le cas général, pour prendre en compte des occurrences distinctes du même événement et rester dans le cadre de cette hypothèse, il suffit de nommer différemment les différents événements liés à une même source (par exemple des événements liés à un même capteur). Par exemple, si nous voulons modéliser deux occurrences distinctes de 'D', il suffirait de les nommer 'D1' et 'D2' dans la STC.

Remarque : Les contraintes temporelles correspondent à un temps relatif entre l'événement de référence et l'événement attendu.

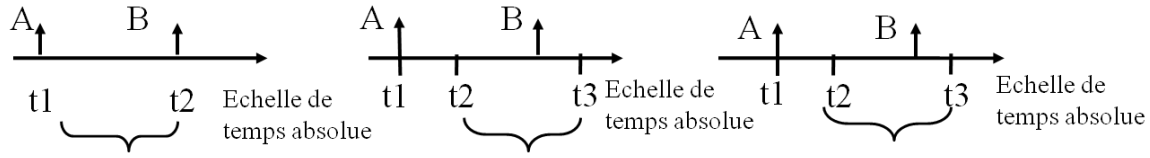
Le sens de l'équation (II-1) est ainsi le suivant. Si on a l'occurrence de l'événement 'A' (respectivement, l'occurrence de l'événement 'C'), ensuite si on a les occurrences de l'événement 'B' satisfaisant la contrainte 'ct1' par rapport à 'A' et l'occurrence de l'événement 'D' satisfaisant la contrainte 'ct2' par rapport à 'A' et la contrainte 'ct3' par

rapport à ‘C’, et si l'on n’a pas l’événement ‘E’ après l'occurrence de ‘D’ tant que la contrainte ‘ct4’ est vraie, on peut en déduire l’évènement ‘G’.

II.II.4.B Expression des contraintes temporelles

Pour exprimer les contraintes temporelles, on distingue deux types de structures : les structures de points et les structures d’intervalles. Nous avons retenu les structures d’intervalles car elles permettent de modéliser des notions comme la durée et elles sont conformes à la modélisation temporelle du comportement du procédé pour la détection (Toguyeni, et al., 1997).

Une structure d’intervalle nous permet de définir trois types de contraintes temporelles : la date, la période et la durée. La date Δt (Figure II-21a) permet de modéliser de manière certaine le temps séparant deux événements. La période P (Figure II-21b) permet de modéliser avec un degré d’incertitude le temps séparant l’occurrence de deux événements. La durée (Figure II-21c) sert à modéliser qu’une information est vraie à partir d’une date Δt et reste vraie sur tout l’intervalle de temps associé. Dans ce dernier cas on ne peut plus vraiment parler d’événement. On caractérise soit l'état d'un signal logique sur une période donnée soit la non occurrence d'un événement sur une période.



a) Date Δt = t2 - t1 b) P = [t2, t3] c) d = (Δt = t2 - t1 V = [t2, t3])

Figure II-21 – Entités temporelles utilisées dans les contraintes

L’avantage de l’approche est la relative simplicité d’écriture des STC à partir par exemple d’une Analyse des Modes de Défaillances, de leurs Effets et Criticité (AMDEC) (Guerraz, et al., 2004). La limitation majeure dans ce cadre, est d’abord de garantir de la complétude de la connaissance anormale ainsi encodée. Le deuxième niveau de difficulté est la cohérence des contraintes temporelles relative à une STC. Comme les contraintes sont définies entre deux événements, l'ensemble des contraintes peut être incohérent. Par conséquent, nous avons emprunté au concept de chronique, l'utilisation d’un graphe temporel pour vérifier la cohérence en établissant pour chaque STC son graphe temporel minimal complet (Dousson, et al., 1993) (Toguyeni, et al., 1990). La Figure II-22 présente le graphe temporel de la STC donnée par l’équation (II-1). C’est le graphe temporel obtenu en modélisant les contraintes de l’utilisateur. Dans un graphe temporel, chaque événement est représenté par un état et chaque contrainte temporelle par un arc orienté entre deux états. Dans la Figure II-22 les contraintes sont exprimées par des intervalles de temps.

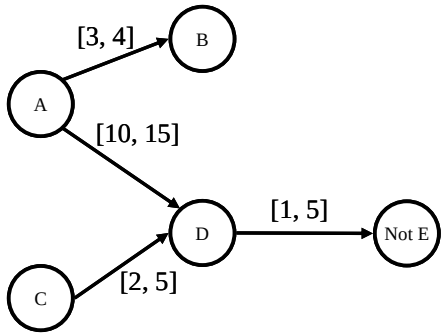


Figure II-22 – Graphe temporel d'une STC d'après les contraintes de l'utilisateur

Un troisième niveau de difficulté réside dans l'interprétation des événements en entrée du système de diagnostic dans le cadre de l'hypothèse de défaillances multiples. Le problème posé dans ce cadre, est la possibilité qu'une séquence d'événements produits par plusieurs défaillances entraine un surdiagnostic, c'est à dire l'isolation de défaillances supplémentaires.

Conclusion du chapitre II

Dans ce chapitre nous avons présenté des outils formels de modélisation du comportement des systèmes discrets. Nous avons introduit les automates à états finis et les automates temporisés. Ensuite. Nous avons présenté les RdP temporisés et les RdP temporels. Dans un second temps, nous avons présenté des outils pour le diagnostic des systèmes discrets. Nous avons présenté les observateurs, ensuite le diagnostiqueur de Sampath, après, le vérificateur de Jiang et enfin nous avons présenté les Signatures Temporelles Causales. Nous allons utiliser ces outils dans les prochains chapitres pour présenter les techniques de diagnostic que nous proposons dans le cadre de ce travail de thèse.

III Diagnosticabilité pour la réduction de l'explosion combinatoire des SEL

Introduction du chapitre III

Dans le chapitre précédent, nous avons choisi les automates temporisés (AT) comme outil de modélisation des systèmes embarqués logiques (SEL). Ce chapitre, est une étude ayant pour objectif d'identifier une méthode qui soit bien adaptée à notre problématique, modéliser un SEL sans subir l'explosion combinatoire des modèles manipulés. L'un des objectifs de cette thèse étant d'adapter l'approche diagnostiqueur au diagnostic des SEL, nous recherchons donc une méthode pour étudier leur diagnosticabilité. La diagnosticabilité est une capacité qui est établie hors-ligne et qui mesure l'aptitude de système à diagnostiquer toute faute d'un système dans un temps fini. Pour étudier la diagnosticabilité des SEL, nous allons commencer par nous intéresser à la diagnosticabilité des systèmes à événements discrets (SED). Cela peut sembler surprenant dans la mesure où le choix du formalisme AT fait que les SEL doivent être considérés plutôt comme des systèmes temporisés. Dans la littérature, il y a plus de travaux relatifs à la diagnosticabilité des SED qu'à celle des systèmes temporisés. Les travaux concernant les systèmes temporisés s'intéressent aujourd'hui à adapter des résultats des SED au contexte des systèmes temporisés. C'est la stratégie que nous allons mettre en œuvre dans le chapitre 3 et le chapitre 4.

Aussi le chapitre comprendra deux grandes parties. Dans la première partie nous allons d'abord définir formellement le concept de diagnosticabilité. Nous proposerons ensuite une synthèse des techniques proposées dans la littérature pour tester hors ligne la diagnosticabilité de SED. Nous présentons les différentes méthodes de vérification de la diagnosticabilité, celles de (Sampath, et al., 1995), de (Yoo, et al., 2002) et de (Cassez, et al., 2008), l'objectif étant d'identifier une technique qui soit adaptable à notre problématique. Dans la deuxième partie du chapitre nous focaliserons notre étude sur les méthodes de diagnosticabilité réparties. La répartition est une méthode pour réduire la complexité. Elle s'apparente à la décomposition d'un problème en sous-problèmes. Dans le cadre de la diagnosticabilité, nous montrerons que la capacité de répartition est fortement liée à la nature de l'observabilité que l'on a du comportement des composants d'un système. Le travail original de Sampath est qualifié aujourd'hui d'approche centralisée ou monolithique. Nous verrons qu'il existe différentes approches de répartition : les approches décentralisées (Debouk, et al., 2000), décentralisées conditionnelles, distribuées (Sengupta, 1998) ou modulaires (Debouk, 2003) (Contant, et al., 2006). Nous proposerons une vision critiques des différentes approches en dégageant leurs avantages et inconvénients afin de faire un choix de méthode que nous jugerons le mieux adapté à combattre l'explosion combinatoire des SEL.

III.I Diagnosticabilité de Sampath (Sampath, et al., 1995)(Sampath, et al., 1996) : Approche centralisée

Soit un système modélisé par un automate à états finis G=(Q, q0, Σ, δ). Soit L(G) le langage généré par G. Ce langage est supposé vivant et il ne doit pas exister dans G de cycles composés uniquement d'événement inobservables. La propriété de vivacité signifie qu'à partir de chaque état q de G, il existe au moins une transition franchissable. L'absence de cycles d'événements inobservables permet de garantir qu'on dispose d'observations régulières pour observer les effets d'une faute.

Définition III-1 : Diagnosticabilité globale (Sampath, et al., 1995)

L(G) est diagnosticable par rapport à la projection P : $\Sigma \rightarrow \Sigma_0$ et par rapport à la partition Π_f de Σ_f, si la condition suivante est satisfaite :

$$\left(\forall i \in \prod_f\right)(\exists n_i \in \mathbb{N})(\forall s \in \varphi(\sum_{fi}))(\forall t \in L/s)(\ \|t\| \geq n_i => D) \qquad \text{(III-1)}$$

Avec la condition de diagnosabilité D / $w \in P_{L(G)}^{-1}\left(P(st)\right) => \sum_{fi} \in w$

- $\varphi(\sum_{fi}) = \{s\sigma \in L(G)/\ \sigma \in \sum_{fi}\}$ *est l'ensemble des traces se terminant par une faute de Σ_{fi},*
- *L/s désigne l'ensemble des séquences suffixes de s dans L(G),*
- $\|t\|$ *désigne la longueur de la séquence t,*
- $P_{L(G)}^{-1}(u)$ *désigne toutes les traces v dans L(G) telles que P(v)=u.*

Remarque : Nous avons donné à la définition de Sampath le qualificatif de diagnosticabilité globale par opposition aux notions de diagnosticabilité modulaire ou de diagnosticabilité locale que nous verrons dans les paragraphes suivants.

La Définition III-1 amène deux remarques. D'abord le fait que l'on suppose qu'après l'occurrence d'une faute, on dispose de suffisamment d'événements pour détecter l'occurrence d'une défaillance. On peut noter toutefois dans cette définition, qu'il n'est pas précisé que les événements doivent être des événements observables. La deuxième remarque que l'on peut faire est que le nombre d'événements nécessaires n'est pas forcément le même pour chaque type de faute. Par contre, si le système est diagnosticable il est clair que ce nombre d'événements est borné par le nombre d'événements générés par le système excepté les fautes.

La Figure III-1 donne un exemple de système G avec son diagnostiqueur G_d. Dans cet exemple, on suppose que le système peut être affecté par trois fautes : σ_{f1}, σ_{f2} et σ_{f3}. Dans le cadre de la construction du diagnostiqueur G_d, on a défini deux types de fautes : $\Sigma_{f1}=\{\sigma_{f1}, \sigma_{f2}\}$ et $\Sigma_{f2}=\{\sigma_{f3}\}$. L'ensemble des événements observables est donné par $\Sigma_o=\{\alpha,\beta,\gamma,\delta\}$. Le système G est diagnosticable. Comme σ_{f1} et σ_{f2} appartiennent au même type, nous pouvons voir

qu'après l'occurrence de σ_{f1}, on a forcément l'occurrence de l'un des deux événements observables γ, δ. Donc le type Σ_{f1} est diagnosticable avec $n_1=2$ car il suffit de l'occurrence de deux événements {u, γ}, { σ_{f2}, γ}, ou { σ_{f3}, δ} après σ_{f1} pour diagnostiquer une faute de ce type. Le type Σ_{f2} est diagnosticable avec $n_2=1$ car il faut l'occurrence de δ pour diagnostiquer la faute de ce type.

Il est important de noter que la diagnosticabilité d'un système dépend bien entendu de l'observabilité que l'on a du système, mais également du partitionnement des fautes en différents types. Considérons toujours le système G de la Figure III-1, mais supposons cette fois-ci que l'on définisse trois types de fautes : $\Sigma_{f1}=\{\sigma_{f1}\}$, $\Sigma_{f2}=\{\sigma_{f2}\}$ et $\Sigma_{f3}=\{\sigma_{f3}\}$ en conservant le même ensemble d'événements observables $\Sigma_o=\{\alpha,\beta,\gamma,\delta\}$. Le fait d'avoir créé un type indépendant pour σ_{f2} rend le système non diagnosticable. En effet, dans ce cas, lorsqu'on a l'occurrence de l'événement observable γ, on ne sait pas si on est passé de l'état q2 à l'état q3 par l'événement 'u' ou l'événement 'σ_{f2}'. Donc le type Σ_{f2} n'est pas diagnosticable et par conséquent le système n'est pas diagnosticable.

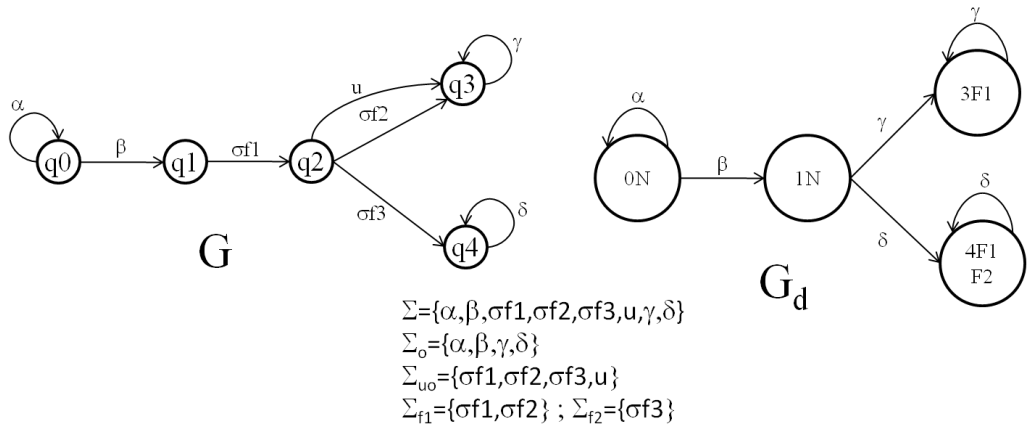


Figure III-1 – Exemple de système diagnosticable

Afin de comprendre l'influence de l'observabilité, considérons dans le cadre précédent, que 'u' est désormais un événement observable. Dans ce cas, on peut distinguer δ(q2, u)=q3 et δ(q2, σ_{f2})=q3. Ainsi si on a 'γ' sans avoir 'u', c'est que l'on a eu σ_{f2}. Dans ce cas le type Σ_{f2} devient diagnosticable avec $n_2=1$ et le système G est diagnosticable.

En pratique pour vérifier la diagnosticabilité, il existe trois méthodes : la méthode de Sampath basée sur la construction du diagnostiqueur du système (Sampath, et al., 1995), la méthode de Yoo basée sur la construction d'un vérificateur (Yoo, et al., 2002) et la méthode de Cassez basée sur le test du vide d'un automate de Büchi (Cassez, et al., 2008).

III.I.1 La méthode de Sampath

Cette méthode suppose avoir déjà construit le diagnostiqueur G_d du système. Dans ce cas, il est également nécessaire de construire ε(G), le modèle automate obtenu par suppression des

transitions de G associées à des événements inobservables. ε(G) n'est en général pas déterministe. Dans ce cas, L(G) est diagnosticable pour Σ_0 et Σ_f, si :

(i) Il n'existe pas de cycle d'états Fi-indéterminé dans G_d,
(ii) Il n'existe pas d'états ambigus dans G_d.

Un cycle d'états Fi-indéterminé de G_d est un cycle d'états Fi-incertains pour lequel on peut trouver un cycle d'états Fi-certains et un cycle d'états normaux dans ε(G) obtenus pour la même séquence d'événements observables.

Considérons le système G donné dans la Figure III-1. $\Sigma_o=\{\alpha, \beta, \gamma, \delta\}$ est l'ensemble des événements observables et $\Sigma_{f1}=\{\sigma_{f1}, \sigma_{f2}\}$ et $\Sigma_{f2}=\{\sigma_{f3}\}$ sont les deux types de fautes. Comme nous pouvons le constater, le diagnostiqueur G_d du système ne contient pas de cycle d'états Fi-indéterminé car il n'y a même pas de cycle d'états Fi-incertains. En effet, l'état 3F1 est un état F1-certain et l'état 4F1F2 est un état qui est à la fois F1-certain et F2-certain. On peut conclure sur la base de ce diagnostiqueur (Automate G_d de la Figure III-1) que le système est diagnosticable.

Dans l'exemple de la Figure III-2, nous supposons que seul σ_{f1} est non observable. Nous avons dans le diagnostiqueur un cycle d'états Fi-indéterminés car nous pouvons remarquer dans ε(G) que les états '3', '4' et '5' correspondent à un cycle de fautes alors que les états '7', '11' et '12' correspondent à un cycle normal aux séquences $(\beta\gamma\delta)^*$.

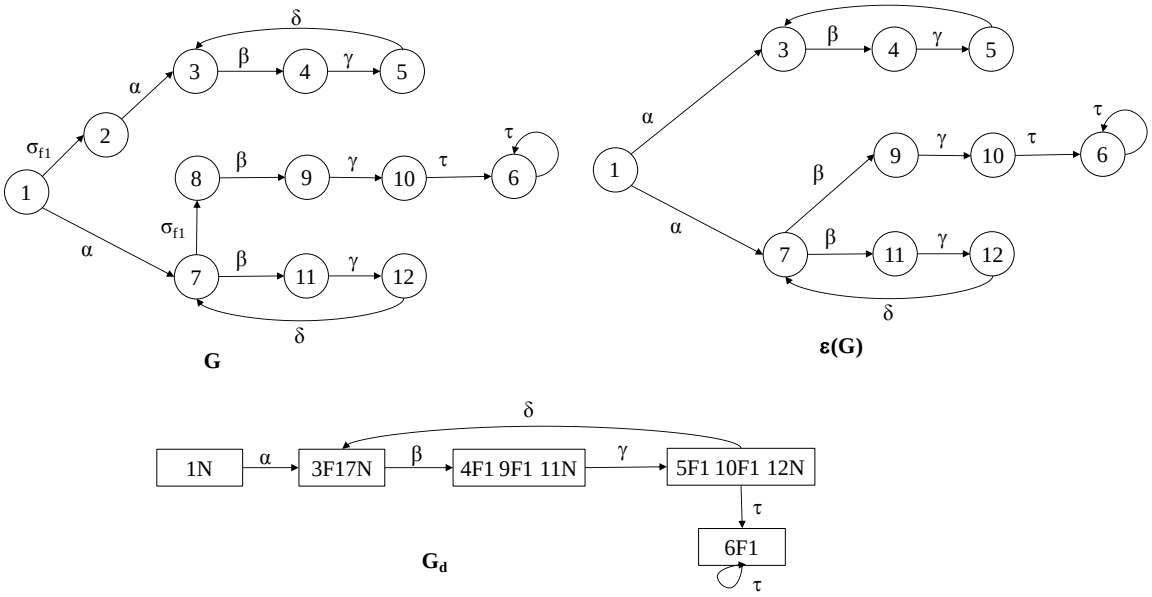
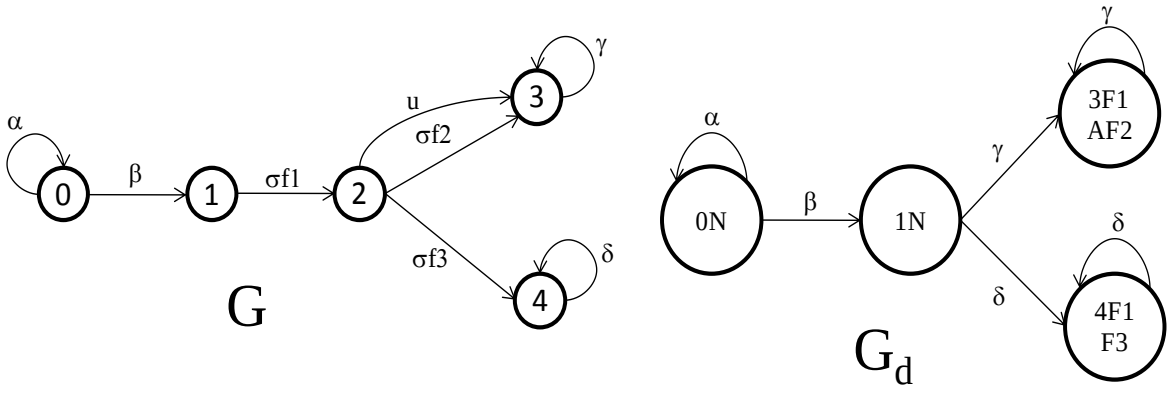


Figure III-2 – Cycles d'états Fi-indéterminés (Sampath, et al., 1995)

Un état $x \in G_d$ est dit ambigu si on peut trouver deux séquences s1 et s2 de L(G) avec $\Sigma_{fi} \in s1$ et $\Sigma_{fi} \notin s2$ tel que $\delta(q0, s1)=\delta(q0,s2) = q1$ et $q1 \in x$. Par construction, les successeurs de x dans G_d sont forcément des états ambigus.

Considérons à nouveau le système G de la Figure III-1 avec l'ensemble des événements observables $\Sigma_o=\{\alpha,\beta,\gamma,\delta\}$. Nous définissons les trois types de fautes: $\Sigma_{f1}=\{\sigma_{f1}\}$, $\Sigma_{f2}=\{\sigma_{f2}\}$ et

$\Sigma_{f3}=\{\sigma_{f3}\}$. Comme nous l'avons vu, l'événement 'u' n'étant pas observable comme la faute σ_{f2}, donc l'état '3' est un état ambigu par rapport au type de faute F2 (Etat AF2 de G_d dans la Figure III-3). Cela permet de conclure à la non diagnosticabilité du système G dans ce cadre, conclusion à laquelle nous étions arrivé informellement.



$\Sigma=\{\alpha,\beta,\sigma f1,\sigma f2,\sigma f3,u,\gamma,\delta\}$
$\Sigma_o=\{\alpha,\beta,\gamma,\delta\}$
$\Sigma_{uo}=\{\sigma f1,\sigma f2,\sigma f3,u\}$
$\Sigma_{f1}=\{\sigma f1\}$; $\Sigma_{f2}=\{\sigma f2\}$; $\Sigma_{f3}=\{\sigma f3\}$

Figure III-3 – Illustration des états ambigus et de la non diagnosticabilité

Remarque :

- La méthode 1 a pour conséquence que la capacité à construire le diagnostiqueur d'un système ne signifie pas que ce système soit diagnosticable. L'exemple de la Figure III-3 en est une parfaite illustration.

Bien qu'intéressante, la méthode de Sampath basée sur la construction d'un diagnostiqueur pose le problème de la diagnosticabilité des systèmes complexes. En effet, elle repose sur la construction a priori du diagnostiqueur du système surveillé. Dans le cas d'un système complexe, la taille du système ne permet pas toujours de construire le diagnostiqueur. D'où la nécessité de trouver d'autres méthodes.

III.I.2 La méthode de Yoo

La deuxième méthode de vérification de la diagnosticabilité d'un SED est basée sur la construction d'un vérificateur (Yoo, et al., 2002).

Définition III-2 : Fi-vérificateur

Soit $G_v^{Fi} = (Q_v^{Fi}, q_{0v}^{Fi}, \Sigma, \delta_v^{Fi})$ le Fi-vérificateur de G avec :

- $Q_v^{Fi} = Q \otimes L_i \otimes Q \otimes L_i$ *avec* L_i *l'ensemble d'étiquette relatif à Fi,* $L_i = \{N, Fi\}$
- $q_{0v}^{Fi} = (q_0, N, q_0, N)$ *l'état initial*
- δ_v^{Fi} *la fonction de transition est non déterministe et est définie par :*
  - $\forall \sigma \in \Sigma_{fi}$, $\delta_v^{Fi}((q,l,q',l'),\sigma) = \begin{Bmatrix} (\delta(q,\sigma), Fi, q', l') \\ (q, l, \delta(q',\sigma), Fi) \\ (\delta(q,\sigma), Fi, \delta(q',\sigma), Fi) \end{Bmatrix}$
  - $\forall \sigma \in \Sigma_{uo}/\Sigma_{fi}$, $\delta_v^{Fi}((q,l,q',l'),\sigma) = \begin{Bmatrix} (\delta(q,\sigma), l, q', l') \\ (q, l, \delta(q',\sigma), l') \\ (\delta(q,\sigma), l, \delta(q',\sigma), l') \end{Bmatrix}$
  - $\forall \sigma \in \Sigma_{o}$, $\delta_v^{Fi}((q,l,q',l'),\sigma) = (\delta(q,\sigma), l, \delta(q',\sigma), l')$

Définition III-3 : Fi-confusion d'un Fi-vérificateur

Un Fi-vérificateur est dit Fi-confondu s'il existe un cycle (q1,q2,…,qn) dans G_v^{Fi} tel que pour tout $q_i = (q^i, l^i, q^{i\prime}, l^{i\prime})$, $l^i = Fi$ et $l^{i\prime} = N$ ou inversement.

G_v^{Fi} est dit Fi-libre-confusion s'il ne contient pas de tels cycles.

Proposition III-1 : Test de diagnosticabilité par vérificateur

L(G) est diagnosticable pour Σ_0 et Π_f si et seulement si G_v^{Fi} est Fi-libre-confusion pour tout i dans Π_f.

III.I.3 La méthode de Cassez

La troisième méthode de vérification de la diagnosticabilité d'un SED repose sur la technique du test du vide sur un automate de Büchi (Cassez, et al., 2008). Elle est basée sur une notion duale de la notion d'états ambigus de la méthode de Sampath.

Définition III-4 : Non diagnosticabilité

Un système est non diagnosticable s'il peut générer deux séquences, une séquence fautive $s1 \in L(G)$ (i.e. $\exists u \in L(G)$ et $u \in \Psi(\Sigma_f)$, et $t \in L(G) \backslash u$ / $s1 = ut$) et une séquence non-fautive $s2 \in L(G)$, conduisant au même état ($\delta(q0,s1) = \delta(q0,s2)$) et avec la même projection sur l'ensemble des événements observables (i.e. $P_o(s1) = P_o(s2)$).

En effet, dans ce cas l'observation des événements observables ne permet pas de conclure sur l'occurrence de la faute ou pas. L'état atteint par les deux séquences s1 et s2 est en fait un état ambigu. Un exemple illustratif correspond à l'état 3F1A2 de la Figure III-3. En effet si on prend $s1 = (\alpha)^*\beta\sigma_{f1}\sigma_{f2}\gamma$ et $s2 = (\alpha)^*\beta\sigma_{f1}u\gamma$, on a $\delta(0,s1) = \delta(0,s2) = 3$ et $P_o(s1) = P_o(s2) = (\alpha)^*\beta\gamma$.

Aussi, pour mettre en œuvre le test du vide dans un automate de Büchi, il faut commencer par construire le produit synchronisé de deux automates G1 et G2 construits à partir de l'automate G modélisant le système.

G1 se comporte comme G mais mémorise quand une faute est survenue. Ainsi il est défini formellement par G1= (Q ⊗ {0,1}, (q0, 0), $\Sigma \setminus \Sigma_f$, δ_1) avec :

$$\forall \sigma_f \in \Sigma_f, \text{si } \delta(q, \sigma_f) = q' \text{ alors } \delta_1((q, n), \varepsilon) = (q', 1) \quad \textbf{(III-2)}$$

$$\forall \sigma \in \Sigma \setminus \Sigma_f, si\ \delta(q, \sigma) = q'\ alors\ \delta_1((q, n), \sigma) = (q', n) \quad \textbf{(III-3)}$$

L'état (q0,0) signifie que l'état initial est supposé exempt d'erreur. Ainsi l'équation (III-2) permet de mémoriser l'occurrence d'une faute au niveau de l'état (mise du label 1 au niveau de l'état destinataire) tout en effaçant l'événement de faute au niveau de la transition. Cet événement est remplacé par l'événement ε, symbolisant l'absence d'événement. L'équation (III-3) modélise la propagation du label d'un état par les transitions associées à des événements ne modélisant pas des fautes. La Figure III-4 illustre la construction d'un tel automate. Notons que les deux transitions étiquettes par ε entre les états <2,1> et <3,1> sont fusionnées pour la suite car on ne peut plus les distinguer.

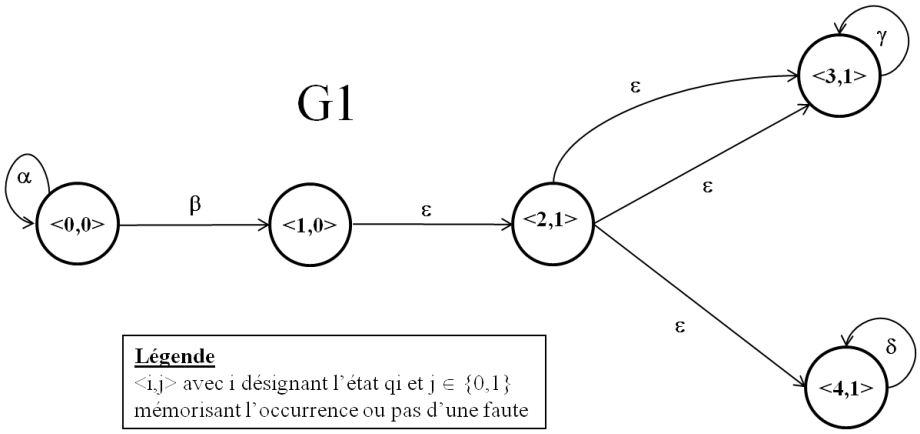


Figure III-4 – Automate G1 correspondant à l'automate G de la Figure III-1

G2 se comporte comme G mais supprime toutes les transitions associées à des fautes. Ainsi il est défini formellement par G2=(Q, q0, $\Sigma \setminus \Sigma_f$, δ_2) avec :

$$\forall \sigma \in \Sigma \setminus \Sigma_f, \text{si } \delta(q, \sigma) = q' \text{ alors } \delta_2(q, \sigma) = q' \quad \textbf{(III-4)}$$

La Figure III-5 illustre la construction d'un automate G2. On notera sur cet exemple que la suppression de la faute σ_{f1} a supprimé les autres états accessibles par comparaison avec l'automate G de la Figure III-1.

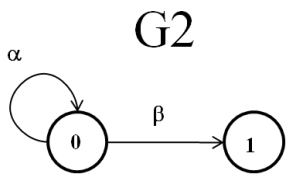


Figure III-5 – Automate G2 correspondant à l'exemple de la Figure III-1

Le produit $G1\|_{\Sigma_o}G2$ est synchronisé uniquement sur les événements observables appartenant à Σ_o. Les événements non observables sont renommés avant la synchronisation en leur faisant porter l'indice i de G_i. Soit $\delta_{1,2}$ la fonction de transition de cet automate.

Sur cette base, on construit l'automate B qui est une version étendue de $G1\|_{\Sigma_o}G2$. Le principe de l'extension consiste à introduire une variable permettant de mémoriser si G_1 a participé à la dernière transition de l'état correspondant atteint dans $G1\|_{\Sigma_o}G2$. Formellement il est défini par B = ((((Q⊗{0,1})⊗Q) ⊗{0,1}), (((q0, 0), q0),0), $\Sigma\backslash\Sigma_f$, δ_B, ϕ, R_B) avec :

$$\forall\sigma \in \Sigma\backslash\Sigma_f \text{ tq } \delta_{1,2}\big((q_1, n), q_2), \sigma\big) = (q_1', n'), q_2'),$$
$$\text{si } \delta_1(q_1, \sigma) = (q_1', 1) \; alors \; \delta_B\left(((q_1, n), q_2), z), \sigma\right) = ((q_1', n'), q_2'),1) \quad \textbf{(III-5)}$$
$$\text{sinon } \delta_B\left(((q_1, n), q_2), z), \sigma\right) = (((q_1', n'), q_2'),0)$$

$$R_B = \left\{\left(((q_1, 1), q_2), 1\right) \,\middle|\, ((q_1, 1), q_2) \in G1 \|_{\Sigma_o} G2\right\} \quad \textbf{(III-6)}$$

L'équation (III-5) permet de mémoriser lorsque G1 a participé à la dernière transition de $G1\|_{\Sigma_o}G2$. L'équation (III-6) définit l'ensemble des états répétés (i.e. un cycle) comme les états pour lesquels on a détecté une faute de Σ_f et pour lesquels G1 a participé à la dernière transition.

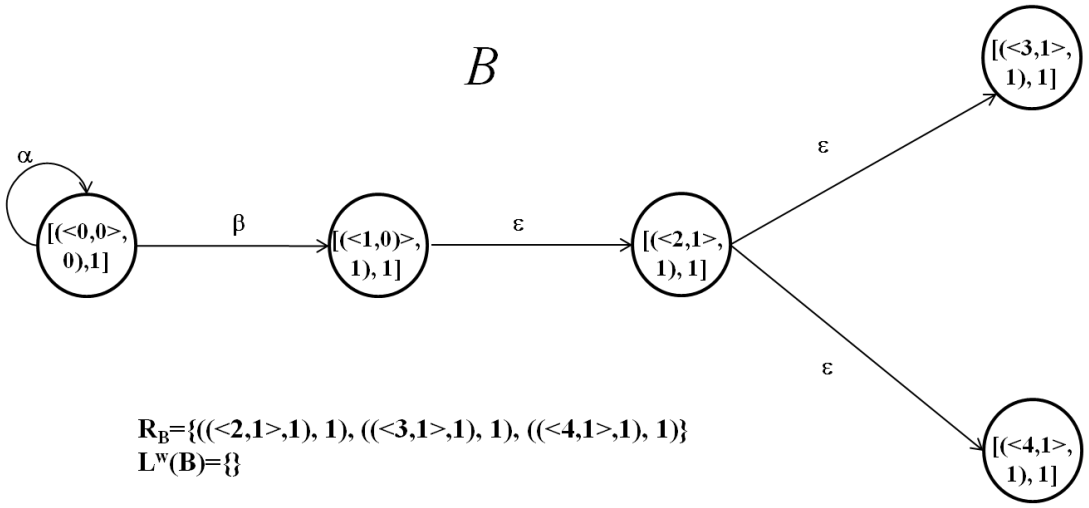


Figure III-6 – Automate de Büchi correspondant au système de la Figure III-1

Proposition III-2 : Diagnosticabilité par le test du vide d'un automate de Büchi (Cassez, et al., 2008)

G est diagnosticable si et seulement si B n'accepte pas de mots infinis ($L^w(B) = \emptyset$).

En effet, si B accepte un mot infini cela signifie que l'on peut trouver une séquence fautive de G1 suffisamment longue et une séquence non fautive de G2 qui ont la même projection sur l'ensemble des événements observables et conduisant toutes les deux dans un état de R_B. Pour avoir un mot infini dans un automate fini, il faut avoir un cycle dans cet automate. Comme ce cycle correspondrait à la fois à des états normaux et des états fautifs par construction de B, alors cela signifierait que dans le diagnostiqueur on aurait un cycle d'état Fi-indéterminé ou au moins un état ambigu. On voit donc que ce test est équivalent à la définition de Sampath.

La Figure III-6 illustre la construction d'un tel automate. Dans le cadre du système de la Figure III-1 et avec la définition des deux types de fautes $\Sigma_{f1}=\{\sigma_{f1}, \sigma_{f2}\}$ et $\Sigma_{f2}=\{\sigma_{f3}\}$, nous pouvons remarquer que les événements observables de γ et δ des états q3 et q4 de G ont été supprimés (Ils n'apparaissaient plus sur l'automate G2 de la Figure III-5). L'ensemble R_B contient les états ((<2,1>,1),1), ((<3,1>,1),1) et ((<4,1>,1),1). Comme il n'y a ni boucle d'événement observable sur ces états, ni cycle formé par ces états, on peut en conclure que $L^{\omega}(B)=\phi$. Donc, on peut en déduire que, sur la base de l'observabilité et de la définition du type de fautes, le système G est diagnosticable.

On trouve dans la littérature des algorithmes polynomiaux qui permettent d'effectuer le test du vide d'un automate de Büchi (Jiang, et al., 2001)(Yoo, et al., 2002).

III.I.4 Conclusion

L'un des inconvénients majeur de la proposition de Sampath est le risque d'explosion combinatoire engendré pour des systèmes complexes. Ces travaux initiaux ont donc fait l'objet de nombreuses extensions. Ces extensions sont liées soit à une réduction du problème initial en sous-problèmes plus simples à résoudre, soit à un changement de formalisme.

Au niveau de la réduction de la complexité par décomposition en sous-problèmes, les différentes extensions concernent la diagnosticabilité décentralisé (Debouk, et al., 2000) ou la diagnosticabilité modulaire (Contant, et al., 2006)(Debouk, et al., 2002). L'approche de Sampath est dans ce contexte identifiée comme une approche centralisée. Le contexte de diagnosticabilité introduit ici est alors qualifié de diagnosticabilité globale ou monolithique par opposition à la diagnosticabilité modulaire (Contant, et al., 2006).

Au niveau extension à d'autres formalismes, il s'agit notamment de l'extension aux automates temporisés (Tripakis, 2002) (Bouyer, et al., 2005b)(Cassez, et al., 2008) ou aux Réseaux de Petri (Genc, et al., 2003) (Liu, et al., 2012).

III.II Approches décentralisées

Afin de réduire la complexité de l'approche centralisée de Sampath, l'idée est de construire plusieurs diagnostiqueurs travaillant sur un sous-ensemble des événements observables. De manière individuelle, chaque diagnostiqueur a une observabilité réduite qui ne lui permet donc pas de diagnostiquer toutes les fautes du système. Les diagnostiqueurs sont synchronisés par un coordonateur qui doit permettre à l'ensemble des diagnostiqueurs plus le coordonateur, d'avoir la même diagnosticabilité que dans le cas centralisé (ou global).

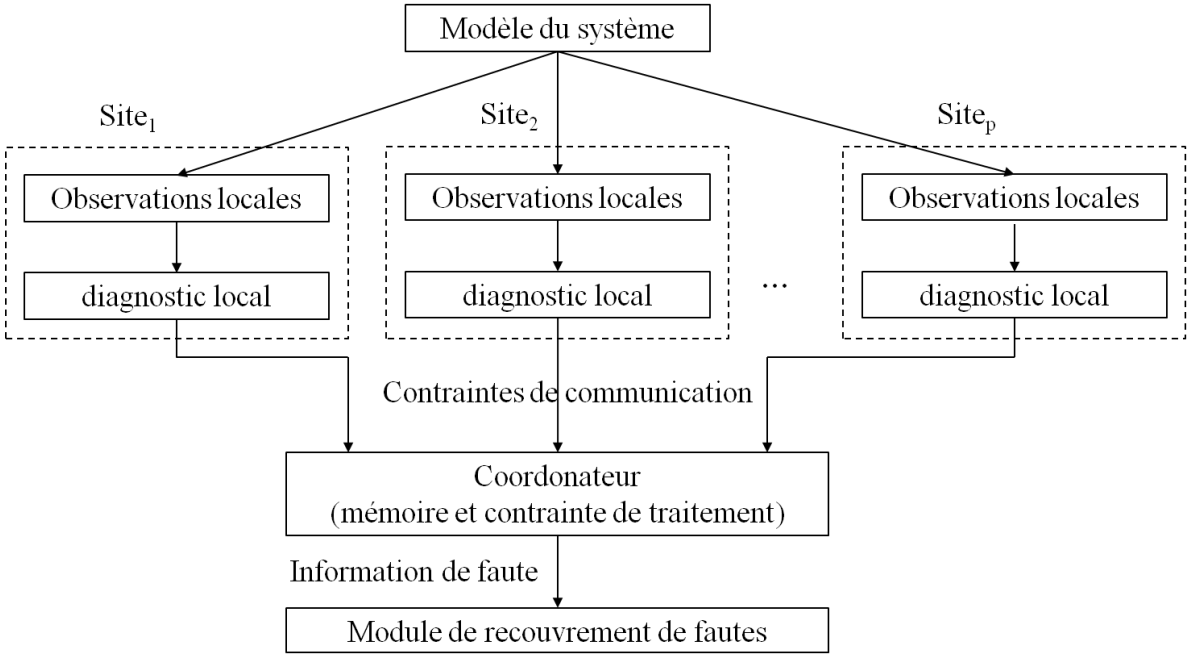


Figure III-7 – Architecture d'un système décentralisé (Debouk, et al., 2000)

L'architecture d'un système décentralisé est résumée par la Figure III-7. Dans cette architecture le système est observé par p sites évoluant en parallèle. Chaque site a une observation locale (ou réduite) du système Σ_o^i avec $i \in [1..p]$. Cette observabilité locale est telle que l'ensemble des observabilités locales est équivalent à l'observabilité globale du système i.e. $\Sigma_o = \bigcup_{i=1}^{p} \Sigma_o^i$. Il est à noter que deux sites peuvent avoir des événements observables communs (les Σ_o^i ne constituent pas une partition de Σ_o).

De manière formelle, on associe une projection P_i à chaque site définie de la manière usuelle de $\Sigma^* \rightarrow (\Sigma_o^i)^*$. Le diagnostiqueur local à chaque site G_{di} est construit par rapport à Σ_o^i et à la partition Π_f des types de fautes. En termes de fonctionnement chaque diagnostiqueur local G_{di} interprète les événements observables qu'il reçoit et transmet des informations de diagnostic comme son état courant au coordonateur. Cette transmission est effectuée selon un protocole de communication qui est une des composantes essentielles du système de diagnostic. Il est important de noter ici qu'en aucun cas les sites ne communiquent directement entre eux. Le coordonateur applique alors des règles de décision qui lui permettent de calculer l'état du

système par rapport à la partition des fautes Σ_f. En cas de détection d'une faute, il transmet l'information au module de recouvrement des fautes.

Les principales hypothèses de ce travail sont :

H1 : Le langage L(G) est vivant,

H2 : G n'a pas de cycle d'événements inobservables par rapport à chaque ensemble d'événements observables Σ_o^i,

H3 : L(G) n'est pas diagnosticable par rapport à P_i et Π_f,

H4 : Chaque site connait les événements observables par les autres sites.

L'hypothèse H3 est évidente. En effet, si un L(G) était diagnosticable par rapport à P_i et Π_f, le diagnostiqueur G_{di} du site i serait suffisant pour diagnostiquer les défaillances du système et on retomberait dans le cadre de l'approche centralisée. L'hypothèse H4 est utilisée par le protocole de communication pour que le site i signale au coordonateur s'il doit regarder l'information donnée par d'autres sites j≠i par rapport à un événement observable. Les hypothèses H1 et H2 correspondent à des hypothèses du même type effectuées dans le cas centralisé. On peut toutefois noter que l'hypothèse H2 qui servira dans la définition de la diagnosticabilité est plus dure que dans le cadre centralisée. En effet, G peut ne pas avoir de cycles d'événements inobservables par rapport à Σ_o, mais en avoir par rapport à la définition d'un sous ensemble d'événements observables Σ_o^i donné. C'est donc une condition necessaire qu'il faut vérifier lors de la spécification de l'observabilité des différents sites.

Soit *C*, les informations de diagnostic du coordonateur.

Définition III-5 : Fi-certain

L'information de diagnostic du coordonateur est dite F_i-certain, si à l'aide de C, le coordonateur est certain qu'une faute du type F_i s'est produite.

Sur cette base, le concept de diagnosticabilité « décentralisée » est défini.

Définition III-6 : Diagnosticabilité décentralisée (Debouk, et al., 2000)

Un langage L(G), préfixe-clos et vivant, est diagnosticable par rapport à un protocole, un ensemble de projections P_i avec $i \in [1..p]$ et une partition Π_f des fautes Σ_f, s'il vérifie la condition suivante :

$$(\forall i \in \Pi_f)(\exists n_i \in \mathbb{N})(\forall s \in \varphi(\Sigma_{fi}))(\forall t \in L(G)/s)\ (\|t\| \geq n_i \Rightarrow C\ est\ F_i - certain) \quad \text{(III-7)}$$

En d'autres termes, un système est diagnosticable si le coordonateur est capable à l'aide de ses informations de diagnostic de diagnostiquer toute faute dans un délai fini après l'occurrence de cette dernière.

Comme souligné auparavant, les deux points clés d'une approche décentralisée sont le protocole de communication mis en place pour la communication entre les sites et le coordonateur, et les règles de décision du coordonateur qui dépendent de ce protocole. Pour illustrer cela, considérons l'exemple du système donné par la Figure III-8. Ce système est caractérisé par les événements Σ={a, b, c, d, e, σ_{f1}}. σ_{f1} est une faute et constitue le seul événement non observable. Nous supposons que le système est surveillé par deux sites : le site 1 caractérisé par l'ensemble d'événements observables Σ_o^1 ={a,c,d,e} et le site 2 caractérisé par l'ensemble d'événements observables Σ_o^2 ={b,d,e}. On peut noter que L(G) est vivant et qu'il n'y a pas de cycle d'événements non observables par rapport à Σ_o^1 et Σ_o^2.

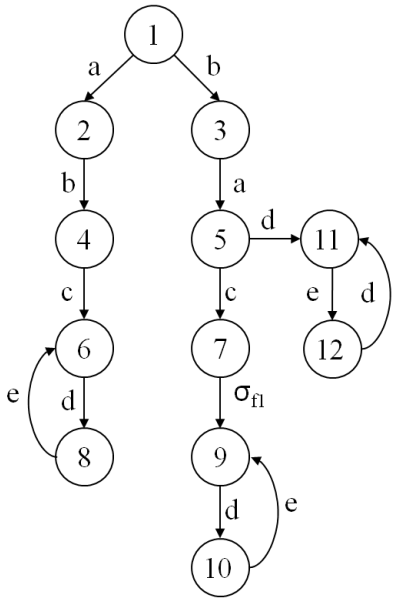


Figure III-8 – Système à diagnostiquer dans le cadre de l'approche décentralisée (Debouk, et al., 2000)

Sur la base de la définition des éléments observables des deux sites, on peut construire les deux diagnostiqueurs donnés par la Figure III-9.

Afin de montrer l'importance du protocole de communication entre les sites et le coordonnateur, et ses règles de décision, nous allons commencer par donner un contre-exemple qui constitue un protocole naïf. Nous supposons dans ce contre-exemple que lorsqu'il reçoit un événement observable, chaque diagnostiqueur calcule son état et le communique au coordonnateur. Si le diagnostiqueur sait que l'événement observé ne l'est également pas l'autre site, il met à 1 son flag SB_i pour signaler au coordonateur d'attendre la communication de l'autre site avant d'appliquer sa règle de décision. Le coordonateur dispose de trois registres nommés R_1, R_2 et SB. Un registre R_i contient le dernier état calculé et communiqué par G_{di}. Le registre SB permet au coordonateur de savoir s'il doit calculer immédiatement le nouvel état du système avec les dernières données reçues (cas où SB=0) où s'il doit attendre l'envoi de l'autre site (cas où SB=1). La nouvelle valeur de SB est calculée par la formule $SB=SB \oplus SB_i$ après utilisation des données émises par le site i. Initialement R_i, $i \in \{1,2\}$, a la valeur de l'état initial de G_{di} et $SB=SB_i=0$.

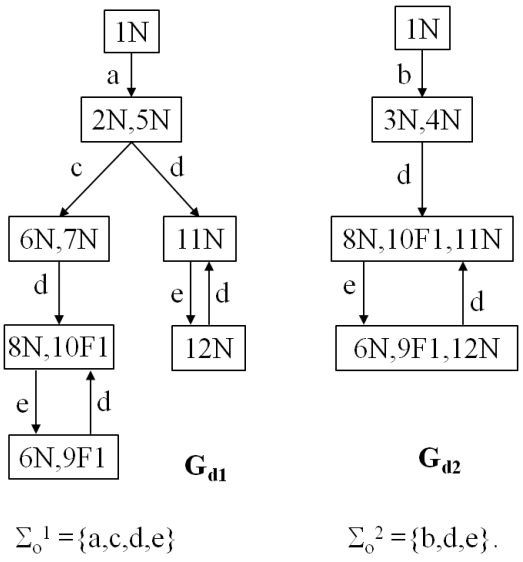


Figure III-9 – Diagnostiqueurs des deux sites surveillant le système de la Figure III-8

La règle de décision appliquée par le diagnostiqueur consiste à faire l'intersection de R_1 et R_2 pour obtenir l'état du système.

La Table III-1 illustre le diagnostic établi par le coordonateur dans le cas du protocole naïf que nous avons proposé précédemment. Dans cet exemple nous avons supposé que le système émet la séquence d'événements observables *'bacded'*. Nous pouvons nous rendre compte, que comme tenu de la règle de décision du coordonateur, certains des états calculés du système sont vides. Bien entendu, cela pose un problème car dans certains cas, on pourrait être dans un état défaillant du point de vue d'un diagnostiqueur centralisé alors que le coordonnateur n'est pas capable de le calculer, compte tenu du système de diagnostic construit. Dans cet exemple, on peut remarquer que dès la réception de la séquence *'bacd'*, on devrait être capable tout de suite de détecter l'occurrence d'une faute F1. On ne peut pas le faire à l'aide de l'envoi du site 1 uniquement en raison de l'asynchronisme des communications. Il faut attendre la réception de l'envoi du site 2 qui dans ce cas est supposé réagir moins vite que le site 1 pour avoir le calcul effectif de l'état fautif du système.

t ou événement	Site	R1	SB1	R2	SB2	SB	C	Nouveau SB
0	1 , 2	1N	0	1N	0	0	1N	0
b	2	1N	0	3N, 4N	0	0	{}	0
a	1	2N, 5N	0	3N, 4N	0	0	{}	0
c	1	6N, 7N	0	3N, 4N	0	0	{}	0
d	1	8N,10 F1	1	3N, 4N	0	0	{}	1
d	2	8N,10 F1	0	8N,10 F1, 11N	1	1	8N, 10 F1	0
e	2	8N,10 F1	0	6N, 9 F1, 12 N	1	0	{}	1
e	1	6N,9F1	1	6N, 9 F1, 12 N	0	1	6N, 9F1	0
d	1	8N,10 F1	1	3N, 4N	0	0	{}	1
d	2	8N,10 F1	0	8N,10 F1, 11N	1	1	8N, 10 F1	0

Table III-1 – Exemple de diagnostic décentralisé dans le cas du protocole naïf

Afin d'illustrer que le problème ici est lié au protocole et aux règles de décision du coordonateur, nous allons utiliser le protocole n°2 donné dans (Debouk, et al., 2000). Dans le cadre de ce protocole, lorsqu'un site i, reçoit un événement observable, il calcule le nouvel état atteint au sein de son diagnostiqueur, mais il calcule également l'ensemble des états potentiellement accessibles à partir des séquences d'événements inobservables de $\Sigma\backslash\Sigma^i_o$. Ce calcul est effectué en se basant sur le modèle du système. A l'état initial G_{d2} est dans l'état '1N'. En se basant sur le modèle du système, on recherche tous les états accessibles à l'aide des séquences s de $(\Sigma\backslash\Sigma^i_o)^*$. L'événement 'a' est un événement inobservable pour le site 2. Donc cet événement permet d'atteindre l'état 2 avec l'étiquette N car la séquence ne contient pas de faute. A partir de l'état 2, le seul événement possible est l'événement 'b' qui est observable du point de vue du site 2. Donc on arrête là la recherche des états potentiellement atteints à l'aide des événements inobservables du site 2.

Dernier rapport reçu de G_{d1}					**Dernier rapport reçu de G_{d2}**				
Règles	SB	SB1	*C*	Nouveau SB	**Règles**	SB	SB2	*C*	Nouveau SB
RD1	0	0	R1∩R4	0	**RD4**	0	0	R2∩R3	0
RD2	0	1	Attente	1	**RD5**	0	1	Attente	1
–	1	0	Impossible	-	–	1	0	Impossible	-
RD3	1	1	R1∩R2	0	**RD6**	1	1	R1∩R2	0

Table III-2 – Règles de décision du coordonateur dans le cadre du protocole n°2 de (Debouk, et al., 2000)

Le nouveau protocole reprend le principe de base du protocole naïf avec les mêmes fonctions pour les registres R1, R2, SB et le rôle des flags SBi associés à chaque site i. Mais il utilise deux nouveaux registres, R3 utilisé par le site 1 pour communiquer l'ensemble des états potentiellement accessibles par le système de son point de vue, et R4 utilisé par le site 2 pour la fonction duale.

Dans ce cas, les règles de décision du coordonateur sont modifiées. Le principe mis en place est que lorsque le site 1 (respectivement le site 2) communique au coordonateur ses informations de diagnostic, si le coordonateur ne doit pas attendre les données du site 2 (respectivement le site 1), l'état calculé du système (valeur de *C*) est égal à l'intersection entre l'état courant de G_{d1} (respectivement G_{d2}) et l'état potentiellement atteint calculé par le site 2 (respectivement le site 1). Ce fonctionnement correspond à la règle de décision RD1 (respectivement RD4) de la Table III-2. Sinon, lorsque les deux sites observent un même événement observable, les règles de décision (RD3 et RD6 dans la Table III-2), consistent à effectuer l'intersection entre les états courants des deux diagnostiqueurs.

La Table III-3 illustre l'application du protocole 2 dans le cadre du même scénario que celui utilisé lors de l'illustration du protocole naïf. On peut remarquer cette fois ci, que le protocole permet de calculer l'état du système au niveau du coordonateur après application des règles de décision. Et on peut remarquer aussi que dans le cadre de la séquence 'bacded', le

coordonateur est capable de détecter la faute σ_{f1}, dès que les deux sites ont observé la première occurrence de l'événement 'd' et qu'ils ont communiqué tous les deux leurs informations de diagnostic.

		Site 1			Site 2			Coordonateur			
t ou événement	Site	R1	R3	SB1	R2	R4	SB2	SB	Dri	C	Nouveau SB
0	1, 2	1N	1N,3N	0	1N	1N,2N	0	0	DR1	1N	0
b	2	1N	1N,3N	0	3N, 4N	3N,4N,5N,6N,7N	0	0	DR4	3N	0
a	1	2N, 5N	2N,5N,4N	0	3N, 4N	3N,4N,5N,6N,7N	0	0	DR1	5N	0
c	1	6N, 7N	6N,7N,9F1	0	3N, 4N	3N,4N,5N,6N,7N	0	0	DR1	6N, 7N	0
d	1	8N,10 F1	8N,10F1	1	3N, 4N	3N,4N,5N,6N,7N	0	0		pas encore calculé	1
d	2	8N,10 F1	8N,10F1	0	8N,10 F1, 11N	8N,10 F1, 11N	1	1	DR6	8N, 10 F1	0
e	2	8N,10 F1	8N,10F1	0	6N, 9 F1, 12 N	6N, 9 F1, 12 N	1	0		pas encore calculé	1
e	1	6N,9F1	6N,9F1	1	6N, 9 F1, 12 N	6N, 9 F1, 12 N	0	1	DR3	6N, 9F1	0
d	1	8N,10 F1	8N,10F1	1	3N, 4N	3N,4N,5N,6N,7N	0	0		pas encore calculé	1
d	2	8N,10 F1	8N,10F1	0	8N,10 F1, 11N	8N,10 F1, 11N	1	1	DR6	8N, 10 F1	0

Table III-3 – Exemple de diagnostic d'un système à l'aide du protocole n°2

Dans (Debouk, et al., 2000), les auteurs ont ainsi proposé trois protocoles (protocole 1, protocole 2 et protocole 3) permettant la mise en œuvre d'une architecture de diagnostic distribuée. Ils ont démontré que ces trois protocoles permettent à l'architecture décentralisée d'avoir la même diagnosticabilité que dans le cas centralisé.

Les trois protocoles proposés sont comparables d'un point de vue performance et simplicité de mise en œuvre. Le protocole 1 est le plus performant mais également le plus complexe à mettre en œuvre. Le protocole 3 est le moins performant mais le plus simple à mettre en œuvre. Ce dernier a notamment inspiré des extensions de ce travail comme la notion de codiagnosticabilité. Le protocole 2 que nous avons présenté ici, a des performances moyennes et une complexité intermédiaire.

En conclusion, nous pouvons toutefois souligner les principales limites de ce travail. Elles résident dans la complexité des protocoles proposés comme nous avons pu le constater au travers des exemples du protocole naïf et du protocole n°2. La complexité de certains protocoles conduit à munir le coordonateur de la capacité de mémorisation afin de pouvoir attendre les informations de diagnostic nécessaires à sa prise de décision. Nous l'avons illustré ici dans le cas du protocole 2 lors de la réception d'événements observables par plusieurs sites. Les conditions d'application de l'architecture décentralisée sont également plus contraignantes que celle de l'architecture centralisée. Nous avons évoqué par exemple l'hypothèse de travail H2, qui impose que le modèle du système ne comporte pas de cycles d'événements inobservables rapporté aux événements de chaque site. D'autre part, dans le cadre d'une mise en œuvre physiquement distribuée, d'autres hypothèses de travail imposent des contraintes sur les performances du réseau et notamment sur le respect de l'ordre de réception par le coordonateur des informations de diagnostic émises par les différents sites. Dans certains cas, cet ordre peut influencer le calcul de l'état du système.

III.III Codiagnosticabilité et codiagnosticabilité conditionnelle

Les travaux relatifs à la codiagnosabilité (Wang, et al., 2005)(Wang, et al., 2007)(Lafortune, et al., 2005) correspondent à une amélioration des travaux initiaux de Debouk sur le diagnostic décentralisé. Ils ont pour objectif de simplifier l'architecture proposée notamment en remplaçant le coordonateur par un centre de fusion plus simple, ne nécessitant pas de capacité de mémorisation. Cela necessite de revoir le fonctionnement des différents sites et notamment implique que les sites prennent des décisions locales. Ces décisions locales sont communiquées au centre de fusion qui agrège l'information pour prendre une décision globale. L'originalité de ce travail réside dans l'introduction d'une nouvelle forme de diagnostic : l'absence de faute. En effet, les sites ne se contentent pas de diagnostiquer s'il y a une faute. Ils peuvent également dire que jusqu'à une certaine évolution du système, il y a absence de faute dans le système.

La codiagnosabilité est la capacité qu'au moins l'un des sites i $\in$ [1..p] soit capable de diagnostiquer l'occurrence d'une faute si cette dernière a réellement affectée le fonctionnement du système, ou de diagnostiquer l'absence de faute dans le cas contraire.

Soit un système caractérisé par Σ un ensemble d'événements et une partition 'Σ_f' de ses fautes tel qu'il existe 'n' types de fautes. Ce système est surveillé par 'p' sites et chaque site i possède un nombre limité d'événements observables Σ_o^i. Chaque site est caractérisé par une fonction de projection P_i définie de la manière usuelle de $\Sigma^* \rightarrow (\Sigma_o^i)^*$.

On définit d'abord la fonction d'estimation des séquences Est(s) ayant la même projection qu'une séquence de référence donnée s.

Définition III-7 : Estimée d'une séquence

$$\forall\, i \in [1..p]\ \forall\, s \in \Sigma^*\ Est_i(s) = P_i^{-1}P_i(s) \cap L(G) \qquad \textbf{(III-8)}$$

Définition III-8 : F-codiagnosticabilité

Le langage L(G) est F-codiagnosticable par rapport à un ensemble de projections P_i avec $i \in$ [1..p] et une partition Π_f des fautes en plusieurs types Σ_{fj} avec $j \in$ [1..n] si :

$$\left(\forall j \in \prod_f\right)\left(\exists n_j \in \mathbb{N}\right)\left(\forall\, s \in \varphi\left(\sum_{fj}\right)\right)(\forall t \in L(G)/s \ \ t.q.\|t\| \geq n_j)(\exists i \in$$
1..p∀u∈Estist u est Fj−fautive **(III-9)**

La Définition III-8 implique que toute défaillance est diagnosticable par au moins l'un des p sites de surveillance locale. De même, on définit la diagnosticabilité de l'absence d'une faute par la NF-codiagnosticabilité.

Définition III-9 : NF-codiagnosticabilité

Le langage L(G) est NF-codiagnosticable par rapport à un ensemble de projections P_i avec i $\in$ [1..p] et une partition Π_f des fautes en plusieurs types Σ_{fj} avec j $\in$ [1..n] si :

$$(\forall j \in \prod_f)(\exists n_j \in \mathbb{N})(\forall s \notin \varphi(\sum_{fj}))(\forall t \in L(G)/s \;\; t.q. \|t\| \geq n_j \; et \; st \notin \varphi(\Sigma_{fj}))(\exists i \in 1..p \forall uv \in Estist \; t.q. \; Piu = Pis \; u \; n'est \; pas \; Fj-fautive \qquad \textbf{(III-10)}$$

La Définition III-9 signifie qu'un système est NF-codiagnosticable, si étant donné une séquence non fautive pour laquelle on peut trouver une séquence d'événements suffisamment longue également non fautive, alors il existe un site capable de diagnostiquer l'absence d'une faute par rapport à un type Σ_{fj} donné pour une séquence d'événements observables ($P_i(s)$).

Mais la F-codiagnosticabilité n'implique pas la NF-codiagnosticabilité et inversement. Ainsi certains types de défaillances peuvent être F-codiagnoticable et ne pas être NF-codiagnosticable. Aussi on peut définir $\Sigma_{f,F}$ le sous-ensemble de fautes de Σ_f dont on veut diagnostiquer la présence de fautes et $\Sigma_{f,NF}$ le sous-ensemble de fautes de Σ_f dont on veut diagnostiquer l'absence de fautes. Sur cette base on peut définir la codiagnosticabilité d'un système par rapport à une architecture décentralisée.

Définition III-10 : Codiagnosticabilité

Un langage L(G) est codiagnosticable par rapport à $\Sigma_{f,F}$ et $\Sigma_{f,NF}$ si :

1 - L(G) est F-codiagnosticable par rapport à $\Sigma_{f,F}$

2 - L(G) est NF-codiagnosticable par rapport à $\Sigma_{f,NF}$

3– $\Sigma_f = \Sigma_{f,F} \cup \Sigma_{f,NF}$

La Définition III-10 signifie qu'un système doté d'une architecture de surveillance décentralisée est codiagnosticable si toute défaillance du système peut être diagnostiquée comme étant présente ou absente.

Les auteurs ont par ailleurs prouvé que la codiagnosticabilité d'une architecture décentralisée implique la diagnosticabilité du système en centralisé, mais que l'inverse n'est pas vrai. Ils ont également montré que le test de la codiagnosticabilité peut être réalisé soit par les approches basées sur le diagnostiqueur (cf. §II.II.2), soit par la construction d'un vérificateur (Yoo, et al., 2002).

Considérons de nouveau l'exemple de la Figure III-8. On peut chercher à en vérifier la codiagnosticabilité. Nous reprenons les mêmes hypothèses que précédemment. Nous supposons que le système est surveillé par deux sites : le site 1 caractérisé par l'ensemble d'événements observables Σ_o^1 ={a,c,d,e} et le site 2 caractérisé car l'ensemble d'événements observables Σ_o^2 ={b,d,e}. Son fonctionnement peut être perturbé uniquement par la faute σ_{f1}.

Si on se rapporte à la Figure III-9, on peut constater que sur les deux diagnostiqueurs, le langage L(G) généré par le système n'est F-codiagnosticable par aucun des deux sites. En effet, G_{d1} possède le cycle d'états F1-indéterminé '8N, 10F1'-'6N, 9F1' qui correspond aux cycles 'q6'-'q8' exempt de faute et au cycle 'q9'-'q10' qui est exécuté suite à l'occurrence de σ_{f1}. De même, G_{d2} possède le cycle d'états F1-indéterminé '8N, 10F1, 11N'-'6N, 9F1, 12N'. Le système n'est pas non plus NF-codiagnosticable. Pour le montrer, considérons la séquence non fautive 'abc(de)*'. La présence du cycle infini d'événements '(de)*' nous amène à exécuter de manière répétitive le cycle d'états F1-indéterminé '8N, 10F1'-'6N, 9F1' par G_{d1}. De même, dans G_{d2}, '(de)*' nous amène à un cycle d'états F1-indéterminé '8N, 10F1, 11N'-'6N, 9F1, 12N'. Donc aucun des deux sites ne nous permet de conclure l'absence de la faute σ_{f1} après un nombre d'observations suffisants, on peut en conclure que le système n'est pas NF-codiagnosticable.

Considérons à présent la séquence non fautive 'ba(de)*'. G_{d1} aboutit avec cette séquence à la conclusion qu'il n'y a pas de faute car on arrive dans le cycle '11N-12N'. G_{d2} aboutit encore au cycle d'états F1-indéterminé donc il ne peut rien conclure. Cet exemple illustre l'intérêt de l'extension de la codiagnosticabilité au concept de codiagnosticabilité conditionnelle. Le principe de cette notion est de donner aux sites la capacité de donner des décisions locales conditionnelles du genre « absence de faute si un autre site ne dit pas le contraire ».

Définition III-11 : F-codiagnosticabilité conditionnelle

Le langage L(G) est conditionnellement F-diagnosticable par rapport à un ensemble de projections P_i avec $i \in [1..p]$ et une partition Π_f des fautes en plusieurs types Σ_{fj} avec $j \in [1..n]$ si :

$$\left(\forall j \in \prod_f\right)\left(\exists n_j \in \mathbb{N}\right)\left(\forall s \in \varphi\left(\sum_{fj}\right)\right)\left(\forall t \in L(G)/s \;\; t.q. \|t\| \geq n_j\right)\left(\exists i \in [1..p]\right)\Big(\forall uv \in Estist\ t.q\ Piu{=}Pis\ et\ uv \notin \varphi\Sigma fj\ \exists k\epsilon 1..p \forall xy \in Estkuv\ t.q\ Pkx{=}Pku\ x\ est\ non\ Fj - fautive \qquad \textbf{(III-11)}$$

La Définition III-11 implique que le site 'i' sait que : si la séquence $P_i(s)$ l'amène à diagnostiquer une séquence F_j-fautive, la possibilité que cette séquence ait été générée par une séquence non fautive jusqu'à un certain niveau (le préfixe 'u'), l'amène à supposer qu'un autre site pourrait diagnostiquer l'absence de faute par rapport à F_j jusqu'à 'u'.

Définition III-12 : NF-codiagnosticabilité conditionnelle

Le langage L(G) est conditionnellement NF-codiagnosticable par rapport à un ensemble de projections P_i avec $i \in [1..p]$ et une partition Π_f des fautes en plusieurs types Σ_{fj} avec $j \in [1..n]$ si :

$$\left(\forall j \in \prod_f\right)\left(\exists n_j \in \mathbb{N}\right)\left(\forall s \notin \varphi\left(\sum_{fj}\right)\right)\left(\forall t \in L(G)/s \;\; t.q. \|t\| \geq n_j\ et\ st \notin \varphi(\Sigma_{fj})\right)\left(\exists i \in [1..p]\right)\left(\forall uv \in Est_i(st)\ t.q.\ P_i(u) = P_i(s)\ et\ u \in \varphi(\Sigma_{fj})\right)\left(\exists k\epsilon[1..p]\right)(\forall w \in \qquad \textbf{(III-12)}$$

$Est_k(uv)$) w est $Fj-fautive$

La Définition III-12 est une définition duale de la Définition III-11. Elle signifie que pour toute séquence suffisamment longue 'st', le site i sait que même s'il a diagnostiqué la séquence 'st' comme non défaillante, il peut exister un site k pour lequel la séquence d'événements observables soit interprétée comme caractérisant l'occurrence d'une défaillance.

Définition III-13 : Codiagnosticabilité conditionnelle

Un langage L(G) est codiagnosticable conditionnellement par rapport à $\Sigma_{f,F}$ et $\Sigma_{f,NF}$ si :

1 - L(G) est conditionnellement F-codiagnosticable par rapport à $\Sigma_{f,F,}$

2 - L(G) est conditionnellement NF-codiagnosticable par rapport à $\Sigma_{f,NF,}$

3– $\Sigma_f = \Sigma_{f,F} \cup \Sigma_{f,NF}$

La Définition III-13 signifie qu'un système est conditionnellement codiagnosticable si toutes ses fautes peuvent être diagnostiquées localement comme étant présente ou absente sous réserve qu'un autre site n'ait pas une décision contraire.

Les auteurs ont montré que si un système est codiagnosticable, alors il est conditionnellement codiagnosticable. Ils ont également montré que si le système est conditionnellement codiagnosticable alors il est diagnosticable au sens centralisé. L'inverse n'est pas toujours vrai.

En résumé la décision globale peut être prise par la fonction de décision en s'appuyant sur Table III-4. Nous pouvons remarquer qu'une décision locale inconditionnelle est toujours prioritaire sur une décision conditionnelle.

Cas	Décision locale i	Décision locale k	Décision globale
1	Faute (F)	pas de décision	Faute (F)
2	Pas de faute (NF)	pas de décision	Pas de faute (NF)
3	F si aucun NF	pas de décision	Faute (F)
4	F si aucun NF	Pas de faute (NF)	Pas de faute (NF)
5	F si aucun NF	Faute (F)	Faute (F)
6	NF si aucun F	pas de décision	Pas de faute (NF)
7	NF si aucun F	Faute (F)	Faute (F)
8	NF si aucun F	Pas de faute (NF)	Pas de faute (NF)
9	pas de décision	pas de décision	pas de décision
10	Faute (F)	Pas de faute (NF)	conflit
11	F si aucun NF	NF si aucun F	conflit

Table III-4 – Fusion des décisions locales

Si nous reprenons notre exemple illustratif de la Figure III-8 avec ses deux diagnostiqueurs Figure III-9. L'utilisation de la diagnosticabilité conditionnelle consisterait pour le site 2 à

émettre la décision « Faute F1 si pas de NF » lors de l'observation de la séquence non fautive 'ba(de)*'. Le site 1 quant à lui, il diagnostiquerait « Pas de Faute (NF) ». La fonction de décision conclurait donc qu'il n'y pas de faute.

En conclusion, on peut constater que la théorie de la codiagnosabilité simplifie effectivement la prise de décision globale. On peut néanmoins être réservé sur l'utilisation pratique de cette théorie. En effet, comment un site met-il en œuvre réellement la codiagnosabilité conditionnelle ? Est-ce que le fait d'être dans un état normal d'un point de vue diagnostiqueur est suffisant pour dire que le site a diagnostiqué une absence de faute? D'autre part, il n'y a plus véritablement de protocole de communication entre les sites et la fonction de décision contrairement à ce que nous avons pu voir dans le cadre du protocole n°2 (Debouk, et al., 2000). La conséquence est que dans un contexte physiquement distribué, l'asynchronisme lié à la réception des décisions locales par la fonction de décision peut entrainer une instabilité par rapport à la prise de décision globale.

III.IV Approches modulaires

III.IV.1 Diagnosticabilité distribuée

(Sengupta, 1998) propose une architecture de diagnostic pour les systèmes physiquement distribués. Dans cette approche, on admet l'échange direct d'information entre sites. Il n'y a ni coordonateur, ni fonction de fusion. L'idée c'est que chaque site à l'aide des événements observables localement et des données échangées entre sites, est capable de diagnostiquer l'état du système par rapport à un sous-ensemble de fautes. Du fait de la distribution physique, il y a un partitionnement par site des événements observables et des fautes. Sous réserve de la capacité d'échanger des messages corrects entre sites, soit concernant l'observabilité d'événements $\Sigma_{o,m}^{i}$, soit concernant les fautes diagnostiquées $\Sigma_{f,m}^{i}$, l'approche distingue trois types de diagnosticabilité : la diagnosticabilité indépendante, la diagnosticabilité distribuée et la diagnosticabilité centralisée.

Remarque : Nous utilisons ici le terme de diagnosticabilité distribuée plutôt que celui de diagnosticabilité décentralisée utilisée dans les publications de Sengupta de manière à ce qu'il n'y ait pas de confusion avec la définition de la diagnosticabilité décentralisée proposée par Debouk (cf. § III.II).

La diagnosticabilité indépendante est la capacité d'un site à diagnostiquer toute faute surveillée par le site à l'aide des seuls événements observables localement.

La diagnosticabilité distribuée consiste en la capacité à diagnostiquer toute faute surveillée par un site en utilisant les événements observables localement et les données apportées par les messages émis par les autres sites. Nous voyons que de ce fait, la diagnosticabilité indépendante peut être considérée comme une diagnosticabilité distribuée sans besoin

d'échange de messages inter-sites. Donc, la diagnosticabilité indépendante implique la diagnosticabilité distribuée. L'inverse n'est pas vrai dans le cas général.

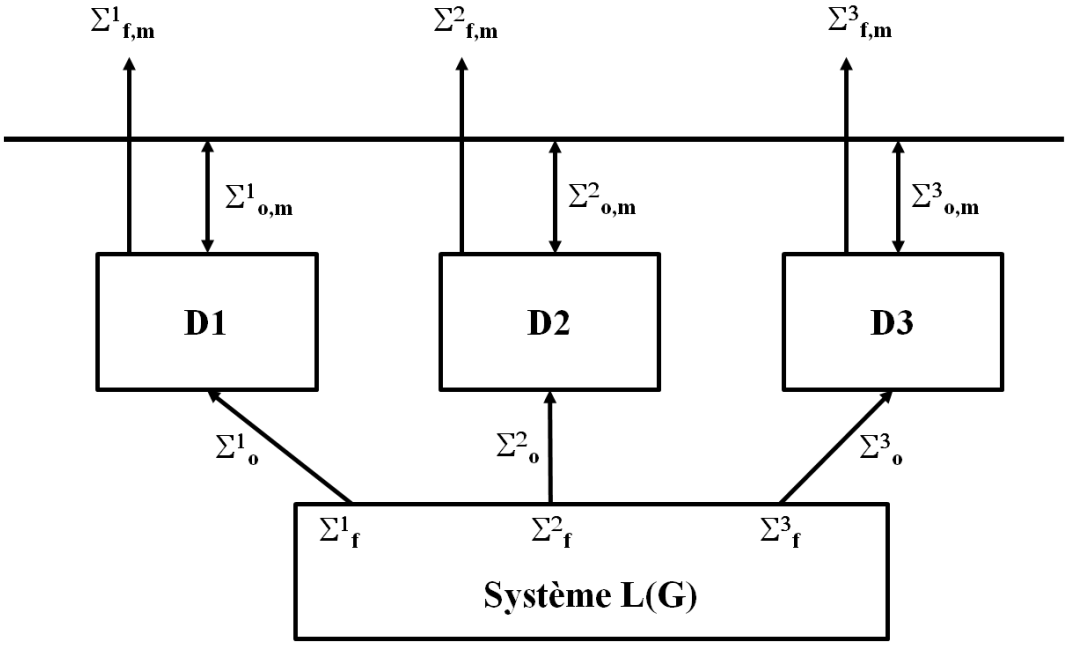


Figure III-10 – Architecture d'un système de surveillance basé sur la diagnosticabilité distribuée (Sengupta, 1998)

La diagnosticabilité centralisée consiste en la capacité à diagnostiquer toute faute du système par un diagnostiqueur unique qui observerait tous les événements observables émis par le système. En fait elle correspond à la diagnosticabilité globale de Sampath (cf. §III.I).

L'originalité de ce travail réside dans la proposition d'un cadre formel pour vérifier que le système d'échange de messages est correct. Il permet également de garantir la diagnosticabilité distribuée.

III.IV.2 Diagnosticabilité de Debouk

Faisant suite à ses travaux sur le diagnostic décentralisé, Debouk s'est intéressé à la définition d'une architecture pour le diagnostic modulaire. Elle est basée sur un partitionnement du système en plusieurs composants ou modules évoluant en parallèle. L'objectif est donc de pouvoir proposer une architecture de surveillance permettant en surveillant séparément chaque module d'avoir une diagnosticabilité équivalente au cas centralisé. Malheureusement, la diagnosticabilité des modules dans ce cadre ne garantie pas la diagnosticabilité du système. Pour que la diagnosticabilité des modules entraine celle du système, il faut garantir que chaque module produise suffisamment d'événements observables pour garantir l'observabilité de ses défaillances au niveau global.

Afin de comparer les approches de R. Debouk et de O. Contant (Contant, et al., 2006), nous avons qualifié ici la diagnosticabilité des modules de Debouk de diagnosticabilité locale. Ce qualificatif local est par opposition au qualificatif de global que nous prendrons souvent pour

qualifier la diagnosticabilité au niveau système au sens de Sampath. Il est également justifié par le fait que les définitions de l'observabilité et des fautes d'un composant est faite localement au composant.

L'approche modulaire proposée suppose qu'à chaque module i ∈ [1..p], on associe son modèle $G^i=(Q^i, q^i_0, \Sigma^i, \delta^i)$. Soit Σ^i_o l'ensemble des événements observables de G^i et Σ^i_{uo} son ensemble d'événements inobservables t.q. $\Sigma^i=\Sigma^i_o \cup \Sigma^i_{uo}$. Les événements de Σ^i sont supposés être locaux au module. Soit Σ^i_f son ensemble de fautes. $L(G^i)$ est supposé vivant et ne possède que des cycles constitués uniquement d'événements inobservables. A chaque module est associé un diagnostiqueur G_d^i qui effectue le diagnostic local des fautes du module. Chaque module est supposé localement diagnosticable au sens de Sampath (cf. §III.I). C'est-à-dire que l'occurrence de toute faute locale au module est diagnosticable par G_d^i par l'observation d'un nombre fini d'événements observables de Σ^i_o.

Le système G correspondant est obtenu par le produit synchrone des modèles des modules ($G=G^1||_sG^2||_s\ldots||_sG^p$) et est défini sur l'alphabet $\Sigma = \cup_{i=1}^{p} \Sigma^i$.

Définition III-14 : Vivacité du langage généré par un système par rapport au langage d'un de ses composants

Soit G un système, et G^i, $i \in [1..p]$ le $i^{ème}$ composant du système G. Un langage L(G) est vivant par rapport à $L(G^i)$, si $\forall s \in L(G)$, $P_i(s)$ est arbitrairement long avec P_i la projection de $(\Sigma)^* \rightarrow (\Sigma^i)^*$.

L'auteur propose alors une condition suffisante pour que la diagnosticabilité locale de chaque module garantisse la diagnosticabilité globale du système.

Proposition III-3 : Condition suffisante de diagnosticabilité globale du système

L(G) est diagnosticable par rapport à un ensemble d'événements observables Σ_o et une partition Π_f sur Σ_f, si $L(G^i)$, $i \in [1..p]$ est diagnosticable par rapport à l'ensemble des événements observables Σ^i_o et Π^i_f sur Σ^i_f, et L(G) est vivant par rapport à $L(G^i)$.

Mais l'auteur a complété cette proposition en proposant un test de diagnosticabilité système basé sur la construction d'un modèle de test :

$$G_{test}=G_d^1||_sG_d^2||_s\ldots||_sG_d^p \qquad \textbf{(III-13)}$$

Proposition III-4 : Test de diagnosticabilité

Le système est diagnosticable si G_{test} ne comporte pas de cycle d'états Fj-indéterminés par rapport à Σ_o et Π_f correspondant à un cycle Fj-indéterminé de G_d^i.

Afin de comprendre les raisons de la Définition III-14 et de la Proposition III-3, considérons l'exemple donné par la Figure III-11. G^1 et G^2 sont les modèles de comportement des deux composants du système considéré. L'événement 'f' de G^1 modélise une faute et il est le seul

événement inobservable du système. L'observabilité est locale. C'est-à-dire Σ^1_o={c,d,e} et Σ^2_o={a,b}.

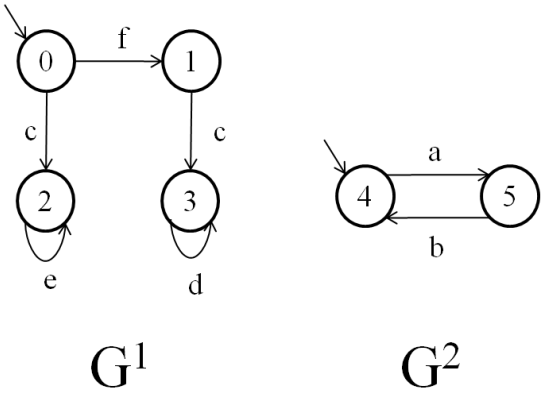


Figure III-11 – Composants d'un système

On peut donc sur cette base construire les diagnostiqueurs de ces deux composants (Figure III-12). Aucun de ces deux diagnostiqueurs ne contient de cycle d'états F1-indéterminé. Donc G^1 est diagnosticable par rapport à Σ^1_o et Π_f, et de même G^2 est diagnosticable par rapport à Σ^1_o et Π_f.

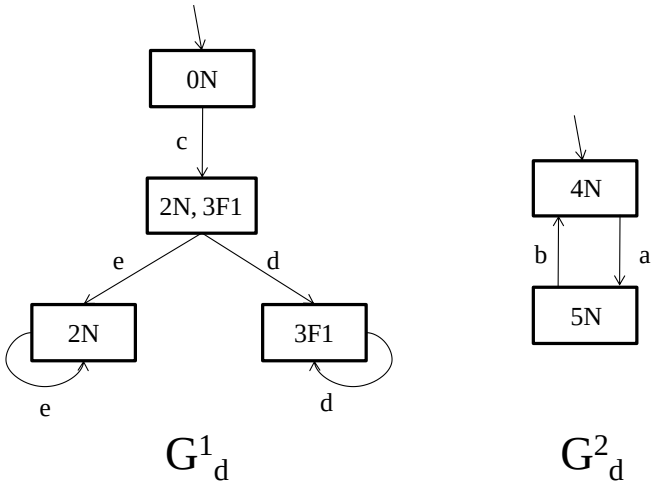


Figure III-12 – Diagnostiqueurs des deux composants de la Figure III-11

On pourrait donc penser que les deux composants étant chacun diagnosticable localement, le système serait également diagnosticable globalement, c'est-à-dire par rapport à $\Sigma_o = \Sigma^1_o \cup \Sigma^2_o$ et Π_f. Le modèle du système G est obtenu par le produit synchrone des modèles des composants (Figure III-13).

Le diagnostiqueur du système composite présente deux cycles d'états F1-indéterminé (Figure III-14). Comme on peut le constater sur le générateur $\varepsilon(G)$ du système (Figure III-13), ces cycles correspondent à des véritables cycles d'états normaux et des cycles d'états F1-fautifs. Donc le système est en fait non diagnosticable par rapport à l'observabilité globale Σ_o et Π_f.

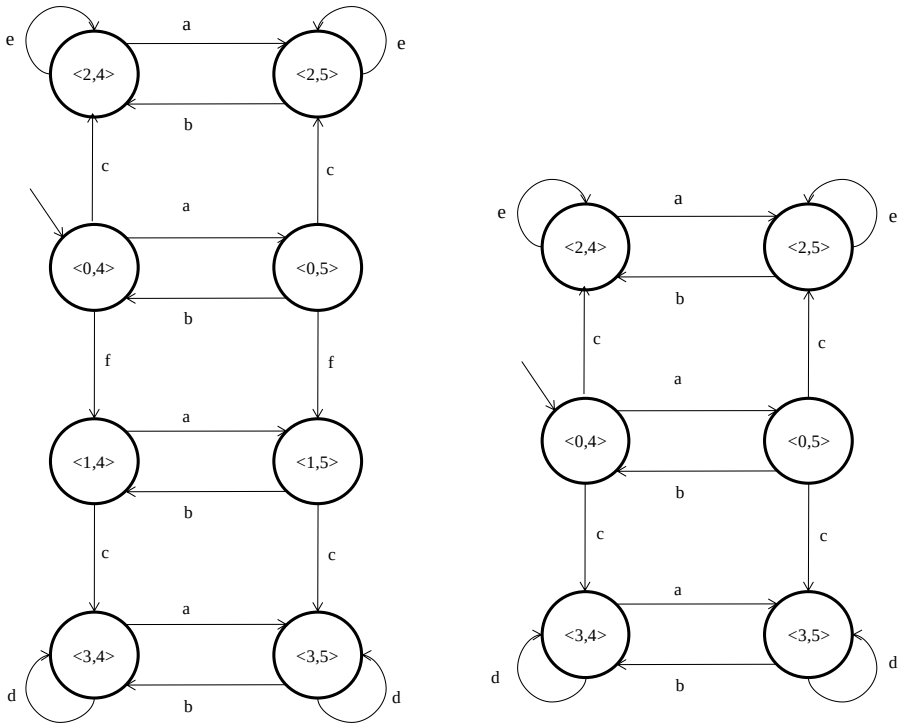


Figure III-13 – Modèle G du système composite et son générateur G'

Si on considère la séquence 'fc(ab)*', on peut comprendre comment on peut avoir de tels cycles d'états F1-indéterminés. En effet, la sous-séquence 'fc' est reconnu par le modèle du composant G^1. L'interprétation du seul événement observable 'c' de cette séquence par le diagnostiqueur G^1_d de ce composant conduit le système dans l'état F1-indéterminé (2N, 3F1) (Figure III-12). La sous séquence '(ab)*' est reconnue par le composant G2. Son interprétation par G^2_d amène à exécuter indéfiniment le cycle '4N–5N'. Le produit des états des deux diagnostiqueurs nous donne donc le cycle '(<2,4>N, <3,4>F1) - (<2,5>N, <3,5>F1)' qui est dû à l'alternance des événements 'a' et 'b' des deux composants. Le système n'est donc pas diagnosticable si jamais on admet que le composant 1 peut se contenter de produire le seul événement observable 'c' après 'f'. D'où d'après la Proposition III-3, la condition de diagnosticabilité du système est réduite à la condition que L(G) soit vivant par rapport à L(G^i) quelque soit i appartenant à [1..p]. Cela impose donc à chaque composant de produire suffisamment d'événements observables. Dans cet exemple, G n'est donc pas globalement diagnosticable car L(G) n'est pas vivant par rapport à L(G^1).

Discussion

D'un point de vue pratique, le contre-exemple proposé par l'auteur pose problème. En effet dans le cadre de ce contre-exemple, l'auteur a montré que la diagnosticabilité locale n'entrainait pas forcément la diagnosticabilité système car la séquence 'fc(ab)*' amène l'automate de test G_{test} dans un cycle d'états F1-indéterminé. Sur la base du modèle du système (Figure III-13), on peut l'admettre. Par contre, quand on regarde le modèle du composant G^1, on se rend compte, qu'il n'est pas compréhensible que ce sous-système

n'exécute que la séquence 'fc'. Dans la pratique, lorsque le fonctionnement d'un composant est affecté par une faute tel que 'f', cela entraine une modification de comportement qui se traduit en général par la non occurrence de certains événements correspondant au fonctionnement normal et à l'occurrence d'événements caractérisant les modes de défaillances du composant. On peut penser que dans la Figure III-11 suite à la séquence 'fc', on devrait avoir l'événement 'd' comme étant le mode de défaillance de la faute 'f'. Dans ce cas, le cycle F1-indéterminé ne serait jamais exécutable de manière infini. Donc le système serait en pratique toujours diagnosticable.

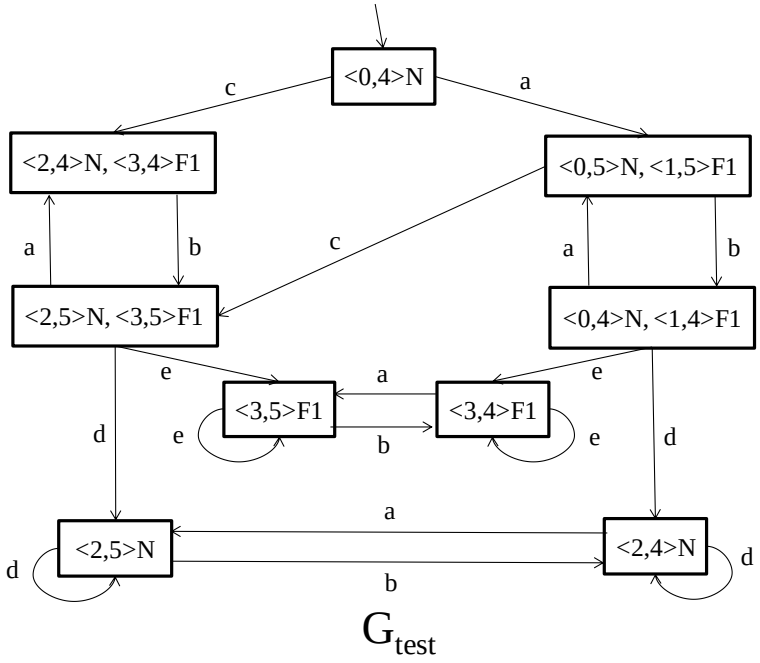


Figure III-14 – Automate de test du système composite

L'intérêt principal des travaux de (Debouk, et al., 2002) sur la diagnosticabilité modulaire est qu'ils montrent que la diagnosticabilité locale n'induit pas automatiquement une diagnosticabilité au niveau système. Ce résultat est surprenant dans le cadre des hypothèses des auteurs qui ont défini localement les observations et les ensembles de fautes caractéristiques d'un module. Dans ce cadre, la diagnosticabilité locale d'un module est équivalente à la diagnosticabilité indépendante proposée par Sengupta. On s'attend donc à ce que cela garantisse la diagnosticabilité du système. Ce n'est pas ce que révèlent ces travaux.

Les conditions de diagnosticabilité données par les auteurs sont des conditions dures. En effet, l'exigence de la vivacité de L(G) par rapport à $L(G^i)$ fait que cette approche est difficilement applicable en pratique dans le cas général. D'autre part, le test de diagnosticabilité système basé sur G_{test} est d'autant moins intéressant qu'il est comparable en termes d'explosion combinatoire à la construction du diagnostiqueur système G_d. Dans le cas des systèmes complexes, l'intérêt d'une approche modulaire est d'éviter la construction de ce diagnostiqueur qui est forcément coûteux en mémoire et en temps de traitement même si ici, la construction de l'observateur de test est faite hors ligne.

Finalement, nous pouvons nous interroger sur l'intérêt de vérifier la diagnosticabilité au niveau système dans ce cadre. En effet, l'objectif pratique est de pouvoir garantir que toute faute soit diagnostiquée. Nous pouvons dire que la diagnosticabilité locale de chaque composant d'un système composite le permet.

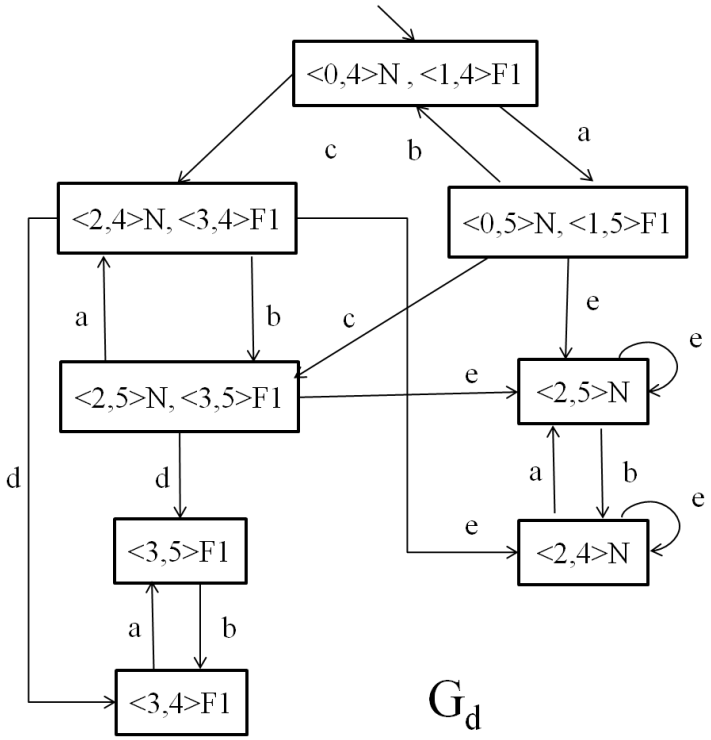


Figure III-15 – Diagnostiqueur de l'exemple de la Figure III-13

III.IV.3 Diagnosticabilité modulaire de Contant

L'idée est de définir une diagnosticabilité propre à un module. La notion de module a pour but de généraliser les notions de composants, sous-systèmes et système. Un module peut être n'importe lequel des constituants d'un système. Appelons donc composant, les constituants élémentaires d'un système. Soit p, le nombre total de ces constituants et T le système. Soit S $\subseteq$ T, un module de T. Nous supposons que le comportement de S est représenté par l'automate à états finis $G^s=(Q^s, q^s_0, \Sigma^s, \delta^s, Q^s_m)$. Les différents éléments de ce quintuplé ont les définitions usuelles indiquées au § II.I.1. Un module possède des événements observables appartenant à Σ^s_o et des événements inobservables appartenant à Σ^s_{uo}. Parmi ces événements inobservables nous distinguons les fautes qui appartiennent à Σ^s_f. Σ^s_{cm} représente les événements communs du module avec d'autres modules du système T. Nous supposons que les événements communs d'un module sont tous des événements observables de ce module ($\Sigma^s_{cm} \subseteq \Sigma^s_o$). Etant donné un composant de S ($z \in S$), $G^s=\|_{z\in S}G^z$. En d'autres termes, le comportement du module est obtenu par produit synchrone à partir du comportement de ses composants.

Définition III-15 : Diagnosticabilité modulaire

Etant donné T={1, ..., p} un système et S ⊆ T avec $G^S = ||_{z \in S} G^z$. Le langage $L(G^S)$ est modulairement diagnosticable par rapport à Σ^S_o et Σ^S_f, si $\forall i \in S$, $\forall f \in \Sigma^i_f$, $\forall s \in L(G^S)$ t.q. $s \in \varphi(\Sigma^i_f)$, $\exists n \in \mathbb{N}$ t.q. $\forall t \in L(G^S)/s$, $||P^i_o(t)|| \geq n \Rightarrow D(st) = 1$
Avec D(st) = 1 si $w \in (P^i_o)^{-1}(P^i_o(st)) \Rightarrow f \in w$, sinon D(st) = 0.

La Définition III-15 doit être comprise comme une forme de diagnosticabilité au niveau système. En effet, si nous considérons S comme étant composé de composants 'i', alors S est modulairement diagnosticable par rapport à l'ensemble de ses événements observables et l'ensemble de ses fautes si chaque faute d'un de ses modules est diagnosticable à l'aide uniquement d'événements observables de ce module. Nous pouvons constater que cette définition est différente de la diagnosticabilité globale définie par Sampath (cf.III.I). Dans le cadre de cette diagnosticabilité globale, il faut que toute faute du système puisse être diagnostiquée à l'aide des événements observables du système. On n'exige donc pas dans la diagnosticabilité globale que les fautes soient diagnosticables par les événements observables du module dans lequel la faute est apparue.

Aussi pour évaluer la diagnosticabilité modulaire, l'auteur a introduit un nouveau concept : la notion de cycle d'états F^{Mi}-indéterminés.

Définition III-16 : Cycle d'états F^{Mi}-indéterminé

Etant donné le diagnostiqueur de S ⊆ T, un cycle d'états F^{Mi}-indéterminé est un cycle d'états Fj-incertains par rapport au type de faute Fj du composant i et dont les transitions sont basées sur des événements observables du composant i.

Ces deux notions permettent de comprendre à présent l'exemple de R. Debouk (Figure III-13). Quand nous observons le diagnostiqueur G_d du système composite (Figure III-15), nous notons qu'il contient deux cycles d'états F1-incertains. Les transitions de ces cycles sont dues aux événements 'a' et 'b' qui sont des événements observables du composant G_2 (Figure III-11). Comme F1 est un type de faute du composant G_1, donc ces cycles ne sont pas F^{M1}-indéterminés. Donc le système G de la Figure III-13 est modulairement diagnosticable.

G^1 et G^2 étant localement diagnosticables, on pourrait en déduire que lorsqu'un système est modulairement diagnosticable alors ses composants sont localement diagnosticables. Ce n'est malheureusement pas vrai dans le cas général. Considérons à présent l'exemple de la Figure III-16. Le système T est constitué de trois composants dont les modèles sont respectivement donnés par G^1, G^2 et G^3. Le modèle du système G^T est obtenu par le produit synchrone des modèles des composants. Les caractéristiques événementielles de cet exemple sont les suivantes :

- $\Sigma=\{a,b,c,d,x,y,f1,f2\}$; $\Sigma_o=\{a,b,c,d,x,y\}$; $\Sigma_{uo}=\{f1,f2\}$; $\Sigma_f=\{f1,f2\}$
- Par rapport au système G^T : $\Sigma^T=\{b,d,f1\}$; $\Sigma^T{}_o=\{b,d\}$; $\Sigma_{uo}=\{f1\}$; $\Sigma^T{}_f=\{f1\}$

- Par rapport au composant G^1 : $\Sigma^1=\{a,c,d,f1,f2\}$; $\Sigma^1{}_o=\{a,c,d\}$; $\Sigma^1{}_{uo}=\{f1,f2\}$; $\Sigma^T{}_f=\{f1,f2\}$
- Par rapport au composant G^2 : $\Sigma^2=\{a,b,c,x,y\}$; $\Sigma^2{}_o=\{a,b,c,x,y\}$; $\Sigma^2{}_{uo}=\{\}$; $\Sigma^2{}_f=\{\}$
- Par rapport au composant G^3 : $\Sigma^3=\{a,b,c,x,y\}$; $\Sigma^3{}_o=\{a,b,c,x,y\}$; $\Sigma^3{}_{uo}=\{\}$; $\Sigma^3{}_f=\{\}$

Comme nous pouvons le remarquer dans cet exemple, seul le comportement du composant G^1 peut être affecté par des fautes. On peut donc construire son diagnostiqueur qui correspond à $G^1{}_d$ sur la Figure III-16. Nous pouvons noter que ce diagnostiqueur comporte deux cycles d'états F1-indéterminé et F2-indéterminé. Comme ces cycles sont dus à des transitions observables du composant, ils correspondent également à des cycles $F1^{M1}$-indéterminé. Donc le langage $L(G^1)$ n'est ni globalement diagnosticable, ni modulairement diagnosticable. Lorsqu'on observe le diagnostiqueur $G^T{}_d$ du système, on constate que lui aussi possède un cycle d'états F1-indéterminé. Par contre ce cycle étant dû à l'événement 'b' qui n'est pas un événement observable de G^1, ce n'est donc pas un cycle $F1^{M1}$-indéterminé. Donc le système T n'est pas globalement diagnosticable mais il est modulairement diagnosticable. Donc le système est modulairement diagnosticable alors que l'un de ses composants n'est pas localement diagnosticable. D'où le contre-exemple.

D'une manière générale, les auteurs ont démontré un certain nombre de propriétés relatives à la diagnosticabilité modulaire.

Proposition III-5 : Relations entre diagnosticabilité modulaire et diagnosticabilité globale

P1 - Etant donné $T=\{1, \ldots, p\}$ un système et $S \subseteq T$ avec $G^s=||_{z \in S}G^z$. Si le langage $L(G^s)$ est globalement diagnosticable par rapport à $\Sigma^s{}_o$ et $\Sigma^s{}_f$ alors $L(G^s)$ est modulairement diagnosticable par rapport à $\Sigma^s{}_o$ et $\Sigma^s{}_f$.

P2 - Etant donné $T=\{1, \ldots, p\}$ un système, $S \subseteq T$ avec $G^s=||_{z \in S}G^z$ et $i \in S$. Si le langage $L(G^i)$ est globalement diagnosticable par rapport à $\Sigma^i{}_o$ et $\Sigma^i{}_f$ alors $L(G^s)$ est modulairement diagnosticable par rapport à $\Sigma^s{}_o$ et $\Sigma^s{}_f$.

De la partie P1 de la Proposition III-5 nous pouvons déduire que pour un composant G^i, non décomposable, la diagnosticabilité modulaire est équivalente à la diagnosticabilité globale du composant. Par analogie avec les travaux de Debouk, nous parlerons de diagnosticabilité locale pour un composant.

De la partie P2 de la Proposition III-5, nous pouvons déduire que pour tout composant G^s, si tous les composants de S sont localement diagnosticables, alors S est modulairement diagnosticable.

L'objectif d'une telle approche est de permettre d'établir la diagnosticabilité modulaire d'un système composite complexe sans avoir à construire son diagnostiqueur. Pour cela, les auteurs ont proposé un algorithme permettant de vérifier la diagnosticabilité modulaire d'un système

à partir de la recherche des cycles d'états F^{Mi}-indéterminé dans les diagnostiqueurs de ses composants (Modular Diagnosticability Algorithm – MDA). Cet algorithme est basé sur les propriétés suivantes :

- Pour qu'un cycle d'états Fj^{Mi}-indéterminé existe dans le diagnostiqueur système, il faut qu'au moins l'un des diagnostiqueurs de ses composants possède un cycle d'états Fj-indéterminé.
- Si un cycle d'états Fj-indéterminé existe dans le diagnostiqueur d'un des composants du système, alors il peut y avoir ou pas de cycle d'états Fj^{Mi}-indéterminé dans le diagnostiqueur système.

Le principe de MDA pour un système composite S composé de composants i, consiste dans un premier temps à vérifier si tous les composants sont localement diagnosticables. Si oui le système est modulairement diagnosticable. Sinon, il faut rechercher dans chaque diagnostiqueur local, ses cycles Fj-indéterminés. Ensuite, vérifier à l'aide des événements communs d'un composant si ses cycles d'états Fj-indéterminés seraient accessibles dans le diagnostiqueur du système. Dès qu'un cycle est accessible alors le système n'est pas modulairement diagnosticable. Sinon le système est modulairement diagnosticable.

En pratique, lorsqu'un système est modulairement diagnosticable, MDA permet de construire une architecture modulaire avec des modules qui sont chacun localement diagnosticable. Le principe de l'approche proposée est incrémental. Il consiste à vérifier dans un premier temps si toutes les fautes d'un composant sont diagnosticables à l'aide des événements observables du composant. Si oui, le composant est diagnosticable de manière modulaire. Sinon, on crée un module sous-système, regroupant le composant avec un autre composant avec lequel il a des événements en communs. L'idée est de voir si l'augmentation du nombre d'événements observables va permettre de rendre diagnosticable le regroupement ainsi créé en supprimant des cycles d'états Fj-indéterminés internes au sous module (cf. la suppression des cycles de $G^1{}_d$ dans G^T dans la Figure III-16). La synchronisation des composants ainsi regroupés est faite sur la base des événements communs aux deux modules regroupés. Si le module est modulairement diagnosticable on s'arrête. Sinon, on poursuit les regroupements. On s'arrête soit quand la diagnosticabilité modulaire est obtenue, soit lorsqu'on n'augmente plus le nombre d'événements observables de l'ensemble regroupé, soit quand le module est équivalent à l'ensemble du système.

Le diagnostic en ligne consiste alors simplement à faire fonctionner en parallèle les diagnostiqueurs locaux de chaque module (Figure III-17).

Le test de diagnosticabilité modulaire est fait module par module. Les travaux de Contant sont basés sur l'utilisation de la méthode du diagnostiqueur (Sampath, et al., 1995). Mais a priori, rien n'empêche d'utiliser la technique du vérificateur ou le test du vide de Büchi présenté dans le § III.I.1 et le §III.I.2.

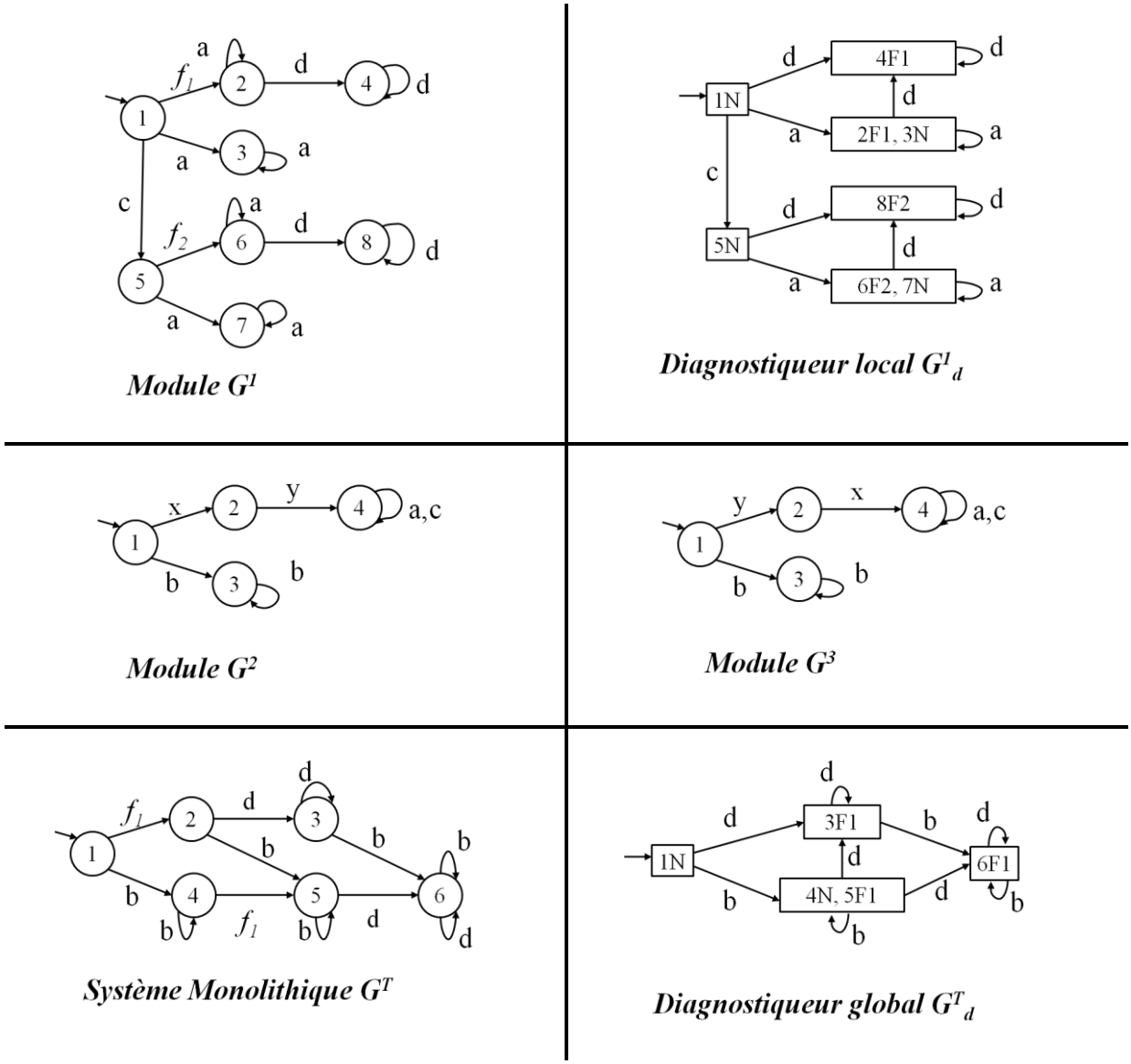


Figure III-16 – Contre-exemple montrant que la diagnosticabilité modulaire n'entraine pas la diagnosticabilité locale (Contant, et al., 2006)

Discussion

Il faut noter que la méthode n'indique pas si les modules construits doivent aboutir à un partitionnement du système. En fait, la méthode semble permettre de réutiliser un même module dans plusieurs regroupements. Dans ce cas, la diagnosticabilité modulaire s'apparenterait à une forme de diagnosticabilité décentralisée au sens de (Debouk, et al., 2000). En effet, dans ce cas, certaines fautes seraient diagnosticables dans plusieurs modules et se poseraient le problème de la synthèse de décisions locales pour obtenir une décision locale. Mais si le système est modulairement diagnosticable, étant donné que toute faute serait diagnosticable dans un module, le système serait en fait F-diagnosticable. Nous pouvons donc en conclure qu'il est possible dans ce cadre d'utiliser une fonction de décision (Wang, et al., 2005) (Lafortune, et al., 2005) pour obtenir un diagnostic global de l'état du système.

Si le regroupement des composants en modules aboutit à une partition du système, le système est non seulement F-diagnosticable mais en plus il est indépendamment diagnosticable (Sengupta, 1998). Nous pouvons donc dans ce cas envisager une mise en œuvre physiquement distribuée du système de surveillance du système.

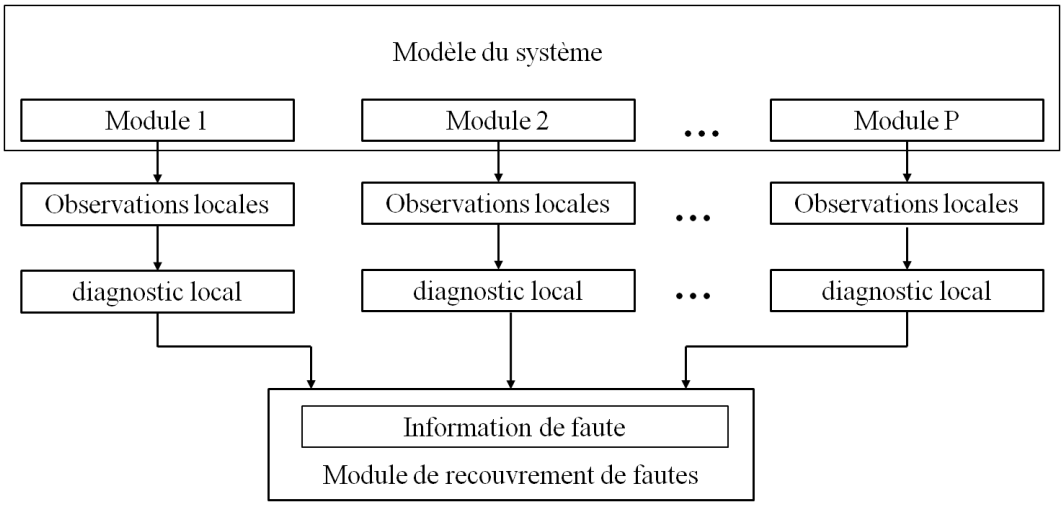


Figure III-17 – Architecture du diagnostic modulaire (Contant, et al., 2006)

Conclusion du chapitre III

Dans ce chapitre, nous avons cherché à dégager une méthode permettant l'étude de la diagnosticabilité des Systèmes Embarqués Logiques (SEL). Rappelons que ces systèmes, du fait de leur grand nombre d'entrées/sorties, ils rendent difficiles la caractérisation de leurs propriétés dynamiques par des approches formelles en raison de risques d'explosion combinatoire.

Pour cela, dans ce chapitre nous nous sommes d'abord aux techniques de test de la diagnosticabilité. Nous avons identifiée dans la littérature 3 techniques : celle basée sur la construction du diagnostiqueur du système, celle basée sur la construction d'un vérificateur et celle basée sur le test du vide d'un automate de Büchi. La première technique pose le problème de la construction a priori d'un diagnostiqueur monolithique avec le risque d'explosion combinatoire qu'elle engendre. Elle est celle la plus utilisée dans la littérature car la plus simple à comprendre. Mais elle n'est clairement pas celle à mettre en œuvre dans le cadre des SEL. La technique du vérificateur repose sur le produit d'automates. Elle semble a priori poser le même problème de complexité que le cas de l'approche diagnostiqueur car elle est basée sur le produit des deux automates. On trouve dans la littérature des algorithmes polynomiaux (Jiang, et al., 2001)(Yoo, et al., 2002) qui permettent d'établir la diagnosticabilité avec une complexité de l'ordre du carré du nombre d'états du modèle du système. Cette technique semble donc difficilement applicable aux SEL. La technique du vide du test de Büchi repose également à un produit de copies du modèle du système mais cette fois ci la construction n'est pas symétrique. On supprime dans la $2^{ème}$ copie toutes les transitions et les états atteignables après la faute considérée. La méthode a une complexité

linéaire par rapport au nombre d'états et de transitions et il existe des algorithmes permettant de faire cette vérification à la volée (Holzmann, 2005). On peut notamment faire simplement des vérifications en utilisant des outils de model-checking qui réduisent le test à une recherche d'accessibilité d'états. C'est cette technique que nous allons chercher à exploiter pour établir la diagnosticabilité des SEL.

Dans la deuxième partie de ce chapitre nous avons fait une étude sur la diagnosticabilité répartie. Cette étude, nous a permis dans un premier temps de nous rendre compte qu'au-delà des hypothèses spécifiques et des terminologies propres à chaque auteur, il faut distinguer deux contextes de décomposition d'un système : la hiérarchisation et la distribution. Dans le cadre de la hiérarchisation système-composants, on peut distinguer trois formes de diagnosticabilité applicables soit au niveau système, soit au niveau local : la diagnosticabilité globale ou diagnosticabilité monolithique, la diagnosticabilité décentralisée, la diagnosticabilité modulaire (cf. synthèse Table III-5).

La diagnosticabilité monolithique s'applique aussi bien au niveau système qu'au niveau local. C'est elle qui est utilisée dans le contexte centralisé. Elle s'applique également dans le cadre de la diagnosticabilité modulaire pour définir si un système est localement diagnosticable ou pas.

Approche	Auteur	Année	Système	Observabilité	Fautes	Centre de décision globale	Nature des échanges
Diagnosticabilité centralisée	Sampath	1995	global	globale	globales	Non	
Diagnosticabilité décentralisée	Debouk	1998	global	distribuée	globales	coordonateur	verticaux
Diagnosticabilité indépendante	Sengupta	1999	local	locale	locales	non	horizontaux
Diagnosticabilité distribuée	Sengupta	1999	local	distribuée	locales	non	horizontaux
Diagnosticabilité par composant	Debouk	2003					
Codiagnosticabilité	Wang	2005	global	distribuée	globales	fusion	verticaux
Diagnosticabilité modulaire	Contant	2006	local	locale	locales	non	

Table III-5 – Synthèse des différentes approches de diagnosticabilité

Dans les architectures décentralisées, il n'y a ni partition des événements observables par site ni partition des événements de fautes. Les sites peuvent diagnostiquer les mêmes fautes en partageant des événements communs. Cela explique que deux sites peuvent arriver à des décisions différentes par rapport à une faute car ils n'ont pas le même degré d'observabilité par rapport à la faute. En général, les architectures décentralisées ne garantissent pas qu'un site puisse diagnostiquer toutes les fautes. Elles permettent d'obtenir sous certaines conditions une diagnosticabilité comparable à la diagnosticabilité globale au niveau du système. Dans le cadre de la codiagnosticabilité, elles garantissent que toute faute du système peut être diagnostiquée comme étant présente ou absente par au moins l'un des sites de l'architecture de surveillance.

La diagnosticabilité modulaire est un concept qui s'applique aussi bien au niveau local qu'au niveau système. Au niveau système, elle est a priori moins exigeante que la diagnosticabilité monolithique. Dans un contexte réparti, elle permet même de ne pas chercher à vérifier la diagnosticabilité monolithique puisqu'en pratique, on peut s'en passer. Au niveau local, elle est équivalente à la diagnosticabilité monolithique. Les concepts de diagnosticabilité indépendante (Sengupta, 1998) et de diagnosticabilité par composant (Debouk, 2003), ne sont en fait que de la diagnosticabilité modulaire appliquée localement.

Dans le cadre de la diagnosticabilité distribuée (Sengupta, 1998), les sites communiquent directement entre eux pour éviter l'utilisation d'un centre de décision. Le diagnostic est donc local à chaque site à partir d'une observabilité globale.

Cette étude nous permet donc de choisir la diagnosticabilité modulaire au sens de O. Contant (Contant, et al., 2006) comme méthode pour étudier la diagnosticabilité des SEL. En effet l'intérêt de la diagnosticabilité modulaire est qu'elle nous permettra de réduire les risques d'explosions combinatoires en décomposant le système en composants. Le grand avantage de cette méthode par rapport aux autres est qu'on n'a pas besoin de centre de fusion pour décider de l'état au niveau système en fonction du diagnostic local de chaque module. Notre idée est de pouvoir vérifier séparément la diagnosticabilité de chaque module et ensuite de mettre en œuvre des diagnostiqueurs indépendants qui diagnostiquent les fautes internes du module. Cela nécessitera d'avoir pour chaque module l'observabilité suffisante pour garantir la diagnosticabilité modulaire. Indirectement, ça pose le problème des signaux que l'on devra être capable d'observer à l'intérieur d'un SEL. Cette étude a également montré qu'il faudra définir de manière judicieuse les classes de fautes à surveiller dans chaque module.

Dans la suite de ce travail, nous adopterons la terminologie suivante :

- Diagnosticabilité locale : C'est la capacité d'un module à diagnostiquer ses fautes internes à l'aide de ses seuls observations locales.
- Diagnosticabilité modulaire : C'est la capacité d'un système à diagnostiquer chacune de ses fautes à l'aide des seules observations du module d'appartenance de la faute.
- Diagnosticabilité système ou diagnosticabilité globale ou diagnosticabilité monolithique : c'est la capacité d'un système à diagnostiquer n'importe laquelle de ses fautes à l'aide de l'ensemble de ses observation.

IV Diagnostic et model-checking applicables aux SEL

Introduction du chapitre IV

Dans le chapitre précédent nous avons identifié une méthode pour le diagnostic des Systèmes Embarqués Logiques (SEL). Nous avons choisi une méthode basée sur la diagnosticabilité modulaire. L'idée de cette méthode est de réduire les risques d'explosion combinatoire en évitant de construire des modèles globaux du système. Le système sera décomposé en composants dont le comportement est moins complexe et donc, dont les modèles sont de taille réduite. D'autre part, pour étudier la diagnosticabilité, nous avons montré qu'il existe différentes techniques dans la littérature et qu'a priori la technique du vide du test d'un automate de Büchi associée à la vérification par model-checking est le moyen le plus simple et le moins combinatoire pour cela.

Dans ce chapitre, nous voulons adapter les résultats précédents à l'étude de la diagnosticabilité des SEL. Ils constituent une classe de systèmes temporisés (ST). C'est la raison pour laquelle dans la première partie de ce chapitre, nous allons d'abord nous intéresser à la diagnosticabilité et à la synthèse de diagnostiqueur pour les systèmes temporisés. Nous verrons notamment que les résultats des SED ne sont pas applicables sans adaptations au cas des ST. Nous verrons notamment que dans le cas général, un diagnostiqueur de ST n'est pas forcément un automate temporisé, mais tout simplement une machine de Türing (Tripakis, 2002). C'est la raison pour laquelle, dans un second temps, nous nous intéressons au cas particulier des automates temporisés déterministes. Nous verrons que dans ce cadre, certains résultats des automates à états fini s'appliquent (Bouyer, et al., 2005a).

Dans la deuxième partie de ce chapitre, nous présenterons la technique du model-checking et la logique temporelle TCTL. L'objectif de cette présentation est de montrer comment le model-checking, qui est une technique de vérification formelle, peut être utilisé pour la vérification de la diagnosticabilité et la synthèse de diagnostiqueur. Nous proposerons une introduction aux deux outils de model-checking, KRONOS et UPPAAL que nous utiliserons dans le chapitre suivant pour mettre en œuvre notre méthode de diagnostic modulaire.

IV.I Diagnosticabilité des systèmes temporisés

Dans le chapitre III, nous avons vu que la diagnosticabilité des SED est la capacité à garantir le diagnostic de toute faute dans un délai borné. Dans un modèle automate à états finis, il n'y a pas une modélisation explicite du temps. La modélisation du temps est implicite au travers

des événements reçus. Donc, pour définir cette notion de délai borné, on compte un nombre fini d'événements après l'occurrence d'une faute, la borne dépendant du type de la faute. Selon, les auteurs, la borne est définie soit par rapport à l'ensemble des événements caractéristiques de comportement du système (Sampath, et al., 1995)(Sampath, et al., 1996) ou par rapport exclusivement aux événements observables (Contant, et al., 2006). Les deux définitions sont équivalentes dans la mesure où dans la définition initiale de (Sampath, et al., 1995), on fait l'hypothèse que le modèle du système ne peut pas comporter de cycles ne contenant que des événements inobservables.

IV.I.1 Expressivité des systèmes temporisés

Dans le cas des systèmes temporisés, le diagnostic ne peut pas être basé juste sur l'interprétation de l'occurrence ou pas d'événements, mais le délai entre les événements est également très important. L'idée est donc de pouvoir définir de manière explicite cette borne temporelle. Considérons à nouveau l'exemple de la Figure IV-1 avec le cas α=3. Ce système est caractérisé par deux événements observables (Σ_o={a,b}) et deux événements inobservables (Σ_{uo}={u,f}). L'événement 'f' caractérise une faute. Le système produit systématiquement la séquence d'événements observables 'a * b'. On peut se demander si l'observation de cette séquence permet de dire si le système est défaillant ou pas?

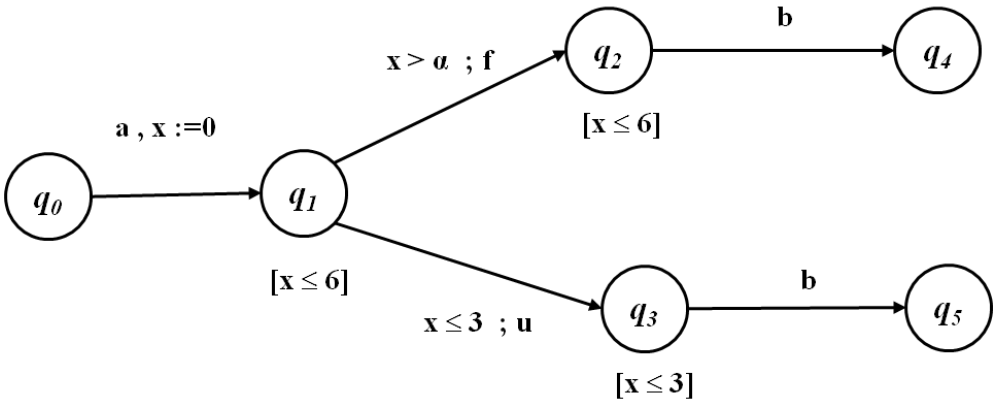


Figure IV-1 – Automate temporisé modélisant un système fautif (Tripakis, 2002)

En cas d'occurrence d'une faute, d'après la valeur de la garde de la transition de $q1$ vers $q2$ et l'invariant de $q2$, on en déduit que dans ce cas l'occurrence de 'b' survient forcément plus de 3 ut après celle de 'a'. Par contre, en cas de comportement normal (séquence 'a*u*b'), l'événement 'b' doit survenir au plus tard 3 ut après l'événement 'a '. Cette analyse et notamment la relation temporelle entre 'a' et 'b' permet donc de diagnostiquer l'état du système au bout de 3 ut après l'occurrence de 'a'. En effet, au bout de 3 ut après 'a', en cas d'absence de 'b', on peut diagnostiquer l'occurrence d'une faute sans même attendre 'b'. Donc, on peut dire que le délai borné pour diagnostiquer l'occurrence d'une faute de ce système est de 3 ut.

Cet exemple montre qu'il est nécessaire d'adapter la définition de la diagnosticabilité proposée par Sampath au cas des systèmes temporisés. Mais avant de proposer cette nouvelle définition, nous allons compléter les définitions et les propriétés relatives aux automates temporisés (cf. chapitre II).

IV.I.2 Vivacité d'un automate temporisé

Dans les travaux de Sampath, une condition préalable à l'étude de la diagnosticabilité des systèmes est la vivacité du système. Rappelons que cette notion de vivacité est définie pour les SED de manière à signifier que le comportement du système est tel qu'il produit suffisamment d'événements pour être capable de diagnostiquer une faute. Dans le cas des systèmes temporisés, le comportement du système résulte de deux mécanismes : les événements (ou actions) et la progression du temps. Donc deux phénomènes peuvent entraver la vivacité d'un système temporisé : un blocage ou une non progression du temps.

Définition IV-1 : Location bloquante par rapport à une valuation (Chamseddine, 2009)

Etant donné un automate temporisé G_t et (q, v) l'un de ses états. On dit que la location q est bloquante par rapport à la valuation v si pour toute date ultérieure, aucune des transitions de sorties de q $(out(q))$ ne peut être franchies.

(q, v) *est bloquant si* $\forall \tau \in \mathbb{R}^+ \nexists \delta_t (< q, v + \tau >, g, a, r) = < q', v' >$
avec $v' = v[r := 0] + \tau$

Définition IV-2 : Etat bloquant – Automate bloquant

Un état (q, v) d'un automate temporisé G_t est dit bloquant s'il existe une valuation v dans $\mathbb{R}^+$ tel que la location q est bloquante. Un automate est dit bloquant s'il possède des états bloquants.

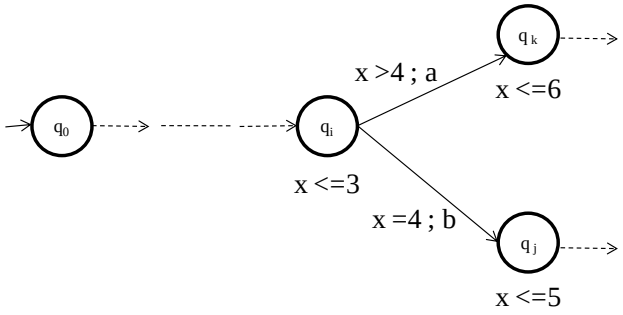


Figure IV-2 – Blocage par incompatibilité entre invariant de l'état source et les gardes des transitions de sorties

Etant donné q une location de G_t appartenant à l'ensemble des états accessibles à partir de q_0, nous pouvons distinguer deux conditions de blocage. Le premier cas de blocage est lorsque qu'il y a une incompatibilité de valeur entre l'invariant associé à l'état source et la garde des transitions de sortie de cet état (Figure IV-2).

Le deuxième cas de blocage est plus subtil. Il survient lorsque la valuation des horloges de la transition entrante ne vérifie pas l'invariant de l'état cible. La transition n'est donc pas autorisée dans ce cas.

Rappelons qu'un mot temporisé sur un alphabet Σ est une séquence finie ou infinie $w=\gamma_1*\gamma_2\ldots*\gamma_n$ où les γ_n sont soit des événements de Σ, soit des délais définis dans $(\mathbb{R}^*)^+$. Entre deux événements, il y a forcément un délai.

Etant donné w, un mot temporisé, nous notons $duree(w)$ la somme de tous les délais de w.

Définition IV-3 : Mot temporisé non-zénon

Etant donné w un mot temporisé infini sur Σ, w est non-zénon si $duree(w) = \infty$.

En d'autres termes un mot temporisé est zénon si une durée finie est associée à une séquence d'événements infinie. De tels mots sont associés à des exécutions d'automates temporisés pour lesquelles le temps ne divergent pas. On parle alors de blocage temporel. Par exemple sur l'exemple de la Figure IV-3, on peut remarquer que l'horloge de la transition de q_2 vers q_3 a pour valeur 'x≤3', donc cette transition ne peux pas être franchie parce que le chemin permettant d'atteindre la localité q_2 entraine que l'horloge 'x' est forcément strictement supérieure à 3. Donc on va avoir une infinité de transitions permettant d'évoluer dans la localité q_2 sans que l'horloge 'x' ne dépasse la valeur 6 à cause de l'invariant de la location q_2. C'est une situation de blocage du temps.

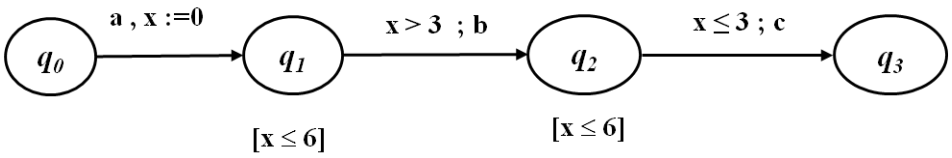


Figure IV-3 – Exemple d'automate zénon

Les comportements zénon ne sont pas souhaitables car ils ne correspondent pas à la réalité. En effet, dans les systèmes temps-réels, le temps diverge. Donc on devra toujours vérifier qu'un modèle de système temps-réel à un comportement non-zénon.

Nous notons $\Delta G_t(q_0)$ l'ensemble des états accessibles de l'automate temporisé G_t à partir de la location initiale q_0.

Proposition IV-1 : Automate temporisé sans blocage temporel (Chamseddine, 2009)

Un automate temporisé G_t *est dit sans blocage temporel si et seulement si* $\forall q \in \Delta G_t(q_0)$ $\exists\, \rho = < q, v > \xrightarrow{g_1 ; a_1 ; r_1}_{d_1} < q_1, v_1 > \cdots < q_{n-1}, v_{n-1} > \xrightarrow{g_n ; a_n ; r_n}_{d_n} < q_n, v_n > \cdots$ *partant de* $< q, v >$ *avec* $< q, v >$ *accessible et une position i de ρ tel que* $\sum_{k=0}^{i-1} d_k \geq 1$.

Autrement dit, on peut toujours, à partir d'un état accessible, extraire d'une exécution infinie, une exécution partielle dont la durée soit supérieure ou égale à 1. Comme on peut répéter ce

raisonnement à partir de tout état atteint de cette exécution infinie, par itération il vient que la durée de l'exécution initiale ρ est infinie.

Définition IV-4 : Automate temporisé fortement non-zénon

Un automate temporisé est fortement non-zénon, si $\exists K \in \mathbb{N}$ *tel que* $\forall \rho = < q_0, v_0 > \xrightarrow{g_1 ; a_1 ; r_1}_{d_1} < q_1, v_1 > \cdots < q_{n-1}, v_{n-1} > \xrightarrow{g_n ; a_n ; r_n}_{d_n} < q_n, v_n >$ *dans* G_t, $n \geq K$ *implique* $\sum_{i=1}^{n} d_i \geq 1$.

Autrement dit, dans chaque cycle d'un automate temporisé fortement non-zénon, il s'écoule au moins une unité de temps. Donc si un automate temporisé contient des cycles accessibles à partir de l'état initial, si la durée des ces cycles est supérieure à 1 ut et que cet automate ne contient pas d'états bloquants, alors cet automate est fortement non-zénon.

La conséquence de la Définition IV-4 est qu'un automate temporisé fortement non-zénon est non-zénon.

Définition IV-5 : Automate temporisé bien formé

Un automate temporisé G_t *est dit bien formé s'il est non bloquant et étant donné* $\Delta G_t(q_0)$ *l'ensemble de ses états accessibles, si toute exécution à partir de* $< q, v > \in \Delta G_t(q_0)$ *correspond à un mot temporisé non-zenon.*

Dans la suite de ce travail, nous supposons que nous travaillons toujours à l'aide d'automates temporisés bien formés. Donc en quelque sorte, nous avons des modèles qui correspondent à des systèmes temps-réel, dont le temps progresse sans être bloqué. Il existe dans la littérature des algorithmes qui permettent de contrôler qu'un automate temporisé est bien formé (Tripakis, 1999) (Chamseddine, 2009).

IV.I.3 Test de diagnosticabilité

Comme nous l'avons vu sur l'exemple du §IV.I.1, la diagnosticabilité des systèmes temporisés réside dans la capacité à diagnostiquer toute faute après observation, durant un temps suffisant, d'événements observables liés par des contraintes temporelles (mots temporisés d'événements observables). Le système n'est pas diagnosticable si certaines fautes ne sont pas diagnosticables même avec une observation suffisamment longue. On peut formaliser cela par la définition suivante :

Définition IV-6 : Diagnosticabilité des systèmes temporisés (Tripakis, 2002)

Etant donné G_t*, un automate temporisé et* Σ_f *une partition de l'ensemble des fautes. On dit que* G_t *est* Δ_f*-diagnosticable par rapport à* Σ_o *et* Σ_f*, si pour tout entier* $\Delta_f \in \mathbb{N}$*, pour toutes exécutions finies* ρ_1 *et* ρ_2 *de* G_t*, si* ρ_1 *est une exécution fautive par rapport à* Δ_f*, alors soit* ρ_2 *est fautive ou* $P_o(\rho_1) \neq P_o(\rho_2)$*.*

On dit que G_t est diagnosticable si pour tout type de faute f, il existe $\Delta_f \in \mathbb{N}$ tel que G_t est Δ_f-diagnosticable.

Remarque : Dans la Définition IV-6, Δ_f est pris en tant qu'entier pour faciliter la détermination d'une valeur minimale pour chaque type de faute. En pratique, Δ_f peut être un réel positif.

Pour tester la diagnosticabilité, l'idée consiste à vérifier s'il est possible de trouver dans G_t, deux exécutions, l'une fautive et l'autre non fautive ayant la même projection par rapport aux événements observables. La méthode proposée dans (Tripakis, 2002) consiste à créer un automate temporisé spécial résultant du produit synchrone de deux copies de l'automate G_t. Nous noterons cet automate $(G_t^1 ||_s G_t^2)^{-f2}$. Cette notation signifie que l'événement $f2$ est supprimé dans l'automate résultant. Pour chaque faute f au sein de G_t, la technique de construction de cet automate est la suivante :

1 – Construction de copies de l'automate temporisé G_t

En fait, cette construction est basée sur la construction de deux copies de G_t que nous noterons $G_t^i, i \in \{1,2\}$. Chaque copie G_t^i est obtenue en renommant à l'aide de l'indice 'i', les états, les horloges et les événements inobservables de G_t. Comme l'objectif est de vérifier si on peut trouver une exécution dans chacun des automates ayant la même projection sur Σ_o, on ne renomme pas les événements observables (cf. Figure IV-4).

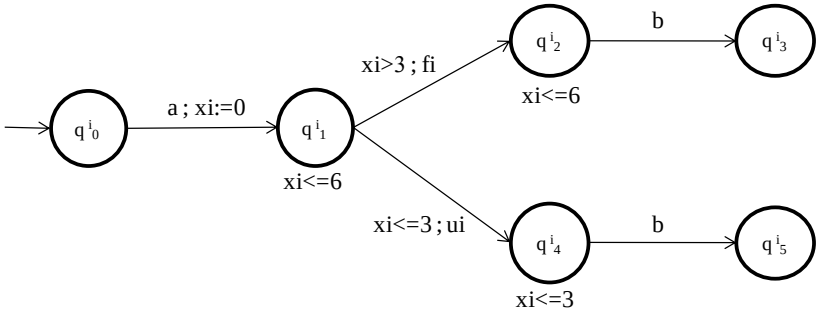


Figure IV-4 – Copie G_t^i du système de la Figure IV-1 pour α = 3

2 – Construction du produit synchrone $G_t^1 ||_s G_t^2$

Les deux copies sont synchronisées sur leurs événements observables. Etant donné que les événements inobservables ont été renommés, les transitions qui leurs sont associées s'entrelacent dans des exécutions parallèles.

Dans la Figure IV-5, considérons par exemple une exécution menant de l'état $< q_0^1, q_0^2 >$ à l'état$< q_3^1, q_3^2 >$. Cette exécution correspond à :

$\rho 1 = (< q_0^1, q_0^2 >; 0) \xrightarrow{a} (< q_1^1, q_1^2 >, 0) \xrightarrow{3,1} (< q_1^1, q_1^2 >; 3,1) \xrightarrow{f_1} (< q_2^1, q_1^2 >; 3,1) \xrightarrow{0,5} (< q_2^1, q_1^2 >; 3,6) \xrightarrow{f_2} (< q_2^1, q_2^2 >; 3,6) \xrightarrow{1,0} (< q_2^1, q_2^2 >; 4,6) \xrightarrow{b} (< q_3^1, q_3^2 >; 4,6) \xrightarrow{0,3} (< q_3^1, q_3^2 >; 4,9)$.

Dans cette exécution on peut remarquer une transition due à 'f1' et une autre due à 'f2'. De même, on a une autre exécution comme ρ_2, dans laquelle on a l'inverse, c'est-à-dire une transition 'f2' suivie d'une transition 'f1'. Cela illustre l'entrelacement des événements inobservables issus des deux copies.

$\rho 2 = (< q_0^1, q_0^2 >; 0) \xrightarrow{a} (< q_1^1, q_1^2 >, 0) \xrightarrow{3,1} (< q_1^1, q_1^2 >; 3,1) \xrightarrow{f_2} (< q_1^1, q_2^2 >; 3,1) \xrightarrow{0,5} (< q_1^1, q_2^2 >; 3,6) \xrightarrow{f_1} (< q_2^1, q_2^2 >; 3,6) \xrightarrow{1,0} (< q_2^1, q_2^2 >; 4,6) \xrightarrow{b} (< q_3^1, q_3^2 >; 4,6) \xrightarrow{0,3} (< q_3^1, q_3^2 >; 4,9)$.

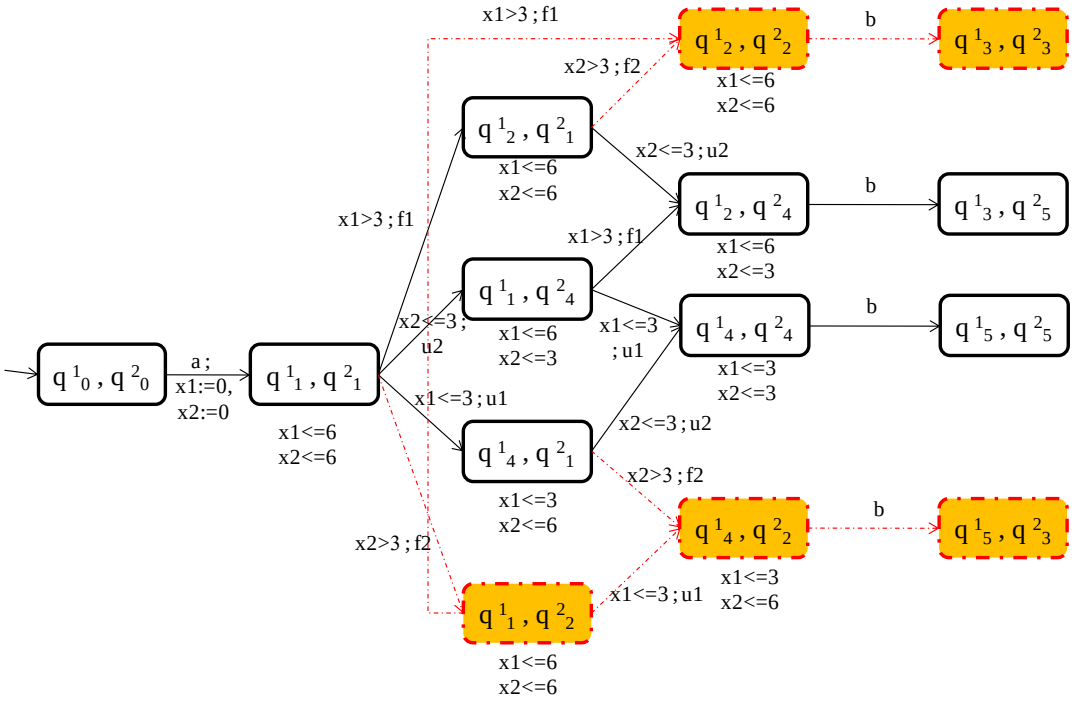


Figure IV-5 – Produit synchrone $G_t^1||_s G_t^2$ correspondant à l'automate temporisé de la Figure IV-4

3 – Construction de $(G_t^1||_s G_t^2)^{-f2}$

Cet automate est obtenu en supprimant dans $G_t^1||_s G_t^2$ toutes les transitions qui dépendent de la copie 'f2' d'une faute 'f', d'où sa notation $(G_t^1||_s G_t^2)^{-\boldsymbol{f2}}$. Si cette transition était la seule permettant d'arriver dans l'état destinataire, alors on supprime cet état ainsi que tous les suivants qui n'étaient accessibles que par une exécution utilisant la transition liée à 'f2'.

Considérons à nouveau l'automate temporisé de la Figure IV-5. Les transitions hachurées et les états hachurés et colorés sont des éléments du graphe qui dépendent directement ou

indirectement de l'action 'f2'. Conformément à la règle de suppression de tout ce qui dépend de f2, nous pouvons supprimer tous ces éléments. Le résultat obtenu correspond à la Figure IV-6.

L'état coloré dans la Figure IV-6 correspond à un état bloquant. En effet, on entre dans l'état $< q_2^1, q_1^2 >$ par une transition dont la garde est $x_1 > 3$. Comme les deux horloges x_1 et x_2 ont été initialisées ensembles, dans cet état l'horloge x_2 a également une valeur strictement supérieure à 3. Or, la garde de la seule transition de sortie de cet état est $x_2 \leq 3$. Cela signifie que cette transition ne peut pas être franchie car sa garde sera toujours fausse. On est bloqué dans cet état.

La transition $\left(< q_1^1, q_4^2 > \xrightarrow{x1>3;f1} < q_2^1, q_4^2 >\right)$ illustre le blocage de transition. Cette transition ne peut jamais être franchie car l'invariant de son état source est $x_1 \leq 6 \; et \; x_2 \leq 3$. Comme les deux horloges ont été initialisées en même temps, et que l'on doit avoir quitté cet état avant qu'elles atteignent une valeur supérieure à 3, alors la garde de la transition est toujours fausse.

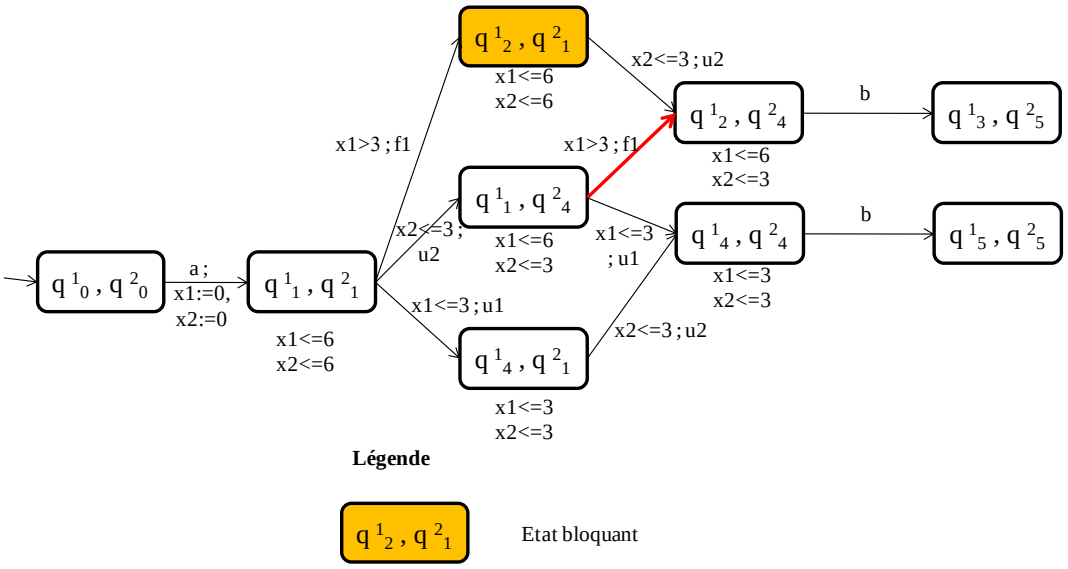


Figure IV-6 – Automate $(G_t^1 ||_s G_t^2)^{-f2}$ correspondant à l'automate temporisé de la Figure IV-4.

Les automates temporisés de départ étant bien formés, la présence d'états bloquants entrainant des exécutions zénon prouve que l'entrelacement des événements non observables des deux copies d'automate conduit à des erreurs de construction d'automates. Donc, on peut en déduire qu'on ne peut pas trouver deux exécutions suffisamment longues de G_t, l'une fautive et l'autre non fautive, dont la projection de la trace observable soit la même. Ainsi, le système de la Figure IV-4 est diagnosticable d'après ce test.

Cet exemple illustre un résultat important du test de la diagnosticabilité des systèmes temporisés.

Proposition IV-2 : Test de diagnosticabilité d'un système temporisé (Tripakis, 2002)

Un automate temporisé G_t est diagnosticable si et seulement si toute exécution fautive de $(G_t^1 ||_s G_t^2)^{-f2}$ est zénon.

IV.I.4 Mise en œuvre du test de diagnosticabilité

Il existe de nombreux outils informatiques qui permettent de tester les propriétés des modèles automates temporisés. La majeure partie sont des outils de model-checking comme KRONOS (Yovine, 1997)(Bozga, et al., 1998) ou UPPAAL (Larsen, et al., 1997). Tous ces outils ont en commun le fait de baser leur fonctionnement sur des mécanismes de recherche d'accessibilité. Or les notions d'état bloquant ou d'exécution zénon sont plutôt relatives à la vivacité. Il est donc nécessaire de trouver une méthode de test basée sur l'accessibilité.

La méthode proposée dans (Cassez, et al., 2008) ressemble à celle proposée dans le chapitre III pour la vérification de la diagnosticabilité des SED. Elle consiste à adapter le test de la section précédente. Pour cela, nous allons construire deux copies de l'automate temporisé $G_t = (Q, q_0, X, \Sigma, E, Inv)$. Par souci de simplicité, nous supposerons que le comportement de G_t ne peut être affecté que par une seule faute que nous notons f.

Dans la copie G_t^1, nous allons mémoriser au niveau de l'état l'occurrence ou pas d'une faute. A chaque état nous associons le label 0 si cet état n'est pas dans une exécution fautive, sinon nous lui associons le label 1. La mémorisation de l'occurrence de la faute au niveau de l'état nous permet de remplacer l'événement de faute par l'événement vide ε. Ainsi, nous définissons $G_t^1 = (Q^1 \otimes \{0,1\}, (q_0^1, 0), X_1, \Sigma^1, E_1, Inv_1)$ de la manière suivante :

- Les états sont définis par $\forall\, q_i^1 \epsilon\, Q^1\ \exists\ q_i \epsilon Q / q_i^1 \cong q_i$;
- L'état initial est défini par $(q_0^1, 0)$ avec la valeur 0 indiquant que initialement le système n'est pas défaillant ;
- X_1 est obtenu en renommant toutes les horloges de X ($x\ renommé\ en\ x_1 / \ x_1 \cong x$);
- Σ_{uo}^1 est obtenu en renommant tous les événements de Σ_{uo} exceptés l'événement f. $\Sigma^1 = \Sigma_o \cup \Sigma_{uo}^1$;
- Les transitions sont définies par :
  - $\big((q_1^1, n), g_1, \sigma, r_1, (q_2^1, n)\big) \epsilon\ E_1\ si\ (q_1, g, \sigma, r, q_2)\ \epsilon\ E\ avec\ g_1 \cong g\ et\ r_1 \cong r\ si\ \sigma \neq f$
  - $\big((q_1^1, n), g_1, \varepsilon, r_1, (q_2^1, 1)\big) \epsilon\ E_1\ si\ (q_1, g, f, r, q_2)\ \epsilon\ E\ avec\ g_1 \cong g\ et\ r_1 \cong r$
- Les invariants des états sont définis par $Inv_1\left((q_i^1, n)\right) = Inv(q_i)$

La deuxième équation de spécification des transitions de G_t^1 permet d'effacer les événements de faute en les mémorisant. Pour cela, on met la valeur 1 dans les états atteints après défaillance. La première équation permet de propager le label normal (valeur 0) ou défaillant (valeur 1) à l'état de son successeur.

La copie $G_t^2 = (Q^2, q_0^2, X_2, \Sigma^2, E_2, Inv_2)$ est construite de la manière suivante :

- Les états sont définis par $\forall\, q_i^2 \epsilon\, Q^2\ \exists\ q_i \epsilon Q / q_i^2 \cong q_i$;
- X_2 est obtenu en renommant toutes les horloges de X ($x\ renommé\ en\ x_2 / \ x_2 \cong x$);
- Σ_{uo}^2 est obtenu en renommant tous les événements de Σ_{uo} exceptés l'événement f qui est supprimé. $\Sigma^2 = \Sigma_o \cup \Sigma_{uo}^2$;
- Les transitions sont définies par :
  - $(q_1^2, g_2, \sigma, r_2, q_2^2) \epsilon\ E_2\ si\ (q_1, g, \sigma, r, q_2) \epsilon\ E\ avec\ g_2 \cong g\ et\ r_2 \cong r$ si σ∈Σ-{f}
- Les invariants des états sont définis par $Inv_2(q_i^2) = Inv(q_i)$

Nous notons que dans la copie G_t^2 nous supprimons l'événement de faute 'f' et donc toutes les transitions et les états qui en dépendent.

Nous obtenons alors $(G_t^1 ||_s G_t^2)^{-f2}$ en réalisant le produit synchrone des deux automates temporisés sur leurs événements communs de Σ_o.

L'objectif est alors de vérifier s'il existe une exécution fautive et une exécution non fautive dont les projections des traces sont identiques en termes de mot temporisé. Nous devons pour cela vérifier que G_t^1 et G_t^2 ne peuvent pas créer artificiellement des exécutions zénon.

En effet, par construction, il peut exister dans G_t^2 des exécutions zénon. Il faut donc vérifier que G_t^2 vérifie une condition de non-zenoness avant de construire le produit synchrone. Sinon il faut modifier G_t^2 pour garantir cette propriété.

Nous avons vu dans l'exemple de la Figure IV-6 que le produit synchrone des deux copies de G_t peut entrainer la présence dans le modèle d'états bloquant l'évolution du temps. L'objectif est de vérifier si les exécutions fautives ρ peuvent correspondre à des exécutions à temps divergents telles qu'on peut trouver une exécution fautive ρ_1 de G_t^1 et une exécution non fautive ρ_2 de G_t^2 avec $P_o(tr(\rho)) = P_o(tr(\rho 1)) = P_o(tr(\rho 2))$. Pour cela, nous définissons l'automate temporisé $Div(y) = (\{D0, D1\}, 0, \{y\}, E, Inv,)$ (cf. Figure IV-7).

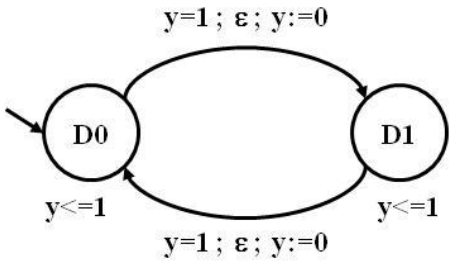


Figure IV-7 – Automate Div(y)

L'horloge y de $Div(y)$ n'est pas une horloge de G_t. A l'aide de $Div(y)$ on définit l'automate temporisé $D = (G_t^1 ||_s G_t^2)^{-f2} \otimes Div(y)$ avec les ensembles $F_D = \phi$ et

$\left(\left((q_i^1, 1), v_1\right), (q_i^2, v_2), (D1, v)\right) \in R_D$ *si* $(((q_i^1, 1), v_1), (q_i^2, v_2))$ est un état fautif de $(G_t^1 ||_s G_t^2)^{-f2}$ et $(D1, v)$ est un état de $Div(y)$.

Proposition IV-3 : Diagnosticabilité et test du vide d'un automate de Büchi (Cassez, et al., 2008)

$L^w(D) \neq \emptyset \Leftrightarrow G_t$ *n'est pas diagnosticable.*

Donc pour tester si G_t est diagnosticable, il suffit d'appliquer un algorithme de test du vide de Büchi sur l'automate temporisé D.

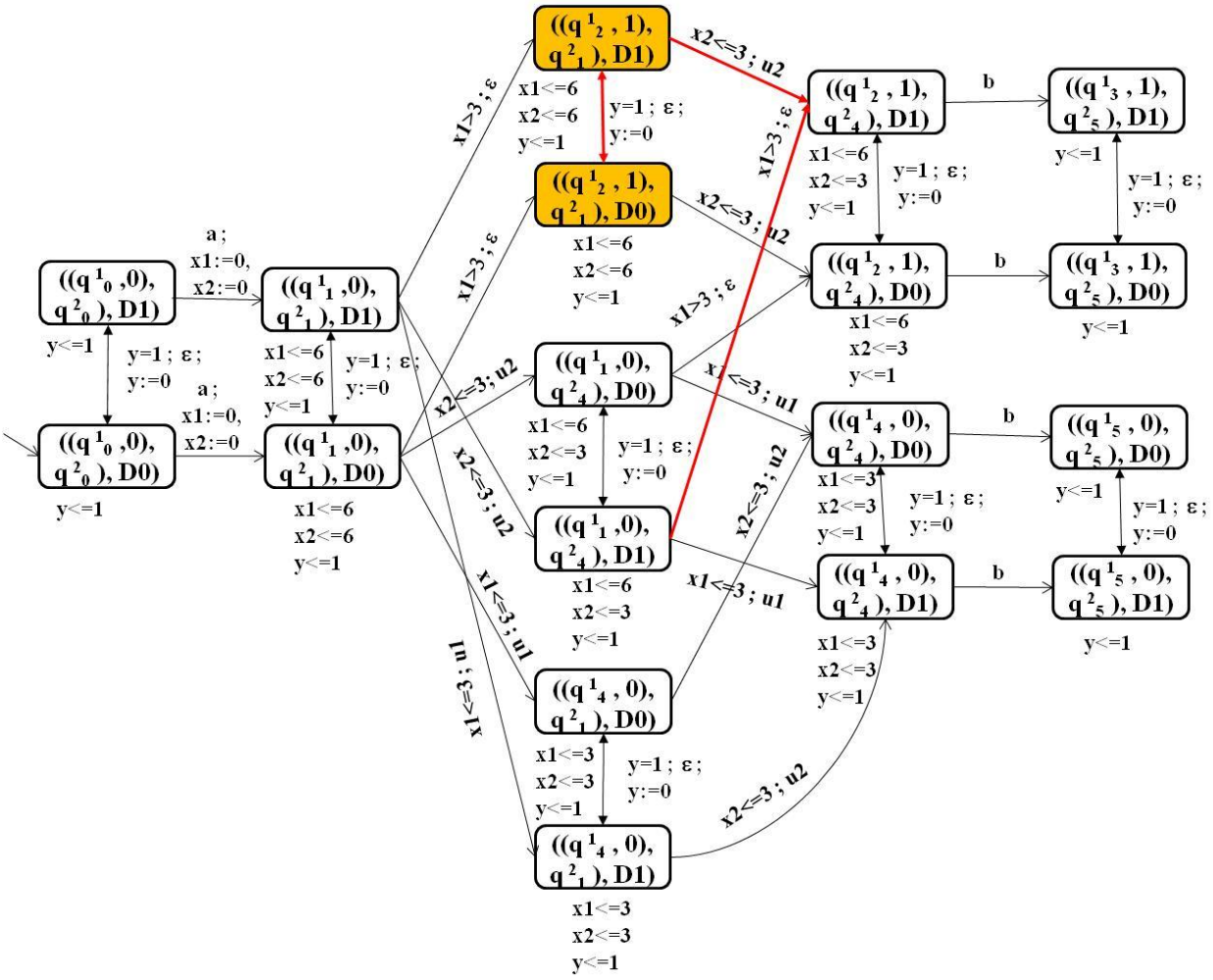


Figure IV-8 – Automate *D* obtenu à partir de l'exemple de Figure IV-1

Considérons l'automate D de la Figure IV-8. Les transitions vers l'état $\left((q_2^1, 1), q_4^2\right), D1)$ et vers l'état $\left((q_2^1, 1), q_4^2\right), D0)$ sont bloquantes en raison d'incompatibilité entre leur garde avec soit l'invariant de l'état prédécesseur, soit l'invariant de l'état suivant. D'autre part, lorsqu'une éxécution arrive dans les états $\left((q_2^1, 1), q_1^2\right), D0)$ et $\left(\left((q_2^1, 1), q_1^2\right), D1\right)$, après quelques transitions d'un état à l'autre, elle se bloque. En effet, la progression des horloges x1

et x2 est bloquée par les invariants de ces états. En conséquence, il n'existe pas d'exécutions infinies pour D. Donc $L^w(D) = \emptyset$ et donc G_t est diagnosticable.

IV.II Synthèse de diagnostiqueurs pour les systèmes temporisés : cas général

Une différence majeure entre les systèmes temporisés et les SED est que si un système temporisé est diagnosticable, son diagnostiqueur n'est pas forcément un automate temporisé. En effet, les automates temporisés ne sont pas toujours déterminisables. Nous ne pouvons donc pas utiliser systématiquement la méthode de déterminisation utilisée dans le cas des SED.

Dans le cas général, le diagnostiqueur d'un système temporisé est un automate à états finis déterministe muni d'une fonction de décision lui permettant de décider si l'état courant est un état défaillant ou pas. Le calcul de l'état suivant du diagnostiqueur dépend de l'événement observé et du temps écoulé depuis l'événement précédent.

Nous notons s un état du modèle G_t du système qui est de la forme $< q, v >$ avec q une location discrète et v une valuation réelle.

Nous notons $Runs(G_t, \Sigma)$ l'ensemble des exécutions du modèle temporisé G_t projetées sur l'alphabet Σ.

Nous notons l un état du diagnostiqueur. Cet état est un macro-état constitué d'états s de G_t.

Etant donné G_t un automate temporisé, nous supposons que ses locations sont partitionnées en deux sous-ensembles, Q_f l'ensemble des locations relatives à des états défaillants et $Q_{nf} = Q - Q_f$, l'ensemble des locations relatives à des états non défaillants. Pour toute exécution fautive ρ, $\rho(s_0) \in Q_f$. $\rho(s_0)$ est une exécution issue de l'état initial s_0 de G_t. Dans ce cadre, l'objectif est de concevoir un diagnostiqueur qui, en ligne, effectue une estimation du nouvel état atteint en fonction du mot temporisé observé. Pour cela, on définit deux fonctions de transitions f_e et f_t qui permettent de calculer le nouvel état en fonction de l'état courant l et de l'événement observé, ou en fonction de l'état courant l et du temps écoulé depuis l'arrivée dans cet état. On définit également une fonction de décision f_d qui permet de décider si l'état courant est défaillant ou pas.

Etant donné l un état du diagnostiqueur, nous noterons $discret(s)$ une location de $l \cap Q$.

Proposition IV-4 : Diagnostiqueur d'un système temporisé par estimation d'état

Etant donné G_t un automate temporisé bien formé et Δ-diagnosticable avec $\Delta \in \mathbb{N}$. *Un diagnostiqueur pour G_t est un tuple G_d=(L, L_0, f_e, f_t, f_d) avec :*

- *L est l'ensemble des états du diagnostiqueur et chaque état du diagnostiqueur est un ensemble d'états de* G_t *;*
- L_0 *est l'état initial du diagnostiqueur. C'est l'ensemble des états atteignables à partir de* $s_0 =< q_0, 0 >$ *par des exécutions d'événements inobservables au bout d'un délai nul. Formellement, on le définit par :*

$$L_O = \{\rho(s_0)\ \ t.q.\ \ \rho\ \in\ \ Runs(G_t, \Sigma_{uo})\ et\ duree(\rho) = 0\} \qquad \textit{(IV-1)}$$

- f_e *est la fonction de transition événementielle définie par* $f_e : L \otimes\ \Sigma_o \rightarrow L$ *:*

$$\forall\ l\ \in L, \forall\ \sigma\ \in\ \Sigma_{o,}\ f_e(l,\sigma) = \{< q', v' >\ \ t.q. < q, v > \in l\ et\ (q, g, \sigma, r, q') \in E\} \qquad \textit{(IV-2)}$$

- f_t *est la fonction de transition temporisée définie par* $f_t : L \otimes\ \mathbb{R}^+ \rightarrow L$ *telle que :*

$$f_t(l,\delta) = \{\ \rho(s)\ t.q.\ s \in l, \rho \in Runs(G_t, \Sigma_{uo}), duree(\rho) = \delta\} \qquad \textit{(IV-3)}$$

- f_d *est la fonction de décision définie par* f_d: $L \rightarrow \{0,1\}$ *telle que :*

$$\forall\ l\ \in L, \forall < q, v >\ \epsilon\ l, si\ q\ \epsilon\ Q_f\ alors\ f_d(l) = 1 \qquad \textit{(IV-4)}$$

$$\forall\ l\ \in L, si\ \exists < q, v >\ \epsilon\ l\ et\ q\ \notin\ Q_f\ alors\ f_d(l) = 0 \qquad \textit{(IV-5)}$$

La fonction de transition temporelle f_t *doit vérifier la propriété suivante :*

$$\forall\ l\ \epsilon\ L, \forall\ \delta, \delta', \delta^{"}\ \epsilon\ \mathbb{R}^+, si\ \delta = \delta' + \delta^{"}\ alors\ f_t(l,\delta) = f_t(f_t(l,\delta'),\delta^{"}) \qquad \textit{(IV-6)}$$

Ainsi, un mot temporisé $w = \gamma_1, \gamma_2, \ldots, \gamma_n$ sur Σ_o, définit une fonction sur les états du diagnostiqueur. Si $l\ \in L, \rho(l)$ est défini comme le dernier état l_n d'une séquence d'états l_0, l_1, …l_n, où l_0=l. Pour i∈[1..n], si γ_i est un délai δ alors $l_i = f_t(l_{i-1}, \delta)$, si γ_i est un événement observable σ, alors $l_i = f_e(l_{i-1}, \sigma)$.

La Figure IV-9 donne une illustration de la construction d'un diagnostiqueur à partir du modèle du procédé. La fonction de transition d'événement permet d'estimer les états du système potentiellement atteints à partir des états estimés dans l'état courant et pour un événement observé donné. Par exemple, si nous supposons que l'état courant est l'état initial $<q_0,0>$ et que l'événement observé à la date 0 est l'événement 'a', le seul état susceptible d'être atteint est l'état $<q_1,0>$ (i.e. $f_e(<q_0,0>,a)=<q_1,0)$).

La fonction de transition temporisée quant à elle gère les transitions dues à des mots temporisés composés d'événements inobservables. Elle revient donc à calculer le délai qui s'est écoulé entre les deux derniers événements observés et à estimer les états atteints après ce délai. Par exemple, après la génération de l'événement observable 'a', on a le mot temporisé 'w=2*u*1*f*1,5'. Le délai associé à w est de 4,5 u.t. D'où $f_t(< q_1; 0 >, 4{,}5) = < q_3;\ 4{,}5 >$.

On montre que f_e et f_t permettent d'estimer en ligne l'état du diagnostiqueur au fil des événements observés et du temps qui s'écoule. Si l'ensemble des états estimés sont évalués

défaillants alors la fonction de décision diagnostique une faute. C'est par exemple le cas pour <q_4 ; 5,5>, d'où $f_d(< q_4;\ 5{,}5 >) = 1$ (Figure IV-9). Sinon on ne peut rien dire.

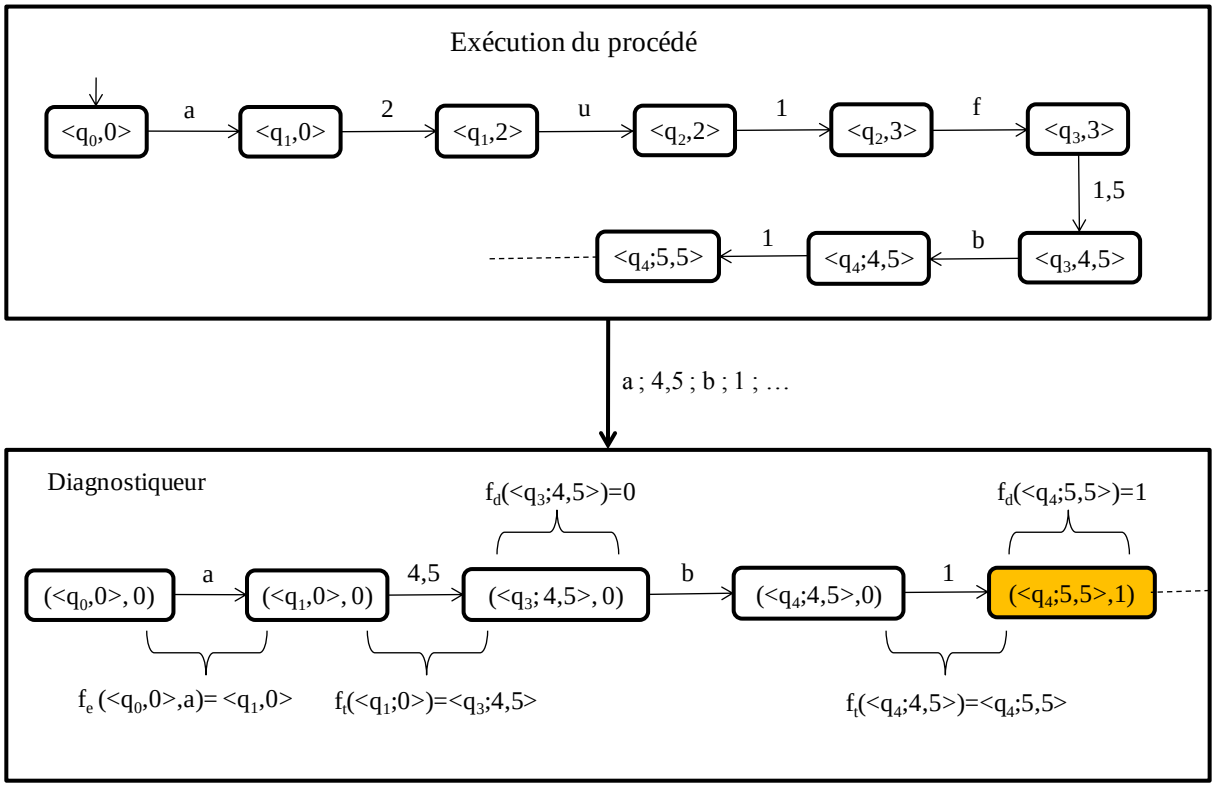


Figure IV-9 – Diagnostiqueur par estimation d'état

L'algorithme nécessaire pour l'estimation d'état est d'une complexité importante. Etant donné, un mot temporisé w, sa complexité est exponentielle par rapport aux nombre d'états de G_t et par rapport à la longueur de w. Cette complexité limite les performances notamment dans le cadre d'un diagnostic en ligne. C'est la raison pour laquelle des travaux sont menés pour trouver des diagnostiqueurs basés sur des automates temporisés déterministes.

IV.III Synthèse de diagnostiqueurs pour les systèmes temporisés non-déterministes

Dans la partie précédente, nous avons étudié une approche de synthèse de diagnostiqueur par estimation d'état (machine de Turing). L'avantage de cette approche est qu'elle est très générale et qu'elle s'applique à tous les modèles de système à base d'automates temporisés. L'inconvénient est sa complexité qui n'en permet pas une exploitation efficace pour le diagnostic en ligne des systèmes complexes. Le problème est que contrairement aux automates à états finis, les automates temporisés ne sont pas tous déterminisables (Alur, et al., 1994). En plus, tester si un automate temporisé est déterminisable est un problème indécidable (Finkel, 2005)(Tripakis, 2006). Face à ce problème, deux approches sont possibles : tester si un automate temporisé déterministe donné peut être le diagnostiqueur d'un système donné, ou identifier des classes d'automates temporisés déterminisables et proposer une heuristique de

déterminisation applicable à ces classes. Avant d'aborder ces problèmes, nous allons donner quelques définitions complétant celles données aux §II.I.2 relatives aux automates temporisés.

Définition IV-7 : Automate temporisé déterministe

Un automate temporisé est déterministe s'il ne contient pas de transitions étiquetés par ε, et si pour toute transition $(q, g_1, \sigma, r_1, q_1)$ et $(q, g_2, \sigma, r_2, q_2)$, les gardes g_1 et g_2 ne peuvent pas être vraies en même temps.

Cette définition a pour conséquence la propriété suivante. Dans un automate temporisé déterministe, pour tout mot temporisé w, il existe au plus une exécution initiale qui génère ce mot temporisé.

Proposition IV-5 : Indécidabilité du test du déterminisme (Finkel, 2005)

Tester si un automate temporisé est déterminisable est un problème indécidable.

Il a toutefois été montré qu'il existe des classes d'automates temporisés qui sont déterminisables (Alur, et al., 1994). On note ATD la classe des automates temporisés déterministes.

Propriété IV-1 : Pouvoir d'expression (Stainer, 2010)

Les automates temporisés déterministes sont strictement moins expressifs que les automates temporisés classiques.

La Propriété IV-1 signifie que l'on ne peut pas spécifier et modéliser n'importe quelle spécification par un automate temporisé déterministe. Donc, on est obligé de permettre l'utilisation d'automates non déterministes qui sont plus expressifs. Cela pose donc le problème en diagnostic de la déterminisation des automates temporisés car un diagnostiqueur est forcément déterministe.

Définition IV-8 : Automate temporisé complet

Un automate temporisé est complet si de tout état et pour toute action, il existe une garde qui est satisfaite.

IV.III.1 Test d'un diagnostiqueur potentiel

Dans certains cas, on peut avoir l'intuition qu'un automate temporisé déterministe soit un diagnostiqueur potentiel d'un système. Pour cela, il faut que la trace du langage généré par le modèle du système soit reconnue par l'automate temporisé déterministe considéré. Il existe différentes sous-classes C de la classe ATD des automates déterministes (Alur, et al., 1994).

On note $NonFaulty\ (G_t)$ l'ensemble des séquences non-fautives d'un automate temporisé G_t et $Faulty\ (G_t)$ ses séquences fautives. D'autre part, étant donné un mot temporisé w généré par G_t, $last(w)$ est le dernier état atteint par la chemin relatif à w.

Définition IV-9 : C-diagnostiqueur (Cassez, et al., 2008)

Soit G_t *un automate temporisé sur* Σ *et* $\Delta \in \mathbb{N}$. *Un* $(\Sigma_o, \Delta) - C - diagnostiqueur$ *pour* G_t *est un automate temporisé déterministe complet* $\Theta = (L, l_o, X, \Sigma_o, E_\Theta, Inv_T, F_\Theta, \Phi)$ *de la classe* C *tel que :*

- *pour tout* $\rho \in NonFaulty\ (G_t)$, $last\left(P_O\left(tr(\rho)\right)\right) \notin F_\Theta$ *(IV-7)*
- *pour tout* $\rho \in Faulty\ (G_t)$, $last\left(P_O\left(tr(\rho)\right)\right) \in F_\Theta$. *(IV-8)*

La Définition IV-9 signifie qu'un C-diagnostiqueur de G_t doit accepter les mots Δ-fautifs et ne doit pas accepter les mots qui ne sont pas Δ-fautifs.

Compte-tenu de cette définition, (Bouyer, et al., 2005a) propose de vérifier si un automate temporisé donné est un diagnostiqueur d'un automate temporisé G_t.

Soit G_t un automate temporisé avec f un événement fautif appartenant à Σ_{uo}. Considérons l'automate temporisé $G_1' = (Q^1 \otimes \{0,1\}, (q_0^1, 0), X_1, \Sigma^1, E_1, Inv_1)$ construit de la manière suivante à partir de G_t :

- Les états sont définis par $\forall\ q_i^1 \epsilon\ Q^1, \exists\ q_i \epsilon Q / q_i^1 \cong q_i$;
- L'état initial est défini par $(q_0^1, 0)$ avec la valeur '0' indiquant qu'initialement le système n'est pas défaillant ;
- X_1 est obtenu en renommant toutes les horloges de X ($x\ renommé\ en\ x_1 / \ x_1 \cong x$);
- Σ_{uo}^1 est obtenu en renommant tous les événements de Σ_{uo} exceptés l'événement f. $\Sigma^1 = \Sigma_o \cup (\Sigma_{uo}^1 \backslash \{f\}) \cup \{f\}$;
- Les transitions sont définies par :
  - $\left((q_1^1, n), g_1, \sigma, r_1, (q_2^1, n)\right) \epsilon\ E_1\ si\ (q_1, g, \sigma, r, q_2)\ \epsilon\ E\ avec\ g_1 \cong g\ et\ r_1 \cong r\ si\ \sigma \neq f$
  - $\left((q_1^1, n), g_1, f, r_1, (q_2^1, 1)\right) \epsilon\ E_1\ si\ (q_1, g, f, r, q_2)\ \epsilon\ E\ avec\ g_1 \cong g\ et\ r_1 \cong r$
- Les invariants des états sont définis par $Inv_1\left((q_i^1, n)\right) = Inv(q_i)$

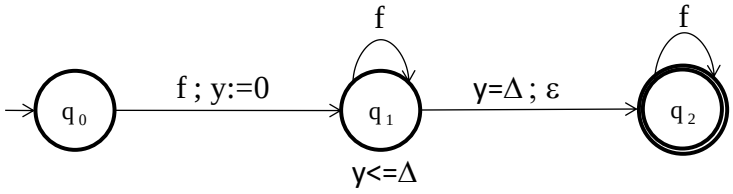


Figure IV-10 – Automate *Obs* (Δ)

On considère d'autre part $Obs(\Delta)$ l'automate de la Figure IV-10. Cet automate permet de mesurer le temps et permet de caractériser l'état du système par rapport à l'occurrence de la faute f et du délai minimal Δ, de détection de cette occurrence. Dans l'état q_0, il n'y a pas encore d'occurrence de f. Sur l'occurrence de f, il y a une transition vers l'état q_1 en initialisant une horloge y. Lorsque y atteint la valeur Δ, le système accède à l'état q_2 qui caractérise des états finaux.

A l'aide des automates G_1' et $Obs(\Delta)$, on construit $G_t(\Delta) = G_1' ||_s Obs(\Delta)$. On définit alors $L_{\Delta,f}$ les états Δ-fautifs de $G_t(\Delta)$ du type $((q,1),q_2)$. Ce sont donc des états pour lesquels on a mémorisé une faute depuis au moins Δ ut. De même, on définit $L_{\text{non\_}f}$ l'ensemble des états du type $((q,0),q_0)$.

Proposition IV-6 : Vérification qu'un automate déterministe est un (Σ_o,Δ)-diagnostiqueur

$\theta = (L, l_o, X_\Theta, \Sigma_O, E_\Theta, Inv_{T,} F_\Theta, \Phi)$ est un (Σ_o,Δ)-diagnostiqueur si pour tout état $\big((q,l),v\big)$ atteignable de $G_t(\Delta)||_s\theta$:

(i) $q \in L_{\Delta,f} \implies l \in F_\theta$;
(ii) $l \in F_\theta \implies q \notin L_{non\_f}$.

Remarque : En pratique cette vérification sera faite à l'aide du graphe des régions de $G_t(\Delta)||_s\theta$.

IV.III.2 Méthode de déterminisation d'un automate temporisé (Baier, et al., 2009)

L'objectif de cette méthode est la transformation sous certaines conditions, d'un automate temporisé non-déterministe, en un automate temporisé déterministe dont le langage généré serait équivalent au langage du modèle source.

La méthode proposée comprend quatre étapes :

- Construction d'un arbre temporisé infini par dépliage
- Construction d'un arbre de régions équivalent
- Déterminisation symbolique
- Réduction du nombre d'horloges

IV.III.2.A Construction d'un arbre temporisé infini par dépliage de l'automate temporisé

Cette première étape consiste à construire un arbre temporisé infini générant le même langage temporisé que l'automate temporisé d'origine. Elle est basée sur le dépliage de l'automate temporisé d'origine. Nous noterons G_t^∞ l'arbre temporisé construit à partir de G_t.

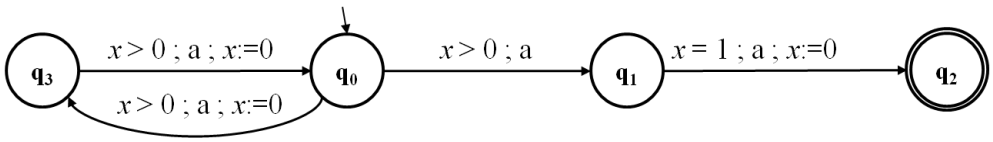


Figure IV-11 – Automate temporisé non déterministe

Le principe du dépliage est le suivant : A chaque niveau 'i' de l'arbre, est associé une nouvelle horloge dont l'objectif est de permettre le codage des contraintes temporelles que

doit satisfaire G_t à ce niveau. Cette horloge que nous noterons z_i est initialisée lors du passage du niveau 'i-1' au niveau 'i'. Soit $Z = (z_{0,} z_{1,} \dots)$ l'ensemble infini des horloges de l'arbre. Chaque nœud n_j de G_t^{∞} est défini par une paire $(q_k, \sigma) \in Q \otimes Z^X$ dans laquelle q_k est un état de G_t et σ est une fonction de $X \to Z$ qui décrit comment chaque horloge de G_t est définie par les horloges de Z.

Considérons l'exemple de l'automate temporisé non-déterministe de la Figure IV-11. Le dépliage de cet automate consiste à associer une horloge z_0 au niveau 0. Cette horloge permet de mémoriser la valeur de x pour la configuration $< q_0, 0 >$. La racine de l'arbre est constituée par le nœud n_0 auquel on associe le label (q_0, z_0) permettant de mémoriser qu'il modélise l'état q_0 avec comme horloge active z_0 dans cette configuration. Désormais pour simplifier la présentation, nous noterons tout simplement le nœud $n_0(q_0, z_0)$. En se référant à l'automate source, nous notons que pour l'action a et la garde $x > 0$, nous avons deux transitions possibles. De ce fait, nous créons deux états $n_1(q_1, z_0)$ et $n_2(q_3, z_1)$. Nous notons que pour le nœud n_1 l'horloge du label est z_0 et non pas z_1. C'est parce que la transition $q_0 \xrightarrow{x>0\ ;a} q_1$ n'effectue pas de remise à zéro de l'horloge x et que dans l'arbre, l'horloge $z_1$est remise à zéro par la transition $n_0(q_0, z_0) \xrightarrow{z_0>0\ ;a\ ;z_1:=0} n_1(q_1, z_0)$. Du coup pour pouvoir exprimer la contrainte temporelle de la transition suivante de l'automate temporisé $q_1 \xrightarrow{x=1\ ;a;x:=0} q_2$, nous sommes obligés dans l'arbre d'utiliser z_0 en tant qu'horloge active

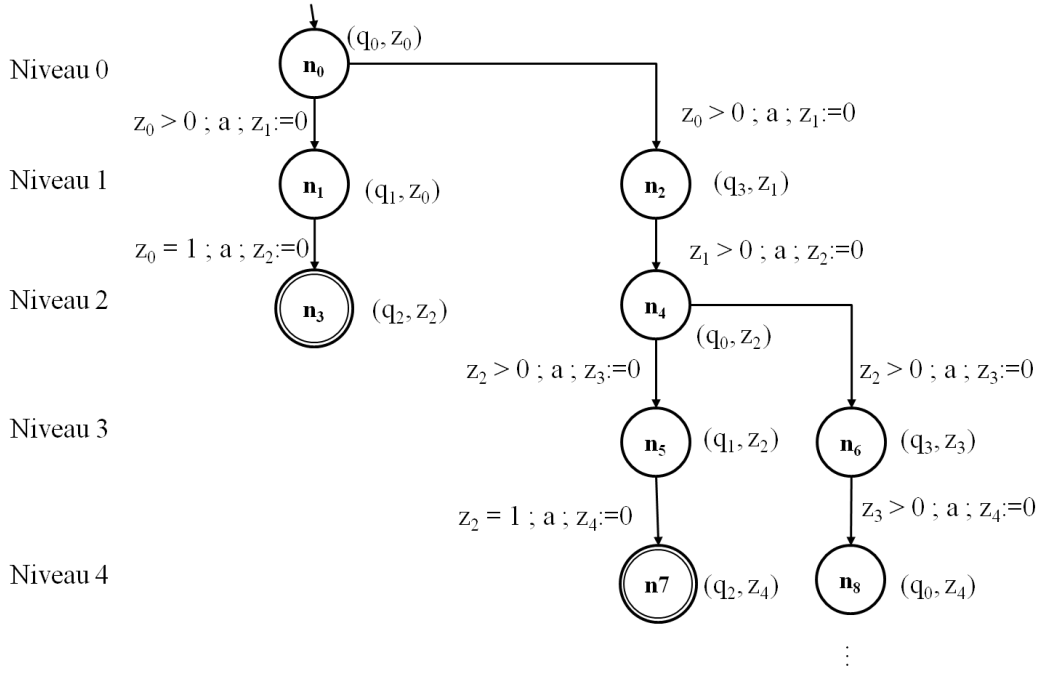


Figure IV-12 – L'arbre temporisé infini Gt∝ associé à l'automate temporisé de la Figure IV-11

La même construction est faite par rapport au nœud n_4 (

Figure IV-12). Ce nœud de l'arbre modélise de nouveau l'état q_0 mais cette fois-ci après avoir transité une fois par l'état q_3. La création de ce nœud n_4 illustre le mécanisme de dépliage dans la mesure où c'est un nœud clone du nœud n_0 au niveau 2 de l'arbre. En conséquence, nous lui associons l'horloge z_2. Sous ce nœud, nous retrouvons donc une structure arborescente similaire à celle obtenue sous le nœud n_0. On notera que les nœuds n_3 et n_7 et d'une manière générale n_{2i-1}, avec 'i' le niveau, sont des nœuds accepteurs car ils modélisent l'état q_2.

La transformation est correcte car la relation entre les états de G_t et ceux de G_t^∞ est une relation de bisimulation forte (Baier, et al., 2009)

IV.III.2.B Construction d'un arbre de régions équivalent à l'arbre temporisé infini

L'objectif de cette étape est d'associer un graphe des régions à l'arbre temporisé infini. Nous notons $R(G_t^\infty)$ l'arbre de régions associé. Le principe de l'abstraction est basé sur le fait qu'à chaque niveau i de l'arbre temporisé infini, seul le sous-ensemble $Z_i = \{z_0, z_1, \dots, z_i\}$ est considéré. Ce sous-ensemble représente les horloges qui sont définies jusqu'au niveau considéré. Les régions correspondant aux états de G_t^∞ au niveau i, doivent donc être définies à l'aide de ces horloges. Le principe de cette abstraction est représenté sur la Figure IV-13. Etant donné R_j^i la région associée au nœud n_j du niveau i et définie sur Z_i. $R_j^{i\to i+1}$ représente une région successeur de R_j^i définie sur Z_i et qui satisfait g_{i+1}. La région R_j^{i+1} est définie sur Z_{i+1}. On doit tenir compte que dans ce cas le passage au niveau $i+1$ entraine l'initialisation de l'horloge z_i. Elle est donc obtenue par :

Pour illustrer la construction de l'arbre des régions, considérons l'arbre temporisé infini de la

Figure IV-12. Au niveau 0, seule l'horloge $z_0$est définie, donc $R_0^0 = (z_0 = 0)$. Nous obtenons alors le nœud $(\{n_0\}, z_0 = 0)$ qui est la racine de l'arbre des régions associé. Nous recherchons ensuite des régions successeurs de R_0^0 qui satisfassent la garde $z_0 > 0$. Comme dans l'automate temporisé G_t, la constante temporelle maximale de l'horloge x est $c_x = 1$, on a par conséquence que $c_{z_0} = 1$. Donc compte tenu de la garde de la transition $n_0 \xrightarrow{z_0>0;a\,;\,z_1:=0} n_1$ une région successeur qui la satisfait est $R_1^{0\to1} = (0 < z_0 < 1)$. En appliquant l'équation (IV-9), nous en déduisons la région $R_1^1 = ((0 < z_0 < 1) \wedge z_1 = 0))$ associée au nœud n_1. Nous obtenons alors le nœud $\{n_1\}, (0 = z_1 < z_0 < 1)$ de l'arbre des régions auquel nous associons l'étiquette (q_1, z_0). Les différents nœuds de cet arbre sont construits comme présenté ici. On obtient l'arbre partiel illustré par la Figure IV-14.

$$\boldsymbol{R_j^{i+1} = R_j^{i\to i+1} \wedge (z_{i+1} = 0)} \qquad \textbf{(IV-9)}$$

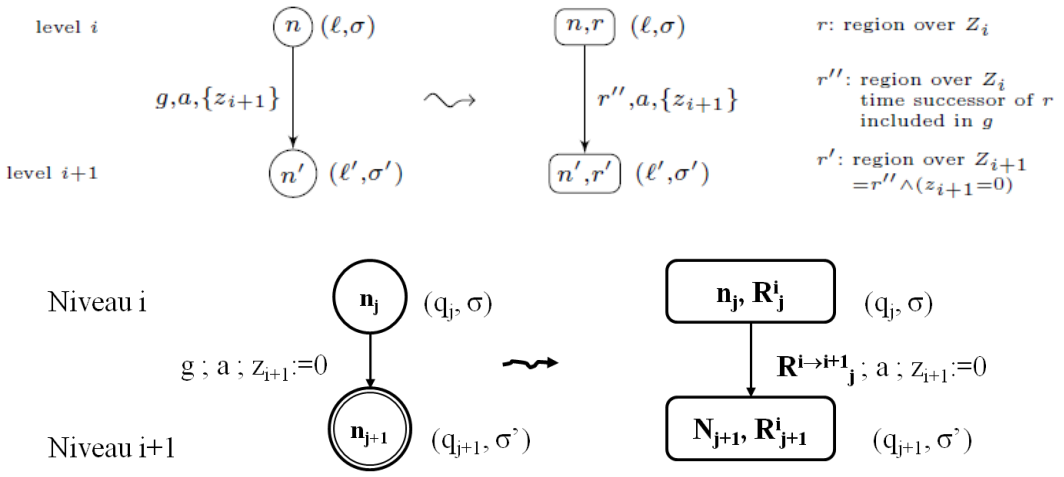


Figure IV-13 – Abstraction de l'arbre temporisé infini en graphe des régions

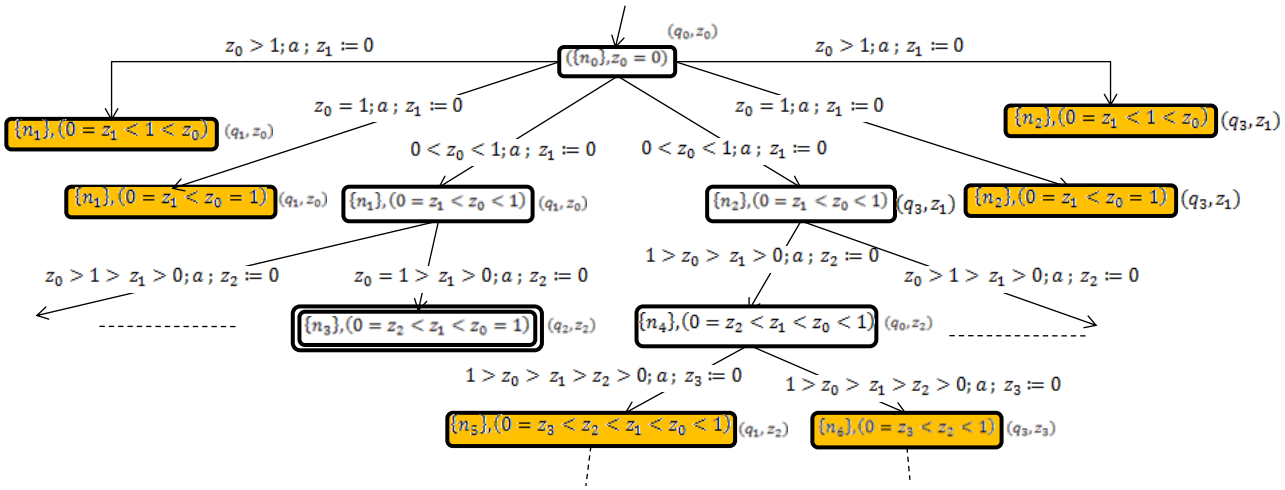


Figure IV-14 – Construction d'un arbre des régions

IV.III.2.C Déterminisation symbolique

Le principe de cette déterminisation symbolique consiste à effectuer des regroupements de nœuds de l'arbre des régions en se basant sur un alphabet symbolique. L'alphabet symbolique est basé sur des paires nœud-région avec les régions définies uniquement à l'aide des horloges actives. Nous appelons nœud symbolique les nœuds du graphe obtenu par déterminisation symbolique. Ces nœuds symboliques sont des sous-ensembles des paires nœud-région d'un même niveau de $R(G_t^{\infty})$. Ils sont sous la forme donnée par les équations (IV-10) et (IV-11):

$$\left(\{n_j^i,\dots,n_k^i\},\ \cup_{q=j}^{k} act(R_q^i)\right) \quad \textbf{(IV-10)}$$

$act(R_q^i)$ une région sur les horloges actives définies par $\sigma_q^i(X)\ t.q.\ label(n_q^i) = (q_q^i,\sigma_q^i)$ **(IV-11)**

Le graphe résultant est un arbre infini que nous notons SymbDet ($R(G_t^\infty)$). Les transitions de ce graphe sont déduites des transitions de $R(G_t^\infty)$ en exprimant la garde de chaque transition à l'aide des horloges actives du nœud source.

Afin d'illustrer cette construction, considérons l'arbre des régions de la Figure IV-14. A partir du nœud racine $\{n_0\}, (z_0 = 0)$ nous avons deux transitions possibles relatives à la garde $1 > z_0 > 0$: ce sont les transitions t1= $\{n_0\}, (z_0 = 0) \xrightarrow{1>z_0>0;a\ ;\ z_1:=0} \{n_1\}, (z_1 = 0 < z_0 < 1)$ et t2 = $\{n_0\}, (z_0 = 0) \xrightarrow{1>z_0>0;a\ ;\ z_1:=0} \{n_2\}, (z_1 = 0 < z_0 < 1)$. Comme l'horloge active de $\{n_1\}, (z_1 = 0 < z_0 < 1)$ est z_0, on peut renommer cette paire nœud-région en $\{n_1\}, (0 < z_0 < 1)$. Par contre l'horloge active associée à $\{n_2\}, (z_1 = 0 < z_0 < 1)$ est z_1. On peut donc renommer le nœud en $\{n_2\}, (z_1 = 0)$. Les transitions t1 et t2 étant équivalentes car elles sont associées à la même garde et à la même action, on peut regrouper les deux nœuds qu'elles permettent d'atteindre selon le principe de l'équation (IV-10). On obtient alors le nœud symbolique $(\{n_1, n_2\}, z_1 = 0 < z_0 < 1)$. Ce nœud symbolique aura comme étiquette la réunion des étiquettes des deux nœuds qui le composent i.e. $\{(q_1, z_0), (q_3, z_1)\}$. On crée alors dans SymbDet ($R(G_t^\infty)$) la transition $(\{n_0\}, z_0 = 0) \xrightarrow{1>z_0>0;a\ ;\ z_1:=0} (\{n_1, n_2\}, z_1 = 0 < z_0 < 1)$.

De même si nous considérons les transitions t3= $\{n_4\}, (0 = z_2 < z_1 < z_0 < 1) \xrightarrow{1>z_0>z_1>z_2>0;a\ ;\ z_3:=0} \{n_5\}, (0 = z_3 < z_2 < z_1 < z_0 < 1)$ et $\{n_4\}, (0 = z_2 < z_1 < z_0 < 1) \xrightarrow{1>z_0>z_1>z_2>0;a\ ;\ z_3:=0} \{n_6\}, (0 = z_3 < z_2 < z_1 < z_0 < 1)$, par une construction analogue nous pouvons construire la transition $(\{n_4\}, z_2 = 0) \xrightarrow{1>z_2>0;a\ ;\ z_3:=0} \{n_5, n_6\}, (0 = z_3 < z_2 < 1)$. Au final nous obtenons l'arbre infini donné par la Figure IV-15.

IV.III.2.D Pliage de l'arbre symbolique des régions en automate temporisé

Ce pliage est basé sur une réduction du nombre d'horloges utilisé par les nœuds de l'arbre symbolique pour faire apparaître des nœuds symboliques équivalents.

Définition IV-10 : borne maximale du nombre d'horloge d'un arbre symbolique

Etant donné $\gamma \in \mathbb{N}$, on dit que SymbDet ($R(G_t^\infty)$) est γ-borné si dans tout pseudo-nœud, le nombre d'horloges est borné par γ.

Si le nombre d'horloges est effectivement borné par γ on peut définir $X_\gamma = \{x_1, x_2, \dots, x_\gamma\}$, un ensemble d'horloges destiné à la ré-écriture des contraintes temporelles de SymbDet ($R(G_t^\infty)$). A partir de sa racine, on renomme de manière déterministe toutes les horloges actives utilisées dans les contraintes temporelles (définition de région et garde) du graphe. Les étiquettes associées à chaque nœud symbolique sont également renommées. En raison de l'hypothèse de la borne supérieure du nombre d'horloge nécessaire pour renommer les horloges actives, chaque fois qu'on a besoin de renommer une horloge, il existera au moins une horloge de X_γ libre et donc utilisable pour cela. Ainsi des nœuds symboliques ayant la

même étiquette et la même définition de région apparaissent dans l'arbre symbolique. De tels nœuds sont isomorphes et fortement bisimilaires sur le plan temporel. En fusionnant ces nœuds, on obtient un automate que nous noterons $BG_{t,\delta}$. Cet automate est un automate temporisé déterministe comme précisé par la Proposition IV 7.

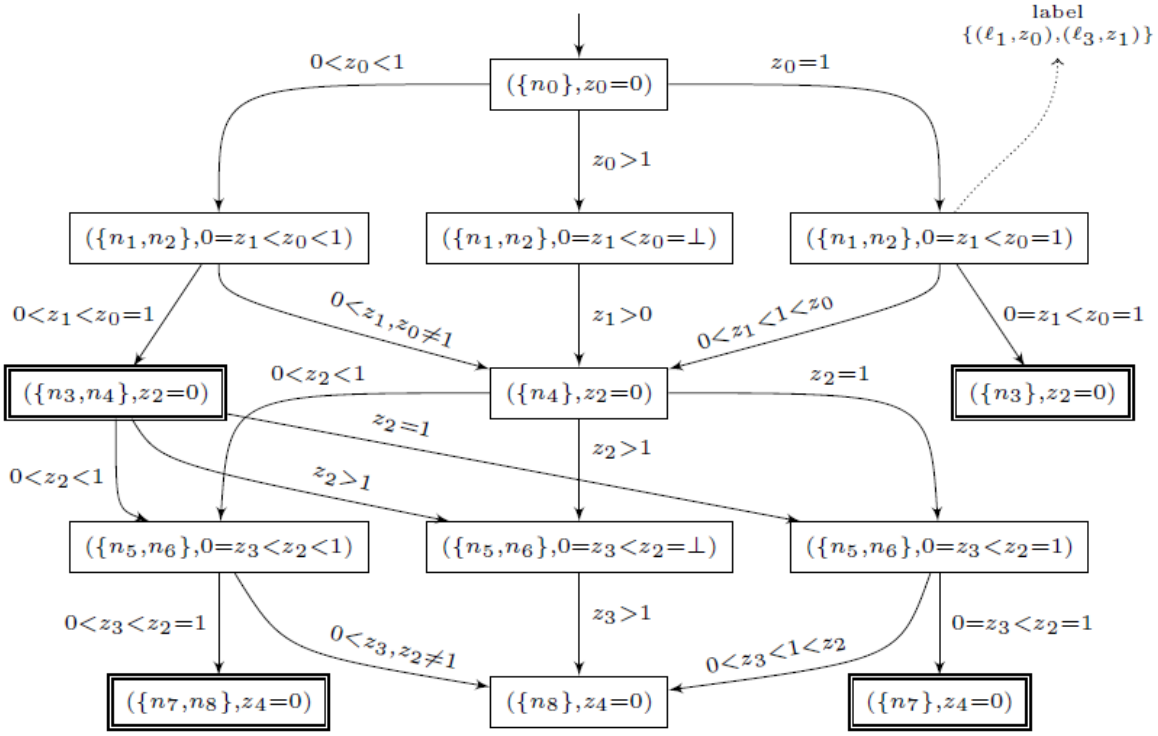


Figure IV-15 – Déterminisation symbolique de l'arbre des régions de la Figure IV-14

Proposition IV-7 : Automate déterminisé

Supposons que SymbDet ($R(G_t^{\infty})$) soit γ-borné, alors $BG_{t,\delta}$ est un automate temporisé déterministe, et $L(BG_{t,\delta}) = L(G_t)$.

Considérons de nouveau le cas de l'exemple de la Figure IV-11 et notamment la déterminisation symbolique qui en résulte. Nous pouvons remarquer que les contraintes temporelles du modèle (Figure IV-15) n'utilisent jamais plus que deux horloges (horloges actives). Donc, nous pouvons dire que ce modèle est 2-borné. Nous pouvons ainsi définir $X_2 = \{x_1, x_2\}$ pour renommer les horloges du modèle. Par exemple, l'horloge z_0 du modèle sera renommée en x_1. Au niveau 1, l'horloge z_1 sera renommée x_2. Au niveau 2, comme x_2 sert déjà à renommer z_1 et que l'horloge z_0 n'est pas utilisée, $x_1$est utilisée pour renommer z_2 et ainsi de suite … Après le renommage des horloges, certains nœuds symboliques sont caractérisés par la même région et la même étiquette. Ainsi, le renommage de z_0 et le remplacement de n_0 par son étiquette transforme $(\{n_0\}, z_0 = 0)$ en $(\{(q_0, x_1)\}, x_1 = 0)$. De même, le nœud $(\{n_4\}, z_2 = 0)$ est ré-écrit en $(\{(q_0, x_1)\}, x_1 = 0)$. On voit donc que la récriture de ces nœuds fait qu'ils sont équivalents. Ils peuvent donc être

fusionnés. L'opération de fusion est étendue à tous les nœuds équivalents par la ré-écriture. Les gardes et initialisation des transitions sont aussi ré-écrites. Ainsi, la transition $(\{n_0\}, z_0 = 0) \xrightarrow{1>z_0>0;a\ ;\ z_1:=0} (\{n_1, n_2\}, z_1 = 0 < z_0 < 1)$ est ré-écrite en $(\{(q_0,\ x_1)\}, x_1 = 0) \xrightarrow{1>x_1>0;a\ ;\ x_2:=0} (\{(q_3,\ x_1), (q_3,\ x_2)\}, x_2 = 0 < x_1 < 1)$. Inversement, la transition $(\{n_1, n_2\}, z_1 = 0 < z_0 < 1) \xrightarrow{z_1>0,\ z_0\neq 1\ ;a\ ;\ z_2:=0} (\{n_4\}, z_2 = 0)$ est ré-écrite en $(\{(q_3,\ x_1), (q_3,\ x_2)\}, x_2 = 0 < x_1 < 1) \xrightarrow{x_2>0\ ;a\ ;\ x_1:=0} (\{(q_0,\ x_1)\}, x_1 = 0)$. Par itération de ce processus de ré-écriture à l'ensemble des nœuds et transitions du modèle, nous obtenons le modèle de la Figure IV-16.

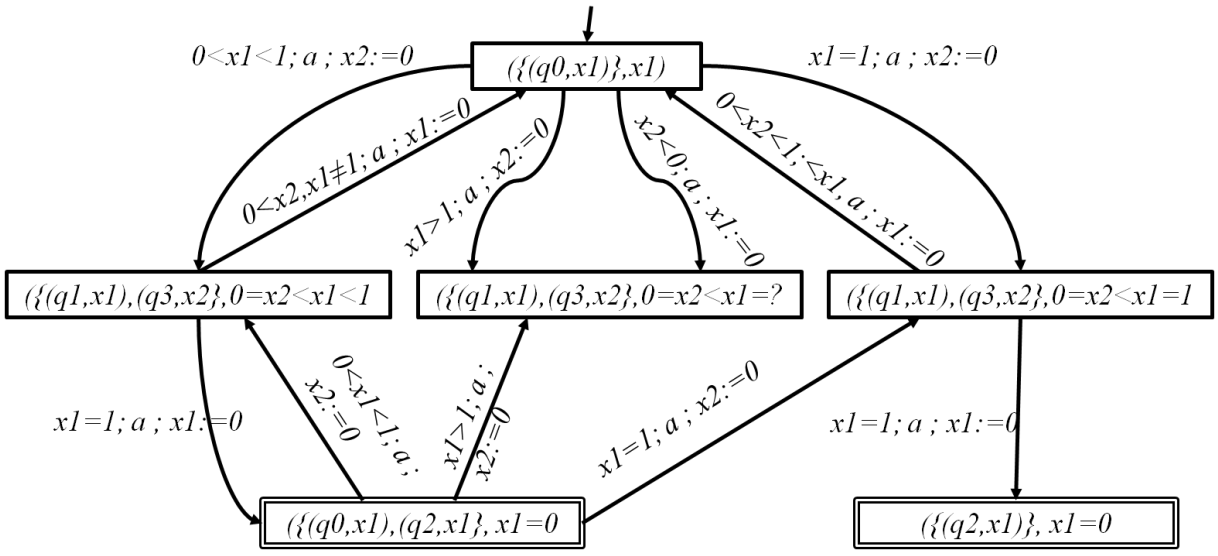


Figure IV-16 – Automate temporisé déterministe $BG_{t,2}$ obtenu par déterminisation de G_t

IV.III.2.E Classes d'automates déterminisables

Dans la partie précédente la seule hypothèse que nous ayons faite porte sur le nombre fini d'horloges actives que doit compter SymbDet ($R(G_t^{\infty})$). Nous pouvons donc chercher à identifier les classes d'automates indéterministes qui sont déterminisables par cette méthode. Quatre classes d'automates ont été identifiées : les automates temporisés de type p-hypothèse (pG_t), les automates temporisés fortement non-zénon (fnG_t), les automates temporisés à événements datés (EDG_t) et les automates temporisés avec ré-initialisation des entiers (IEG_t).

Définition IV-11 : Automate temporisé de type p-hypothèse (pG_t)

Etant donné $p \in \mathbb{N}$, un automate temporisé G_t vérifie la p-hypothèse si pour tout $n \geq p$, pour toute exécution $\rho = < q_0, v_0 > \xrightarrow{g_1\ ;a_1;\ r_1}_{d_1} < q_1, v_1 > \cdots < q_{n-1}, v_{n-1} > \xrightarrow{g_n\ ;a_n;\ r_n}_{d_n} < q_n, v_n >$

, pour toute horloge $x \in X$, alors x est réinitialisé le long de ρ ou $v_n(x) = \Omega$ avec Ω symbolisant une très grande valeur.

On montre que si un automate temporisé G_t vérifie la p-hypothèse alors son SymbDet($R(G_t^\infty)$) est p-borné par rapport aux horloges (Baier, et al., 2009). Donc ce type d'automate temporisé est déterminisable par la méthode présentée.

Proposition IV-8 : Automate temporisé vérifiant la p-hypothèse

Un automate temporisé fortement non-zénon vérifie la p-hypothèse pour $p \in \mathbb{N}$.

Proposition IV-9 : Construction d'un automate déterministe

Pour tout automate temporisé non-déterministe vérifiant la p-hypothèse, nous pouvons construire un automate temporisé déterministe $BG_{t,\gamma}$ dont la taille est doublement exponentielle par rapport à la taille de l'automate temporisé initial G_t, et qui reconnait le même langage que cet automate.

Définition IV-12 : Automate temporisé à événements datés (EDG_t) (Alur, et al., 1999)(Baier, et al., 2009)

C'est un automate temporisé qui ne contient que des horloges mémorisant l'occurrence d'événements. Pour chaque événement de référence $a \in \Sigma$, il existe une horloge x_a qui est réinitialisée à chaque occurrence de a et qui permet de dater des événements postérieurs par rapport à l'événement de référence.

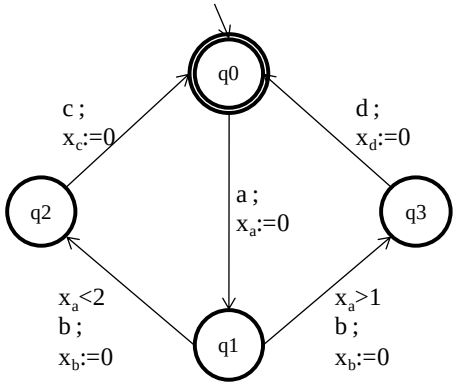


Figure IV-17 – Exemple d'automate EDG_t non déterministe (Alur, et al., 1999)

La Figure IV-17 est un exemple d'automate de type EDG_t. Nous pouvons remarquer que l'occurrence de chaque événement permet d'initialiser une horloge. Dans cet exemple, seule l'horloge x_a est vraiment utile pour spécifier les gardes des deux transitions issues de l'état q_1. Le comportement de cet automate est indéterminé pour $1 < x_a < 2$.

Nous montrons que pour ce type d'automate temporisé, l'arbre temporisé déterministe G_t^∞ est $|\Sigma| - borné$ par rapport aux horloges. D'où la proposition suivante :

Proposition IV-10 : Déterminisation d'un automate temporisé EDG_t

Pour tout automate temporisé de type EDG_t, on peut construire un automate temporisé déterministe $BG_{t,\gamma}$ dont la taille est exponentielle par rapport à la taille de l'automate temporisé initial G_t et qui reconnait le même langage.

Considérons de nouveau l'automate temporisé non déterministe de la Figure IV-17. Nous pouvons le déterminiser en construisant l'automate temporisé de la Figure IV-18. Pour cela nous ajoutons l'état $< q_2, q_3 >$ qui est accessible à partir de q_1 sur l'occurrence de l'événement b rendant indéterministe l'automate temporisé initial. La transition $q_1 \xrightarrow{2>x_a>1;b;x_b:=0} < q_2, q_3 >$ a une garde qui lève l'indéterministe dû aux gardes des deux transitions issues de q_1 dans l'automate temporisé initial.

Définition IV-13 : Automate temporisé avec ré-initialisation des entiers (IEG_t) (Suman, et al., 2008)(Baier, et al., 2009)

C'est un automate temporisé dans lequel toute transition $(q_{i-1}, g_i, a_i, r_i, q_i) \in E$ est telle que r_i est non vide si et seulement si g_i contient au moins une contrainte temporelle atomique de la forme $x = c$, pour certaines horloges x.

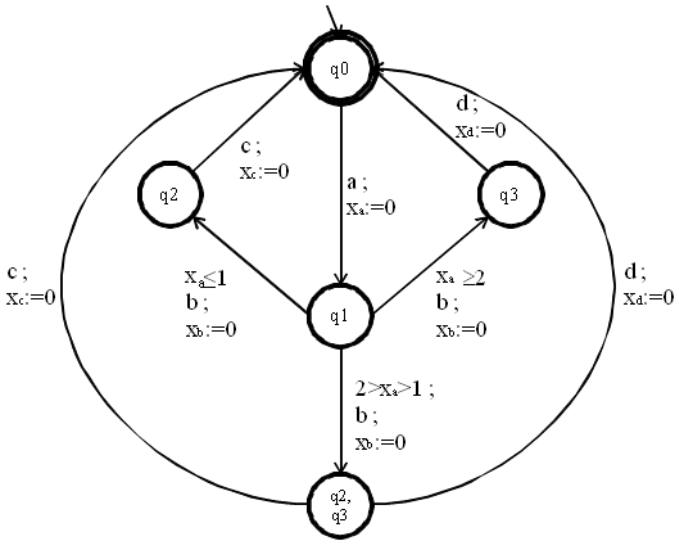


Figure IV-18 – Automate temporisé déterministe construit à partir de l'exemple de la Figure IV-17

On montre que pour ce type d'automate temporisé, l'arbre temporisé déterministe G_t^{∞} est $|\mathrm{M}+1|-born\acute{e}$ par rapport aux horloges. D'où la proposition suivante :

Proposition IV-11 : Déterminisation d'un automate temporisé IEG_t

Pour tout automate temporisé de type IEG_t, on peut construire un automate temporisé déterministe $BG_{t,\gamma}$dont la taille est doublement exponentielle par rapport à la taille de l'automate temporisé initial $G_t$et qui reconnait le même langage.

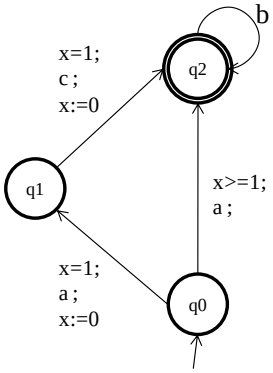


Figure IV-19 – Exemple d'automate IEG_t non déterministe (Suman et al., 2008)

IV.IV Calcul du délai minimal pour diagnostiquer une faute

Dans les parties précédentes, nous avons introduit le paramètre $\Delta \in \mathbb{N}$ qui correspond au délai minimal nécessaire pour diagnostiquer toute faute dans le système considéré. Ce paramètre est important car il permet au concepteur de borner la réactivité de son système de diagnostic. Si cette réactivité n'est pas conforme au cahier des charges, il peut être amené à enrichir les capacités d'observation du système. Dans cette partie, nous allons proposer une méthode permettant le calcul de ce paramètre.

La méthode est basée sur la construction du modèle $G_t'(\Delta) = G_1' ||_s G_2 ||_s Obs(\Delta)$. G_1' et $Obs(\Delta)$ correspondent aux modèles présenté dans le paragraphe §IV.III.1. Par construction, G_1' contient les exécutions Δ-fautives de G_t et G_2 les exécutions qui ne sont pas Δ-fautives. $Obs(\Delta)$ quant à lui est un automate temporisé qui permet de mesurer qu'il s'est écoulé Δ u.t. depuis la première occurrence d'une faute f donnée (Figure IV-10). Si l'on peut atteindre l'état q_2 de $Obs(\Delta)$ cela signifie que G_t n'est pas (Σ_o,Δ)–diagnosticable. Si l'on sait par ailleurs que G_t est Σ-diagnosticable, cela signifie qu'il faut prendre une valeur de Δ plus importante. En effet, on montre que pour tout $\Delta, \Delta' \in \mathbb{N}\ t.q.\ \Delta < \Delta'$ si G_t est (Σ_o,Δ)–diagnosticable alors G_t est (Σ_o,Δ')–diagnosticable. Cette propriété permet donc d'effectuer une recherche basée sur une approche dichotomique.

Le principe de la méthode consiste dans un premier temps à rechercher le plus petit k $\in \mathbb{N}\ t.q. \Delta = 2^k$ et G_t est $(\Sigma_O, 2^k)$–diagnosticable. Appelons k_{min} cet entier. Donc G_t est $(\Sigma, 2^{k_{min}})$–diagnosticable mais n'est pas $(\Sigma, 2^{k_{min}-1})$–diagnosticable.

Dans un deuxième temps, il suffit de rechercher par incrémentation $\Delta_{opt} \in [2^{k_{min}-1} + 1, 2^{k_{min}}]$ tel que $G_t\ est\ (\Sigma, \Delta_{opt})$–diagnosticable mais G_t n'est pas $(\Sigma, \Delta_{opt} - 1)$–diagnosticable.

IV.V Test de diagnosticabilité des systèmes temporisés par model- checking

Dans cette partie, nous définissons dans un premier temps le model-checking. Ensuite, nous présentons la logique temporelle et la logique temporelle temporisée TCTL. Enfin, nous présentons les deux principaux outils de model-checking que nous avons utilisés KRONOS et UPPAAL.

IV.V.1 Présentation du model-ckecking

Le model-checking est une technique permettant de vérifier si un modèle $\mathcal{M}$ d'un système vérifie une propriété φ : on note $\mathcal{M} \vDash \varphi$. L'approche classique consiste à modéliser $\mathcal{M}$ par un système de transitions et φ par une formule temporelle. Le model-checking permet ainsi de vérifier si le modèle contient des erreurs ou vérifie une spécification. Par exemple, le model-checking est une technique qui est de plus en plus utilisée pour la vérification des logiciels. Elle est plus efficace que le test dans l'identification de bogue. En effet, quand la propriété n'est pas vérifiée le model-checking peut donner un contre-exemple pour expliquer cette non vérification.

Il existe deux méthodes pour évaluer si le modèle satisfait la propriété : soit par un programme dédié, soit par l'utilisation d'un model-checker. Dans ce travail nous avons choisi l'exploitation de model-checkers.

Le model-checking permet de vérifier six catégories de propriétés :

- L'accessibilité : C'est la propriété de base que tout model-checker doit pouvoir vérifier. C'est la capacité à atteindre un état donné du système.
- L'invariance : Chaque état respecte une bonne propriété.
- La sûreté : C'est la certitude que quelque soit l'évolution du système, rien de mauvais n'arrivera jamais.
- La vivacité : C'est la certitude que le système pourra toujours évoluer vers une situation positive. Par exemple la capacité à toujours pouvoir réinitialiser un système est une propriété de vivacité.
- L'équité : C'est la capacité pour le système à répéter infiniment souvent une action. On peut par exemple illustrer cela par la capacité d'un système à obtenir infiniment souvent une ressource qu'il demande.
- L'équivalence comportementale : C'est la capacité à montrer que deux systèmes sont équivalents.

Ces propriétés peuvent être exprimées de deux manières : avec une logique temporelle (LTL, CTL, CTL*, …), ou temporisée (TCTL, …) ou par un modèle états-transitions. Dans ce travail, nous avons choisi de les exprimer en logique temporelle. Cela permet d'avoir des formulations non ambigües et génériques permettant d'automatiser la vérification.

La principale limite du model-checking est liée à l'explosion combinatoire lors de la vérification d'une propriété. Pour éviter cette explosion combinatoire le système de transitions doit être fini et suffisamment petit. Un système de transitions est fini si par exemple, il est basé sur des variables à domaines finis, ou s'il modélise un nombre borné de composants dont les états sont finis. Le système est petit si l'on peut stocker l'ensemble de ses états.

La model-checking est un processus en trois étapes. La première étape consiste en la modélisation du système (création du modèle $\mathcal{M}$) et en la formulation de la propriété à vérifier φ. La deuxième étape est la vérification proprement dite. Si cette vérification est positive, il faudra dans un second temps vérifier le lien avec le système réel. Quand la vérification échoue, le model-checker fournit un contre-exemple. La 3ème étape consiste à analyser ce contre-exemple en vérifiant qu'il correspond à une vraie erreur sur le système réel. Si ce n'est pas une vraie erreur, cela signifie qu'il faut corriger ou affiner le modèle $\mathcal{M}$.

Dans cette thèse, nous avons choisi d'utiliser le model-checking pour vérifier un certain nombre des propriétés du modèle du système à diagnostiquer ou des modèles construits pour établir sa diagnosticabilité voire pour calculer le délai maximum nécessaire pour diagnostiquer une défaillance. La principale propriété que nous serons amenés à vérifier est l'accessibilité de certains états. Nous chercherons également à vérifier si a priori un automate temporisé est bien formé ou pas.

IV.V.2 Logique temporelle

Dans le cadre du model-checking, on exprime généralement les propriétés à vérifier à l'aide d'une logique temporelle. Les logiques temporelles permettent d'exprimer des propriétés génériques en les positionnant dans le temps. Certaines logiques comme TCTL permettent même d'exprimer des contraintes de temps explicites. L'avantage des logiques temporelles par rapport au langage naturel est qu'elles sont non ambigües et elles permettent donc d'automatiser la démarche de vérification. Par rapport aux formalismes graphiques, elles ont souvent une plus grande précision et expressivité.

Une logique temporelle peut se définir comme une logique standard à laquelle on ajoute des opérateurs dédiés au temps. On distingue deux types d'opérateurs temporels : des connecteurs temporels et des quantificateurs de chemins.

IV.V.2.A Les connecteurs temporels

Un chemin ou une exécution représente une séquence d'états dans lesquels transite le système sur l'occurrence d'événements. On distingue deux catégories de logiques temporelles : les logiques linéaires et les logiques réifiées. Dans les logiques linéaires, il n'existe qu'un seul chemin dans le futur. Dans les logiques temporelles réifiées, l'état courant est la source de plusieurs chemins, donc de plusieurs futurs possibles. Les connecteurs temporels s'intéressent

à l'occurrence des événements relatifs à un seul chemin. On considére en général 4 types de connecteurs : U (up to- jusqu'à), X (neXt – suivant), G (désormais), F (finalement).

Le connecteur X exprime le fait qu'une propriété φ sera vraie dans l'état suivant l'état courant sur le chemin considéré (Figure IV-20a). Nous le notons $X\varphi$.

Le connecteur F (finalement) exprime le fait que sur le chemin considéré il existe un état futur où la propriété φ sera vérifiée (Figure IV-20b). On le note $F\varphi$. On pout ainsi utiliser ce connecteur pour spécifier les propriétés d'accessibilité. Considérons par exemple un système qui exécute une tâche qui doit se terminer. On peut spécifier ce comportement par $F(statut = fin)$, ce qui signifie que dans le futur le système va atteindre un état dont le statut sera fin.

Le connecteur G (dorénavant) exprime le fait que sur le chemin considéré à partir de l'état courant la propriété φ sera toujours vérifiée (Figure IV-20c). On le note $G\varphi$. En utilisant ce connecteur, on peut exprimer des propriétés d'invariance. Par exemple, on peut spécifier qu'une tâche ne se termine jamais par $G{\sim}(statut = fin)$. Le symbole $\sim$ représente ici la négation. Ce connecteur permet également d'exprimer la vivacité d'un système. On peut spécifier que si une faute apparait dans un système non réparable, il ne pourra plus avoir un fonctionnement normal par $G(faute \rightarrow F(status = anormal))$.

Le connecteur U (Up to – jusqu'à) exprime que la propriété φ est vérifiée jusqu'à ce que la propriété ψ le devienne (Figure IV-20d). On le note $\varphi U \psi$.

IV.V.2.B Les quantificateurs de chemins

Les connecteurs temporels s'intéressent à une seule exécution à la fois. Or, il se trouve que certains systèmes peuvent avoir plusieurs futurs possibles en fonction de certaines actions. On parle de futur réifié qui s'exprime par plusieurs exécutions ou chemins possibles à partir de l'état courant. On représente cela par une structure arborescente. Il peut donc être intéressant de vérifier si une propriété est vérifiée quelque soit l'évolution future. Cela amène à introduire des quantificateurs de chemins. Il existe deux quantificateurs de chemins :

- A : tous les chemins futurs vérifient la propriété ;
- E : il existe un chemin futur qui vérifie la propriété.

Mais ces quantificateurs doivent être utilisés conjointement avec les connecteurs temporels.

Considérons par exemple la spécification temporelle $EF\varphi$. Elle signifie qu'il existe une exécution pour laquelle dans le futur il y aura un état pour lequel la propriété sera vérifiée (Figure IV-21).

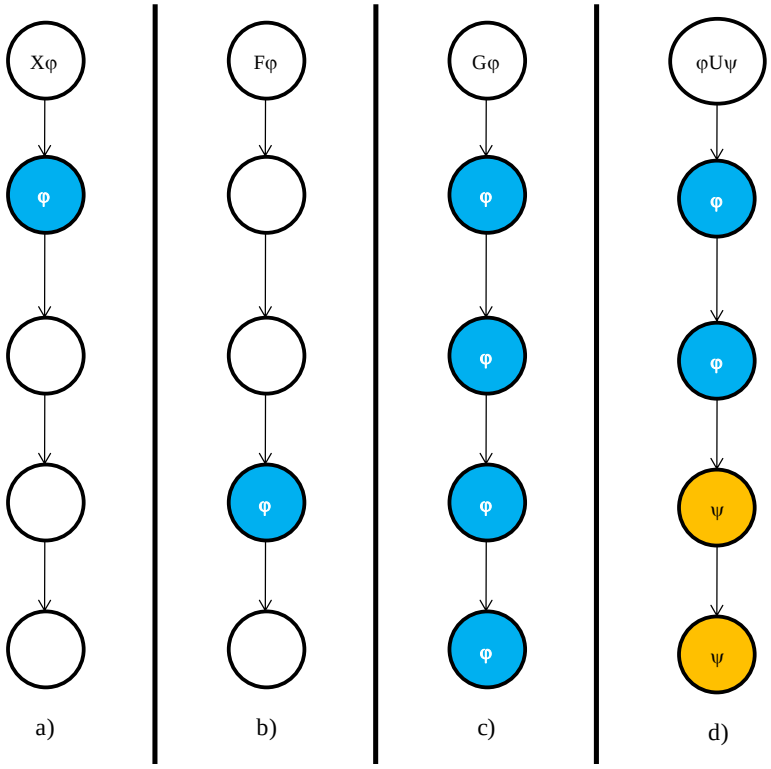


Figure IV-20 – Illustration des différents connecteurs temporels

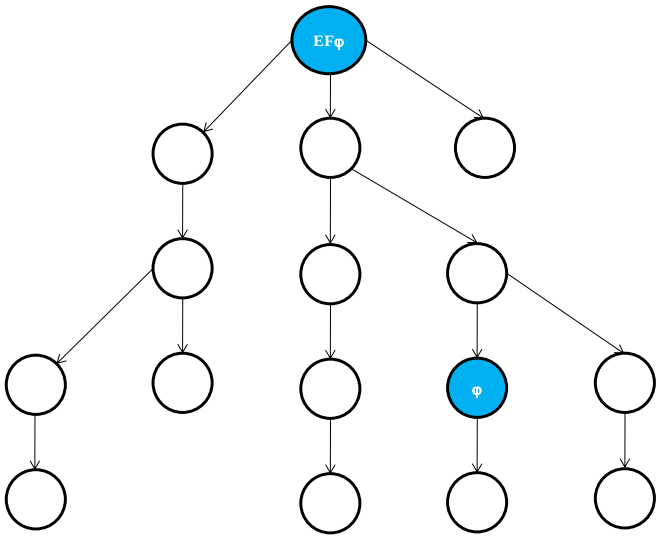


Figure IV-21 – Futur réifié et quantificateur E

Nous voyons au travers de cet exemple que le quantificateur E permet de sélectionner un chemin. Ainsi, une spécification du type $EG\varphi$ signifie qu'il existe une exécution pour laquelle la propriété φ est vérifiée dans tous les états (Figure IV-22).

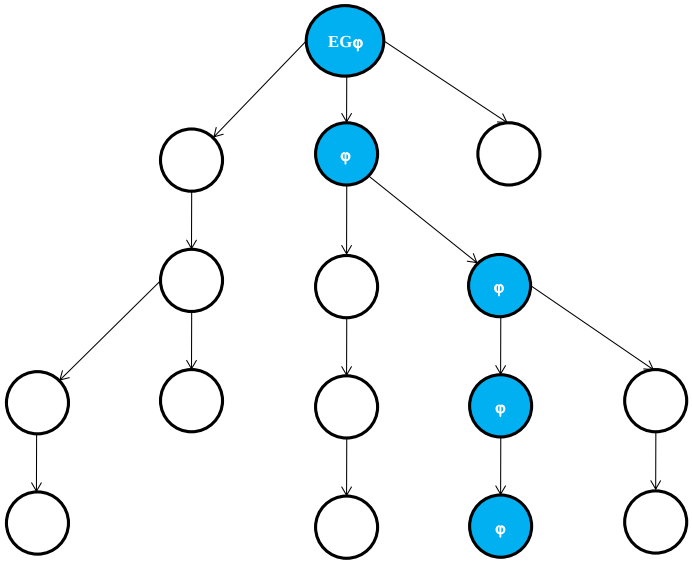


Figure IV-22 – Quantificateur E associé au connecteur G

Nous pouvons de même utiliser le quantificateur A avec les connecteurs F et G. Une spécification du type $AF\varphi$, signifie que quelque soit l'exécution, il existe un état pour lequel la propriété φ sera vérifiée (Figure IV-23).

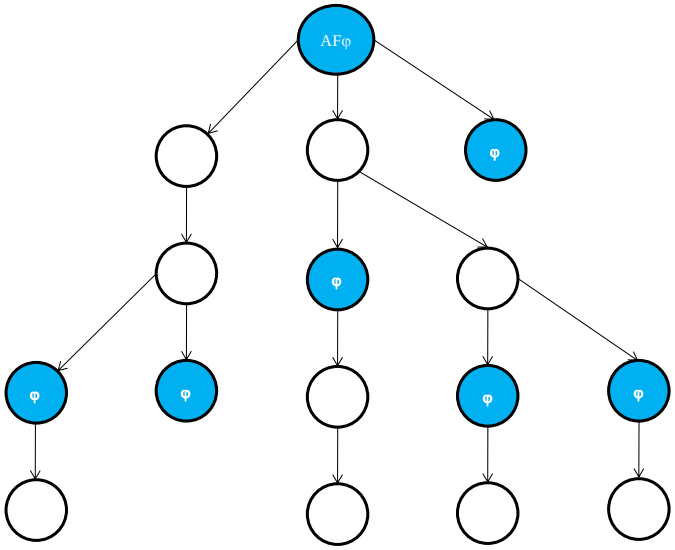


Figure IV-23 – Quantificateur A associé au connecteur F

Par contre la spécification $AG\varphi$ signifie que pour toutes les exécutions, la propriété φ est vérifiée dans tous les états (Figure IV-24).

Dans la littérature, on présente en général deux logiques temporelles : LTL (Linear Temporal Logic) qui est une logique du temps linéaire (Pnueli, 1977) et CTL (Computation Tree Logic) qui est une logique du temps arborescent (Clarke, et al., 1981).

Si nous considérons un ensemble Prop de propositions atomiques, les formules de LTL sont définies par la grammaire :

$$\varphi, \psi ::= P \mid \sim \varphi \mid \varphi \wedge \psi \mid X\varphi \mid \varphi U \psi \quad \textbf{(IV-12)}$$

avec $P \in Prop$.

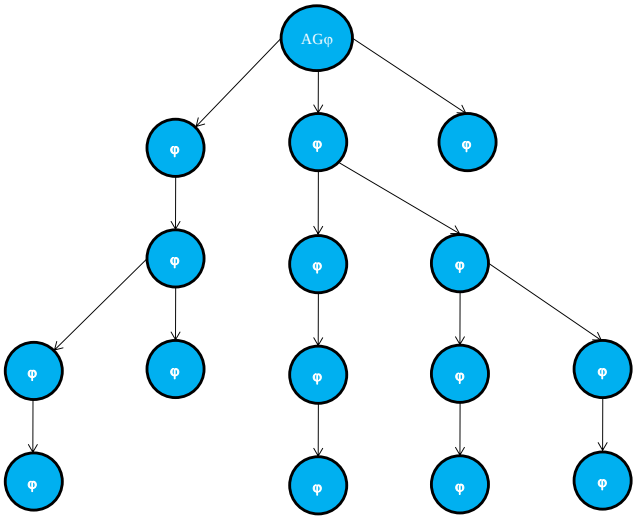


Figure IV-24 – Quantificateur A associé au connecteur G

Les formules de LTL s'interprètent pour une exécution σ partant de l'état initial. Si on admet que $\sigma(i)$ représente le i^{ème} état de σ (i.e. q_i), on définit alors inductivement la sémantique de LTL par :

- $\sigma, i \vDash P$ si P est dans l'ensemble $l(\sigma(i))$ des étiquettes de q_i ;
- $\sigma, i \vDash \sim \varphi$ si σ, i ne satisfait pas φ ;
- $\sigma, i \vDash \varphi \wedge \psi$ si q_i satisfait à la fois φ et ψ ;
- $\sigma, i \vDash X\varphi$ si $\sigma, i+1 \vDash \varphi$;
- $\sigma, i \vDash \varphi U \psi$ s'il existe $j \geq i$ tel que $\sigma, j \vDash \psi$ et pout tout $k, i \leq k < j,\ \sigma, k \vDash \varphi$.

Nous pouvons remarquer que dans cette logique les connecteurs temporels F et G n'avaient pas été définis. Ils se déduisent avec les connecteurs présentés ci-dessus par :

$F\varphi \approx trueU\varphi$ qui signifie qu'à partir de l'état courant, $true$ est vrai jusqu'à ce que φ le soit. En d'autres termes, comme $true$ est satisfaite dans tous les états, cela signifie que dans le futur, il existera un état où φ sera satisfait.

De même $G\varphi \approx\sim F(\sim \varphi)$, ce qui exprime que dans tous les états à partir de l'état courant, φ est toujours vrai.

En utilisant ces connecteurs, nous pouvons exprimer que toute occurrence d'une faute dans le système, sera suivie par un arrêt du système.

$$G(faute \Longrightarrow Farrêt) \quad \textbf{(IV-13)}$$

La logique CTL étant une logique basée sur un temps arborescent. Elle utilise les quantificateurs A et E. Elle impose que les connecteurs temporels soient toujours précédés des quantificateurs temporels. Cela a du sens dans la mesure où les quantificateurs permettent de sélectionner le ou les exécutions auxquelles vont être appliquées les connecteurs temporels.

La syntaxe de CTL est définie par la grammaire donnée par l'équation (IV-14).

$$\varphi, \psi ::= P | \sim \varphi | \varphi \wedge \psi | EX\varphi | AX\varphi | E\varphi U\psi | A\varphi U\psi \quad \textbf{(IV-14)}$$

Nous adoptons les notations suivantes : pour $j \geq 0, \sigma_q^j$ représente une exécution issue de l'état courant q et pour $i \geq 0, \sigma_q^j(i)$ représente le i[ème] état de σ_q^j.

- $\sigma, j, i \vDash P$ si P est dans l'ensemble $l(\sigma_q^j(i))$ des étiquettes de $\sigma_q^j(i)$;
- $\sigma, j, i \vDash \sim \varphi$ si σ, j, i ne satisfait pas φ ;
- $\sigma, j, i \vDash \varphi \wedge \psi$ si $q_i^j(i)$ satisfait à la fois φ et ψ ;
- $q \vDash EX\varphi$ s'il existe une exécution $\sigma_q^j\ t.q.\ \sigma_q^j(1) \vDash \varphi$;
- $q \vDash AX\varphi$ si toute exécution σ_q^j est telle que $\sigma_q^j(1) \vDash \varphi$;
- $q \vDash E\varphi U\psi$ s'il existe une exécution σ_q^j issue de q et $i \geq 0$ tel que $\sigma_q^j(i) \vDash \psi$ et pour tout $k, 0 \leq k < i,\ \sigma_q^j(k) \vDash \varphi$.
- $q \vDash A\varphi U\psi$ si pour toute exécution σ_q^j issue de q, il existe un entier $i \geq 0$ tel que $\sigma_q^j(i) \vDash \psi$ et pour tout $k, 0 \leq k < i,\ \sigma_q^j(k) \vDash \varphi$.

Nous spécifions « à l'occurrence d'une faute, le système atteindra un état dans lequel il va s'arrêter quel que soit l'évolution du système » par ce qui suit :

$$AG(faute \Longrightarrow AF\ arrêt) \quad \textbf{(IV-15)}$$

Ces logiques ne permettent pas d'exprimer de manière explicite une durée. Par exemple, nous souhaitons spécifier que dans un délai de moins de 10 ut le système s'arrête après l'occurrence d'une faute. C'est la raison pour laquelle ont été proposées des extensions temporisées de ces logiques. Dans ce travail nous utilisons l'extension TCTL.

IV.V.2.C Logique temporisée TCTL (Alur, et al., 1991)(Alur, et al., 1994)(Bérard, 2008)

TCTL (Timed Computation Tree Logic) est une extension temporisée de CTL. Le domaine du temps est $\mathbb{R}^+$ et les exécutions sont supposées infinies. Cette logique est interprétée sur des systèmes de transitions temporisés.

La syntaxe de TCTL correspond à la grammaire (IV-16):

$$\varphi, \psi ::= P | \sim \varphi \, | \varphi \wedge \psi \, | \, E\varphi U_{\bowtie c} \psi \, | A\varphi U_{\bowtie c} \psi \qquad \textbf{(IV-16)}$$

où P est une proposition atomique de $Prop$, l'opérateur $\bowtie \in \{<, \leq, >, \geq\}$, et c est une constante dans $\mathbb{N}$.

La sémantique de TCTL est définie sur un système de transitions temporisées $T = (S, s_0, E, L)$, sur un alphabet Σ et l'ensemble de propositions $Prop$. Considérons s une configuration de T et une position i d'une exécution ρ_s^j issue de s. Nous notons $\rho_s^j(i)$ la configuration atteinte par l'exécution ρ_s^j à la position i. La sémantique de TCTL est analogue à celle de CTL avec les règles supplémentaires suivantes permettant la spécification de délai explicite :

- $s \vDash E\varphi U_{\bowtie c} \psi$ s'il existe une exécution ρ_s^j issue de s tel que $\rho_s^j \vDash \varphi U_{\bowtie c} \psi$;
- $s \vDash A\varphi U_{\bowtie c} \psi$ si pour toute exécution ρ_s^j issue de s, $\rho_s^j \vDash \varphi U_{\bowtie c} \psi$.

Avec :

$\rho_s^j \vDash \varphi U_{\bowtie c} \psi$ s'il existe une position $i \geq 0 \; de \; \rho_s^j$ telle que $\text{duree}(\rho_s^j(i)) \bowtie c$, $s_i^j \vDash \psi$ et pour toute position $k <_{\rho_s^j} i, s_k^j \vDash \varphi$.

Remarque : On dit qu'un automate temporisé G_t satisfait une formule φ de TCTL (i.e. $G_t \vDash \varphi$) si la configuration initiale $s_0 = (q_0, 0)$ du système de transition associé satisfait φ.

Pour illustrer l'expressivité de TCTL, considérons l'exemple suivant : « Toute faute critique dans le système provoque l'arrêt du système dans un délai d'au plus 10 u.t. ». Cette spécification se traduit par l'équation (IV-17).

$$AG(faute_{critique} \Longrightarrow AF_{\leq 10} arrêt) \qquad \textbf{(IV-17)}$$

Il existe d'autres extensions de CTL comme TCTL_h qui manipulent explicitement des horloges dans les expressions.

Dans ce travail, nous choisissons d'utiliser la logique temporisée TCTL car nos modèles sont à base d'automates temporisés et qu'à partir d'un état plusieurs exécutions sont possibles. D'autre part, nous avons besoin de vérifier des propriétés comme le fait qu'un automate temporisé soit non-zénon. Cette propriété s'exprime en TCTL par la formule donnée par l'équation (IV-18)(Henzinger, et al., 1994) (Yovine, 1997).

$$init \Longrightarrow AGEF_{=1} true \qquad \textbf{(IV-18)}$$

Avec $init$ une propriété caractérisant l'état initial.

Le choix de TCTL est également justifié par l'existence de nombreux model-checkers libre de distribution et basés sur cette logique temporisée. Dans ce travail nous avons notamment utilisé les model-checkers KRONOS et UPPAAL.

IV.V.3 Présentation de KRONOS

KRONOS est un model-checker qui a été conçu pour la conception des systèmes temps réel (Yovine, 1997). Depuis une vingtaine d'années qu'il existe, il a été utilisé dans différentes applications comme la vérification des protocoles des réseaux informatiques (Daws, et al., 1994) ou dans la vérification des circuits asynchrones temporisés (Maler, et al., 1996). Le système à vérifier est modélisé sous la forme d'un automate temporisé ou d'un ensemble d'automates temporisés pouvant être synchronisés par des signaux communs aux composants du système. KRONOS permet de faire de la vérification logique en spécifiant les formules à vérifier en TCTL. Il permet également de faire de la vérification comportementale en modélisant la spécification à vérifier sous la forme d'un automate temporisé. La vérification comportementale consiste à faire une abstraction du comportement temporel des deux automates de manière à vérifier si leurs comportements sont équivalents par bisimulation.

Dans ce travail, nous avons plutôt utilisé la vérification logique.

Les modèles sont directement construits par édition de fichiers textes. Considérons par exemple le modèle de la Figure IV-1. Sa traduction dans la syntaxe de KRONOS est donnée par la Figure IV-25.

La description textuelle commence par la déclaration du nombre de nœuds du modèle appelés états (#states 6), et du nombre de transitions (#trans 7) du modèle. Il faut également déclarer toutes les horloges utilisées dans le modèle (#clock x). Si le modèle doit être synchronisé, on doit déclarer les signaux de synchronisation (#sync a b). Nous reviendrons ultérieurement sur cet aspect.

Ensuite chaque état est décrit en spécifiant son numéro à l'aide de l'instruction $state : i$ avec $i \in [0..n-1]$, n étant le nombre d'états du modèle. L'instruction $invar : x \leq 3$ permet de spécifier que $x \leq 3$ est l'invariant associé à l'état considéré. Les transitions sortantes sont définies par l'instruction $trans$. Chaque transition est déclarée selon la syntaxe $< garde > \Longrightarrow < signal > ; reset(< h_1 >, \dots, < h_j >) ; goto\ i$ avec les h_j les horloges à réinitialiser et i l'état de destination de la transition. Par exemple, la transition $x > 3 \Rightarrow f; ; goto\ 2$ de la Figure IV-26 permet de spécifier la transition de l'état 1 vers l'état 2.

Une des particularités de KRONOS est la possibilité d'associer une liste de propositions à chaque état par l'instruction $prop$. Par exemple, l'état initial possède par défaut la proposition $init$. Nous pouvons voir sur la Figure IV-26, que nous avons associé la

proposition $faulty$ à l'état 2. Ce type de propositions est très intéressant pour vérifier des propriétés d'accessibilité ou de vivacité d'un modèle.

```
/*Declarations */

#states 6 /*Nombre d'états */
#trans 7  /*Nombre de transitions*/
#clocks x /* Déclaration des horloges du modèle */
#sync a b  /* Déclaration des signaux de synchronisation */

/*Description des etats*/

state: 0 /* Numérotation des états */
prop: init
invar: true
trans:
true =>a; reset{x}; goto 1

state: 1
prop: attend
invar: x <= 6
trans:/* Liste des transitions */
x>3 => f; ; goto 2
x<=3 => u; ; goto 4

state: 2
prop: faulty /* Propriétés associées à l'état */
invar: x <= 6
trans:
true => b; ; goto 3

state: 3
prop: detect
invar: true
trans:

state: 4
prop: nonfaulty
invar: x <= 3 /* Invariant de l'état */
trans:
true => b; ; goto 5

state: 5
prop: none
invar: true
trans:
```

Figure IV-25 – Codage en KRONOS de l'automate temporisé de la Figure IV-1

KRONOS offre deux types d'analyse : l'analyse avant et l'analyse arrière. L'analyse avant consiste à vérifier une propriété en parcourant le graphe à partir de l'état de départ. Cette analyse peut être effectuée par exploration en largeur ou en profondeur d'abord. Pour une vérification d'une propriété qui doit être vérifiée quelque soit l'exécution, il est préférable de faire une recherche exhaustive en largeur. Lorsqu'il s'agit de vérifier qu'il existe un état d'une

exécution où la proposition est vérifiée, il est judicieux pour des questions d'efficacité de privilégier une recherche en profondeur d'abord.

Pour effectuer une analyse avant il faut exécuter la commande donnée par l'équation (IV-19).

$$kronos\ -forw\ gt.tg\ prop.tctl \qquad \textbf{(IV-19)}$$

avec $gt.tg$ le fichier contenant le modèle de l'automate temporisé et $prop.tctl$ le fichier contenant la formule TCTL à vérifier. Pour faire une analyse en profondeur plutôt qu'en largeur, il faut utiliser l'option $-DFS$.

Dans tous les cas KRONOS indique par $true$ ou $false$ si la vérification aboutit ou non. En cas d'échec, il génère le fichier $\_path.reach$ qui donne un contre-exemple sous la forme d'une séquence de transitions menant à un état ne vérifiant pas la proposition.

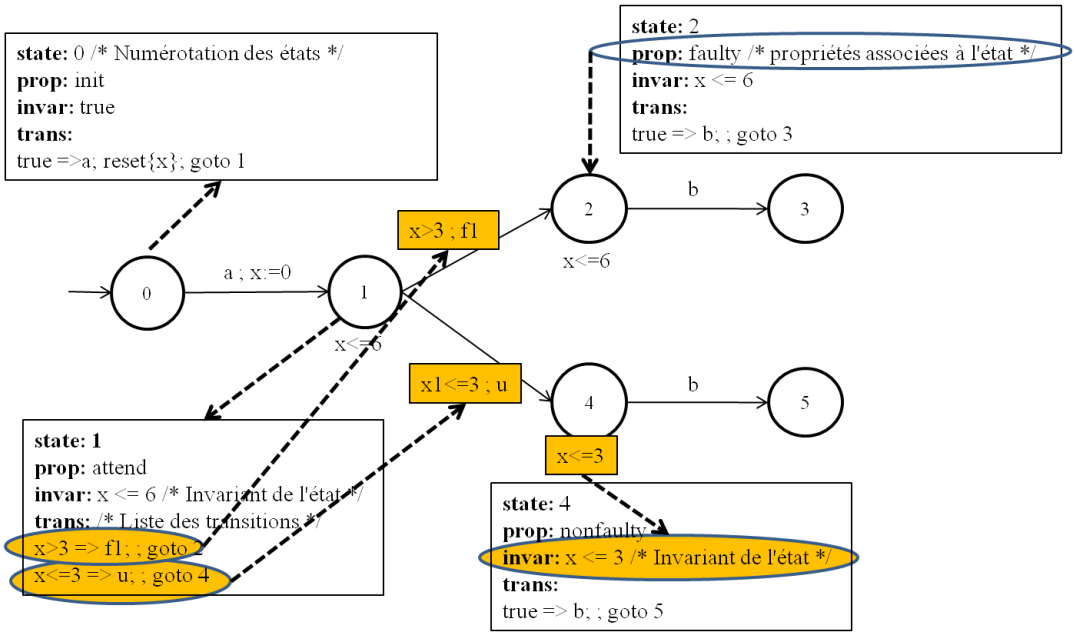


Figure IV-26 – Transcription du modèle graphique en description textuelle.

L'analyse arrière est effectuée par la commande donnée par l'équation (IV-20).

$$kronos\ -back\ gt.tg\ prop.tctl \qquad \textbf{(IV-20)}$$

L'analyse arrière est le mécanisme de vérification par défaut. KRONOS génère le résultat dans le fichier $prop.eval$. En cas de succès, ce fichier résultat contient un ensemble d'états de l'espace d'états vérifiant la formule TCTL de $prop.tctl$.

Pour vérifier qu'un état d'un modèle est accessible à partir de l'état initial, il suffit de saisir $state(i)$ comme proposition à vérifier. Si on veut vérifier qu'il n'existe pas d'état bloquant dans un modèle, il suffit d'associer une proposition de type fin à tous les états finaux du modèle est de vérifier que quelque soit l'exécution, l'état final est atteint. Il suffit donc de vérifier la formule donnée par l'équation (IV-21).

$$init \Longrightarrow \mathrm{A}F\ fin \qquad \textbf{(IV-21)}$$

Prenons l'exemple de la Figure IV-6. Nous cherchons à évaluer s'il existe des exécutions fautives qui arrivent dans l'état $q_{3,}^{1}q_{5}^{2}$. Il suffit de mettre fin comme proposition de cet état est de vérifier la formule donnée par l'équation (IV-22).

$$init \Longrightarrow EFfin \qquad \textbf{(IV-22)}$$

En utilisant l'accessibilité, on peut également vérifier des propriétés de vivacité. Par exemple, si on veut vérifier que le système n'évolue jamais vers des états non sûrs, il suffit par exemple de mettre la proposition $safe$ dans tous les états qu'on estime sûr de fonctionnement et de vérifier la formule (IV-23).

$$init \Longrightarrow \mathrm{AG}\ \mathrm{safe} \qquad \textbf{(IV-23)}$$

D'autre part, KRONOS permet de mettre en œuvre la modélisation d'un système de manière modulaire. En effet, chaque module du système peut être modélisé indépendamment par un automate temporisé. Pour vérifier les propriétés du système, il y a deux possibilités : faire un produit synchronisé des différents modèles des modules pour créer le modèle du système ou faire une vérification à la volée sans création d'un modèle explicite du système. Pour le produit synchronisé, il suffit d'exécuter la commande (IV-24).

$$kronos\ -out\ S.tg\ \ m_1.tg\ m_2.tg.\ldots\ m_n.tg \qquad \textbf{(IV-24)}$$

avec $S.tg$ le modèle du système produit par la commande et $m_i.tg$ les modèles des composants.

KRONOS possède de nombreuses fonctionnalités qui permettent de réduire la taille de l'espace exploré pour vérifier une propriété et d'optimiser les temps de traitement (Yovine, 1997). Ce sont les principales raisons et notamment l'approche modulaire qui nous ont poussées à choisir cet outil.

IV.V.4 Présentation de UPPAAL

L'outil UPPAAL a été développé en 1995, conjointement par l'université d'UPPsala (UPP) en Suède et l'université d'AALborg (AAL) au Danemark, d'où son nom (Larsen, et al., 1997).

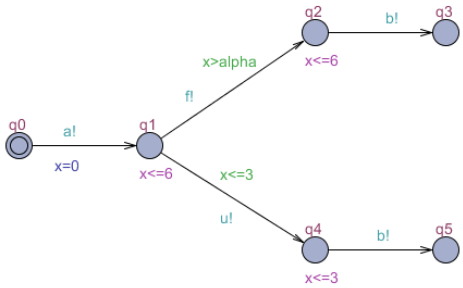


Figure IV-27 – Modélisation avec UPPAAL de l'automate temporisé de la Figure IV-1

Il s'agit d'un outil intégré pour la modélisation, la simulation et la vérification des systèmes temps réel modélisés par des automates temporisés avec variables entières bornées, actions urgentes, etc... Les propriétés qui peuvent être vérifiées sont principalement des propriétés d'accessibilité, de vivacité ou d'états bloquants. La logique utilisée par UPPAAL est uniquement un fragment de TCTL qui ne permet pas de vérifier l'ensemble des formules TCTL, les opérateurs de temps ne pouvant pas être imbriqués. L'algorithme implémenté dans UPPAAL est essentiellement un algorithme d'analyse en avant. On peut obtenir des informations supplémentaires sur le site web d'UPPAAL accessible par le lien http://www.UPPAAL.com.

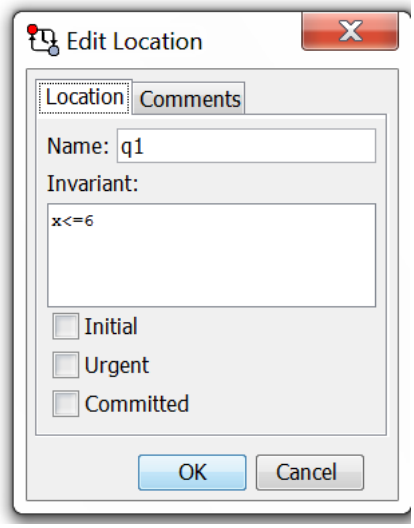


Figure IV-28 – Construction d'un état sous UPPAAL

Les avantages d'UPPAAL par rapport à KRONOS sont d'une part son interface graphique, très conviviale et d'autre part, son module de simulation qui permet, lors de la phase de modélisation de faire des tests du modèle et de détecter d'éventuelles erreurs de modélisation. En effet, une des caractéristiques de UPPAAL est qu'il basé sur une communication asynchrone par canaux à la différence de KRONOS qui est basé sur une communication synchrone. Ce type de communication correspond à la réalité. D'autre part, un composant ne peut pas recevoir un signal s'il n'a pas été émis par un autre composant. Cela oblige à respecter l'évolution logique entre les différents états d'un signal.

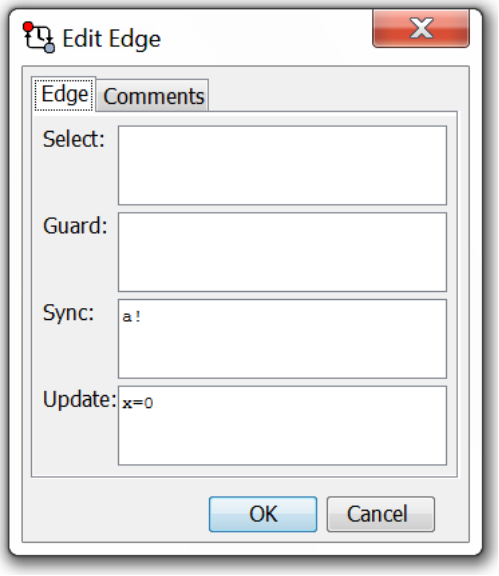


Figure IV-29 – Construction d'une transition sous UPPAAL

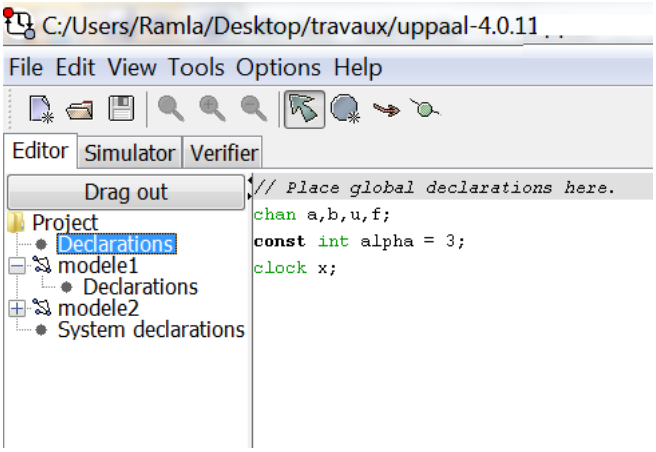


Figure IV-30 – Déclaration sous UPPAAL des horloges et des variables de l'automate de la Figure IV-27

Dans l'éditeur d'UPPAAL (menu 'Editor'), les états sont modélisés par des cercles. L'état initial est représenté par un double cercle ($q0$ dans la Figure IV-27). Si on double clique sur un cercle, une fenêtre s'ouvre (Figure IV-28) pour indiquer le nom de l'état et donner son invariant s'il y en a pour cet état. Par exemple, l'état $q1$ de la Figure IV-27 a un invariant : $x \leq 6$. On met dans le champ 'Invariant' l'expression $x <= 6$. Les transitions sont modélisées par des flèches orientées. Une action est modélisée par un canal. En effet, sous UPPAAL, un modèle consiste en un ensemble d'automates temporisés qui communiquent par une synchronisation binaire utilisant des canaux et une syntaxe de type émission/réception. Ainsi, sur un canal 'c', un émetteur envoie le signal $c!$ et un récepteur se synchronise avec lui par le signal complémentaire $c?$. Les horloges sont des variables qui évoluent d'une manière synchrone avec le temps. Prenons l'exemple de la transition entre les états $q0$ et $q1$. Un double clic sur la flèche reliant les deux états ouvre une fenêtre (Figure IV-29). Dans le champ 'Guard', on met la garde de la transition s'il y en a. Dans le champ 'Sync', on met

l'action synchronisée. Et dans le champ 'Update', on met les horloges à initialiser et éventuellement les variables à mettre à jour. Les variables et les horloges sont déclarés dans une zone 'Declaration' (Figure IV-30).

Après avoir conçu les modèles, nous passons à la simulation. Nous ouvrons le menu 'Simulator'. Si le message d'erreur de la Figure IV-31 apparait, c'est qu'il existe des erreurs dans les modèles. Il faut alors corriger ces erreurs. Sinon, nous pouvons faire la simulation de nos modèles et nous obtenons la fenêtre de la Figure IV-32. Le simulateur peut être utilisé de trois façons:

- Nous pouvons compiler le système manuellement et choisir chaque fois les transitions à franchir. Le simulateur, dans la partie de contrôle 'Enabled Transitions', nous propose une liste de transitions qui peuvent être franchies.
- Nous pouvons basculer au mode de lecture aléatoire pour que le système fonctionne en mode automatique (Appui sur le bouton 'Auto' dans la partie de contrôle)
- Nous pouvons passer par une trace (enregistrée ou importée du vérificateur) pour voir comment certains états sont accessibles (Appui sur le bouton 'Open' dans la partie de contrôle).

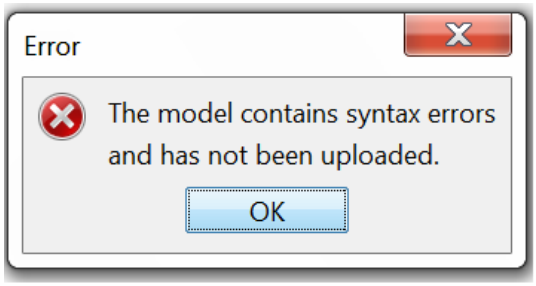


Figure IV-31 – Message d'erreur en cas d'erreur de conception sous UPPAAL

Le simulateur permet d'enregistrer la trace d'une simulation, les différents échanges entres modèles et les valeurs des horloges et des variables.

Le vérificateur (menu 'Verifier') permet de vérifier les propriétés sous forme TCTL. Nous pouvons vérifier une ou plusieurs propriétés à la fois. Nous pouvons insérer de nouvelles propriétés ou supprimer des propriétés. Nous pouvons aussi basculer l'affichage pour voir les propriétés ou leurs commentaires respectifs dans la liste. Quand une propriété est sélectionnée, nous pouvons modifier sa définition ou ajouter des commentaires pour documenter ce que signifie cette propriété informellement. Le panneau d'état, au bas de la vue, montre la communication avec le serveur (Figure IV-33).

Remarque : Sous Uppaal, le connecteur temporel G est représenté par [] et le connecteur temporel F est représenté par <>.

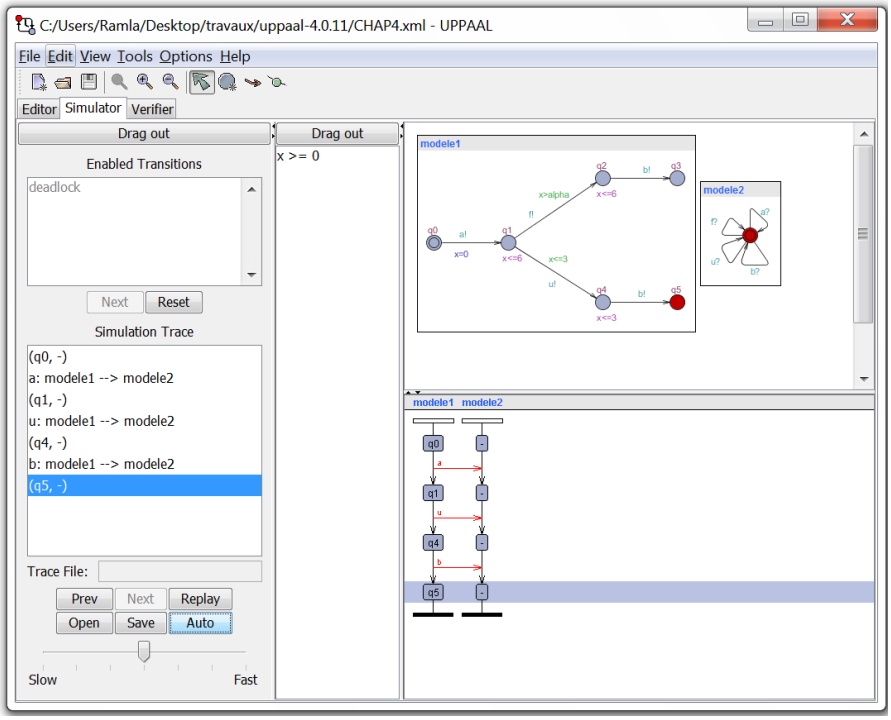


Figure IV-32 – Simulation de l'exemple de la Figure IV-27

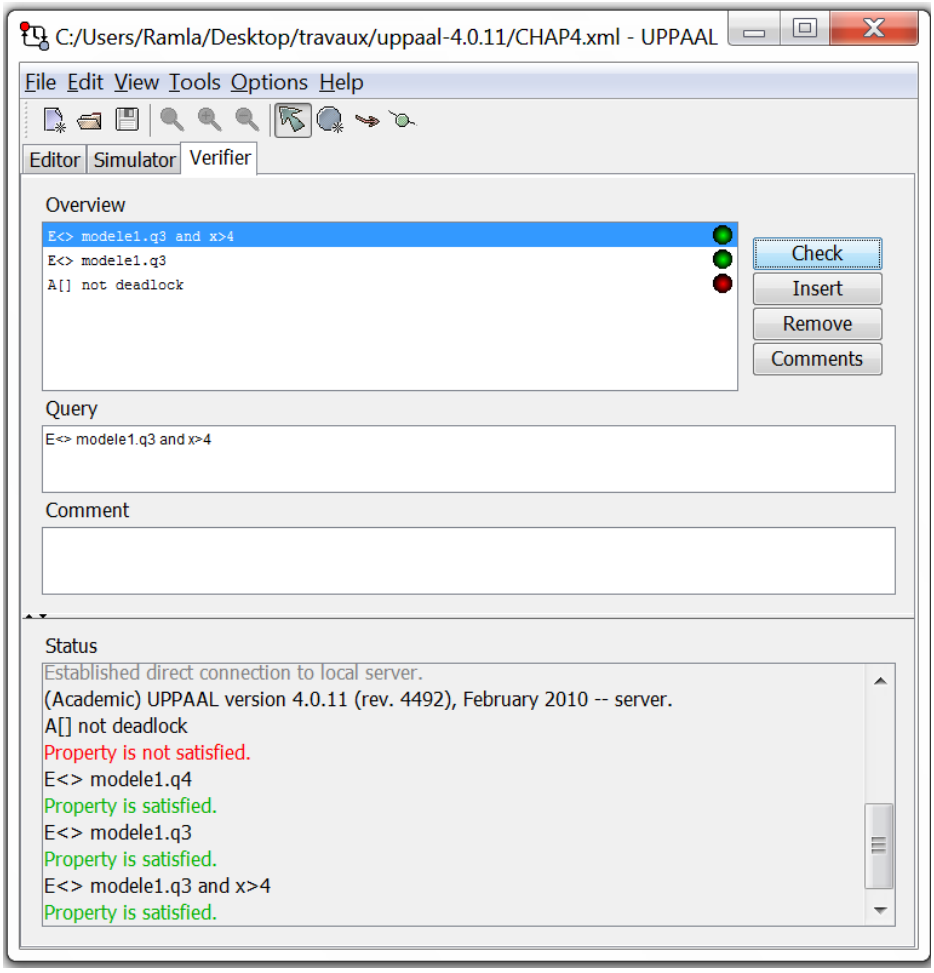


Figure IV-33 – Vérification avec UPPAAL

Pour vérifier qu'il n'existe pas d'état bloquant dans un modèle, il suffit de vérifier la formule donnée par l'équation (IV-25).

$$A\ []not\ deadlock \quad \textbf{(IV-25)}$$

Pour vérifier qu'un état est accessible, il suffit de vérifier la formule donnée par l'équation (IV-26). Avec $modele1.q3$ le nom de l'état dont on souhaite vérifier l'accessibilité.

$$E <> modele1.q3 \quad \textbf{(IV-26)}$$

Nous pouvons trouver des exemples détaillés et la documentation complète de UPPAAL dans le tutorial (Behrmann, et al., 2004).

Une autre caractéristique d'UPPAAL est qu'il permet d'écrire des gardes en utilisant des variables autres que des horloges. Cet outil nous permet donc de mettre en œuvre le concept d'automates à transitions gardées que nous définirons au chapitre V

IV.V.5 Principe d'application du model-checking au test de la diagnosticabilité

Dans cette partie nous allons expliquer comment nous allons utiliser le model-checking pour tester la diagnosticabilité d'un SEL. Comme nous l'avons présenté au §IV.I.4, le test de la diagnosticabilité se fait soit en montrant que toutes les exécutions fautives de $(G_t^1||_s G_t^2)^{-f2}$ sont zénon, soit en faisant le test du vide d'un automate de Büchi. Dans les deux cas, nous devons être capables de caractériser les exécutions fautives d'un automate. Comme la principale propriété vérifiable par les outils de model-checking est l'accessibilité, nous devons ramener la caractérisation de ces exécutions à une rechercher d'accessibilité. Pour cela, étant donné un automate G_t et f un des ses événements caractérisant une faute du système modélisé, il suffit de réaliser le produit synchrone de G_t par l'automate A_f de la Figure IV-34.

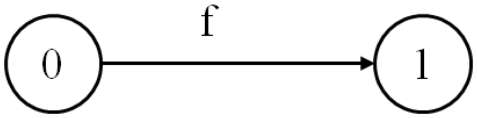
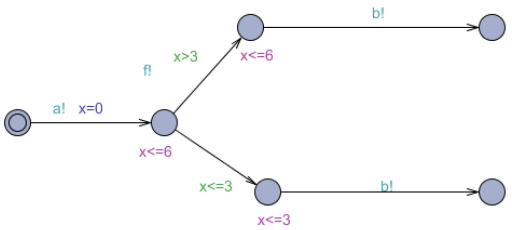


Figure IV-34 – Automate A_f

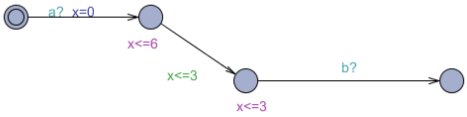
La synchronisation des deux automates est réalisée uniquement sur l'événement commun f. En utilisant KRONOS, cela revient par exemple à associer à l'état 1 de A_f la propriété « prop : faute ». Du coup la recherche des exécutions fautives se ramène à chercher toutes les exécutions qui conduisent à un état vérifiant la propriété $faute$. Aussi les exécutions fautives zénons sont données par l'équation par (IV-27).

$$faute \Longrightarrow not\ (AG\ EF_{=1} true) \qquad \textbf{(IV-27)}$$

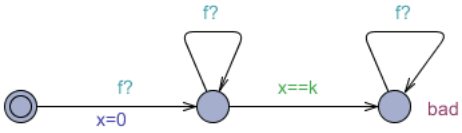
Pour la vérification du vide du test d'un automate de Büchi, nous utilisons le même principe. Il suffit d'affecter la propriété $buchi0$ à l'état 0 de l'automate $Div(y)$ présenté à la Figure IV-7. L'automate D et alors construit par des produits synchrones comme présentés au §IV.I.3. Le test du vide de D peut être effectué avec Open-Kronos (Tripakis, 2009), une extension de Kronos. Sinon, avec Kronos. Il faut montrer que les exécutions fautives de D sont zénons.



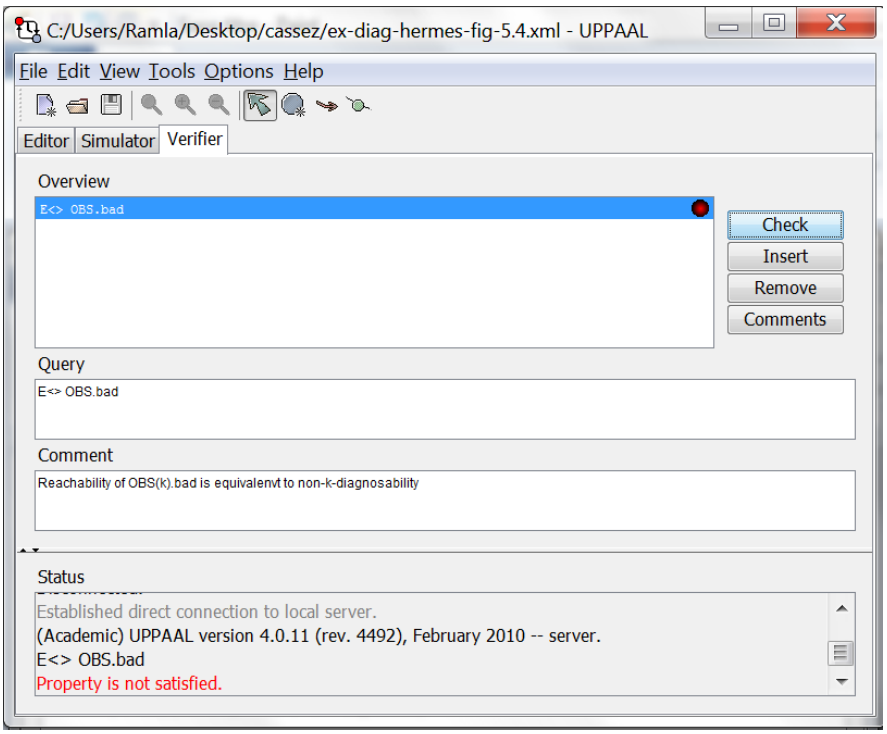
a)



b)



c)



d)

Figure IV-35 – Model-cheking utilisant UPPAA pour le test de diagnosticabilité

On peut également utiliser UPPAAL pour vérifier la diagnosticabilité. Pour cela, le principe consiste à verifier le caractère zénon des éxécutions fautives des deux copies à l'aide d'un automate observateur. L'utilisation d'UPPAAL pour le test de diagnosticabilité du modèle de la Figure IV-4 en se basant sur la méthode de Cassez (§III.I.3) est représenté par la Figure IV-35. La Figure IV-35a représente la copie du modèle avec la faute. La Figure IV-35b représente le modèle sans la faute f. Le test de diagnosticabilité revient à vérifier la non atteignabilité de l'état « bad » de l'automate temporisé nommé « OBS » du modèle de la figure Figure IV-35c. Cet automate est un observateur de la faute f, dont la transition vers l'état BAD s'effectue au bout d'un temps générique de k ut après la première occurrence de f. Si les éxécutions fautives du produit des deux premiers automates temporisées sont zénon pour un k donné, alors les hotloges sont bloquées et ont peut pas atteindre l'état BAD. Ainsi, pour k=2, le système n'est pas diagnosticable. A partir de k=3, le système est diagnosticable car l'état BAD n'est pas atteignable dans ce cas (cf. le message « property is not satisfied » dans Figure IV-35c). Ici, nous avons vérifié la propriété ($E <> OBS.bad$ 5 Figure IV-35d).

Ce principe sera repris et illustré au chapitre V en l'appliquant à un cas d'étude.

Conclusion du chapitre IV

Dans ce chapitre, nous avons présenté une étude sur le diagnostic des systèmes temporisés. L'objectif de cette étude était double : d'une part étudier les particularités du cas des automates temporisés et d'autre part identifier les moyens pour mettre en œuvre le test de la diagnosticabilité et la synthèse des diagnostiqueurs.

Dans un premier temps, nous avons souligné l'expressivité du formalisme des automates temporisés comparé aux automates à états finis. Cette plus forte expressivité permet de gagner en concision au niveau modélisation et donc participe à la réduction de la combinatoire. C'est la raison pour laquelle nous faisons le choix d'utilisation de ce formalisme par rapport aux automates à états finis. Mais la majeure partie des travaux sur la diagnosticabilité concernent les automates à états finis comme nous avons pu le montrer au chapitre III. Nous avons donc étudié la diagnosticabilité des systèmes temporisés modélisés par des automates temporisés. Cela nous a permis de montrer que dans le cas des automates temporisés la propriété importante à vérifier est le caractère zénon ou pas d'un modèle. Dans ce cadre, nous avons montré qu'il existe deux manières de faire de tester la diagnosticabilité d'un automate temporisé : soit en démontrant que toutes les transitions fautives de $(G_t^1 ||_s G_t^2)^{-f2}$ sont zénons, soit par le test du vide d'un automate de Büchi.

Nous nous sommes ensuite intéressés à la synthèse de diagnostiqueurs pour les automates temporisés. En effet, il est établi que tous les automates temporisés ne sont pas déterminisables contrairement aux automates à états finis et qu'en plus les automates temporisés non déterministes ont une expressivité supérieure à celle des automates temporisés déterministes. Nous avons donc montré qu'il existe des classes d'automates temporisés non déterministes qui peuvent être déterminisés et donc conduire à la synthèse de diagnostiqueur

par déterminisation. Dans le chapitre V, afin de simplifier la présentation de notre approche nous utiliserons des automates temporisés déterminisables.

Dans la deuxième partie de ce chapitre, nous avons cherché à proposer une mise en œuvre du test de la diagnosticabilité et de la synthèse de diagnostiqueur par approche par outils de model-checking. Cela nous a mené à présenter la logique temporelle TCTL et à donner une brève présentation de deux model-checkers, KRONOS et UPPAAL. Pour la suite, nous ne faisons pas le choix de l'un de ces deux outils au détriment de l'autre. Nous estimons qu'ils sont complémentaires. En effet les capacités de vérification de KRONOS sont plus importantes que celle d'UPPAAL. KRONOS permet également de faire des produits synchrones de modèles, nous donnons la possibilité de construire simplement le modèle d'un système à partir des modèles de ces composants. Pour combattre l'explosion combinatoire, KRONOS permet également de faire de la vérification à la volée. Mais cet outil est basé sur une communication synchrone qui ne reflète pas la réalité de fonctionnement des systèmes asynchrones. A contrario UPPAAL est plus convivial et est basé sur un mécanisme de communication asynchrone. Ces deux caractéristiques permettent de détecter facilement des erreurs de modélisation. UPPAAL permet également de modéliser des gardes basées sur des variables autres que des horloges. Cette capacité nous permettra de mettre en œuvre le concept de transition gardée événementielle que nous introduirons dans le chapitre suivant.

V Approche diagnostiqueur pour la surveillance des SEL

Introduction du chapitre V

Dans ce chapitre, nous proposons une nouvelle approche de modélisation des systèmes embarqués logiques pour le diagnostic de leurs fautes. Elle est basée sur une décomposition structurelle du système et sur une extension de la diagnosticabilité modulaire au contexte des systèmes temporisés.

En effet, l'approche proposée part d'une décomposition du système en composants. Ensuite, nous étudions chaque composant pour construire un modèle de son comportement. La construction du modèle de comportement d'un composant est basée sur une double abstraction : une abstraction du comportement séquentiel et une abstraction du comportement temporel. La disponibilité d'un modèle comportemental du système est une condition nécessaire pour l'application de la technique du diagnostiqueur que nous proposons d'utiliser ici. Mais les systèmes auxquels nous nous intéressons étant caractérisés par de nombreuses entrées/sorties, la construction d'un modèle du système peut se heurter à un problème d'explosion combinatoire. Aussi, afin de faire face à cette contrainte, nous proposons une nouvelle approche de diagnostic distribué qui prend en compte la décision de diagnostic du diagnostiqueur de chaque composant du système surveillé afin de caractériser l'état du système.

Tout au long de ce chapitre, nous allons illustrer nos propositions sur un exemple de système embarqué logique, le cas d'étude que nous avons traité dans le cadre du projet FerroCOTS. Nous parlerons donc de la carte FerroCOTS.

V.I Les exigences de surveillance

D'un point de vue fonctionnel, la surveillance des systèmes auxquels nous nous intéressons a d'abord pour priorité de garantir le bon fonctionnement d'un système en exploitation. Les exemples typiques qui nous intéressent sont les cartes de contrôle-commande dans les systèmes de transport dans le domaine aéronautique, ferroviaire ou automobile. Prenons l'exemple d'une carte qui contrôlerait le train d'atterrissage d'un avion. Il est évident que les sorties de cette carte seraient critiques dans le sens où il n'est pas envisageable de commander l'entrée du train d'atterrissage alors que l'avion n'a pas décollé ou inversement de commander sa sortie alors que l'avion est en vitesse de croisière à 10000 mètres d'altitude. Le premier objectif du système de surveillance est donc de déterminer si une sortie a la bonne valeur compte tenu de l'état du système, et si elle garde cette valeur conformément à cet état. Dans ce cadre, on doit surveiller si :

- O1 : une sortie critique doit changer de valeur et reste collée à sa valeur courante,
- O2 : une sortie critique doit conserver sa valeur courante et change intempestivement de valeur.

Le deuxième objectif de la surveillance est de garantir le bon fonctionnement du système surveillé à sa mise en exploitation. En effet, les systèmes complexes sont souvent conçus en prenant en compte les taux de défaillances conditionnel des composants pour en déduire leurs probabilités de pannes. Par exemple, dans le cadre de la commande d'un système critique, on peut mettre en place des équipements redondants permettant de garantir le bon fonctionnement du système commandé, même en cas de défaillance de l'un de ses composants. Par exemple, si on a deux cartes pour commander le train d'atterrissage avec un système de vote et que l'on sait qu'il n'est pas possible que les deux cartes tombent en pannent de manière concomitante dans un délai de 10h, on pourra envisager d'effectuer une mission de moins de 10h si jamais l'une des deux cartes tombaient en panne au début de la mission. Par contre, si les tests initiaux de bon fonctionnement nous indiquent que l'une des deux cartes est en panne, la mission ne peut plus être réalisée sans avoir réparé le système car la 2ème carte est susceptible de tomber en panne en cours de la mission. Donc, la surveillance devra également permettre de mettre en œuvre des BITE (Built In Tests) des systèmes logiques considérés.

Le troisième objectif de la surveillance est de réduire les durées d'indisponibilité des systèmes en facilitant leur maintenance. Cela passe notamment par l'identification rapide des composants en panne afin de les réparer ou de les remplacer. Cet objectif est moins prioritaire d'un point de vue sécurité mais tout aussi indispensable sur le plan économique.

V.II Approche composants

Pour modéliser un système, notre approche consiste à le décomposer en sous-systèmes. Ensuite nous décomposons chaque sous-système en des sous système plus petits jusqu'à ce qu'on arrive au plus petit élément de décomposition que nous appelons un composant élémentaire. Après, nous modélisons le composant élémentaire en une machine à états. Pour avoir le modèle global du système, classiquement, nous faisons le produit synchrone des modèles de ses éléments. Or, les systèmes complexes que nous considérons se composent d'un nombre très importants de composants ce qui nous conduit au problème classique de l'explosion combinatoire lors de la modélisation de ce type de systèmes.

Dans cette partie, nous présentons la méthode de décomposition d'un système en composants. Nous définissons ensuite le concept de composant logique (CL).

V.II.1 Décomposition d'un système en composants

Les systèmes complexes ne sont pas simple à modéliser vu le grand nombre d'éléments qui les composent et les relations entre ces éléments. Cependant, on peut représenter un système en sous-systèmes. Cette opération peut être renouvelée pour chaque sous-système jusqu'à

atteindre les composants élémentaires. Le plus bas niveau de décomposition d'un sous-système est le composant élémentaire.

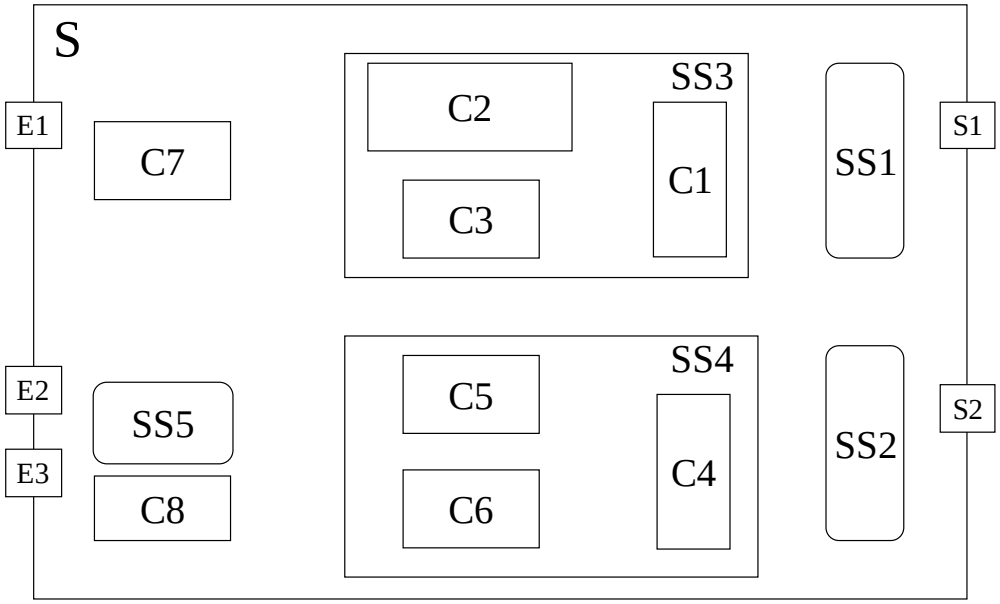


Figure V-1 – Décomposition d'un système en sous-systèmes et composants

La Figure V-1 montre un système décomposé en sous-systèmes et composants. Il comprend cinq sous-systèmes notés SS_i, $i \in [1..5]$. Il comprend également des composants nommés C7 et C8, ainsi que des ports d'entrées notés E_n, $n \in [1..3]$ et des ports de sorties nommés S_k, $k \in [1..2]$. Les sous-systèmes sont eux-mêmes décomposés en sous-systèmes et/ou composants. Par exemple, SS3 est décomposé en trois composants C1, C2 et C3. Afin de faciliter la désignation des composants nous utiliserons la notation SSi.Cj pour désigner le composant Cj du sous-système SSi.

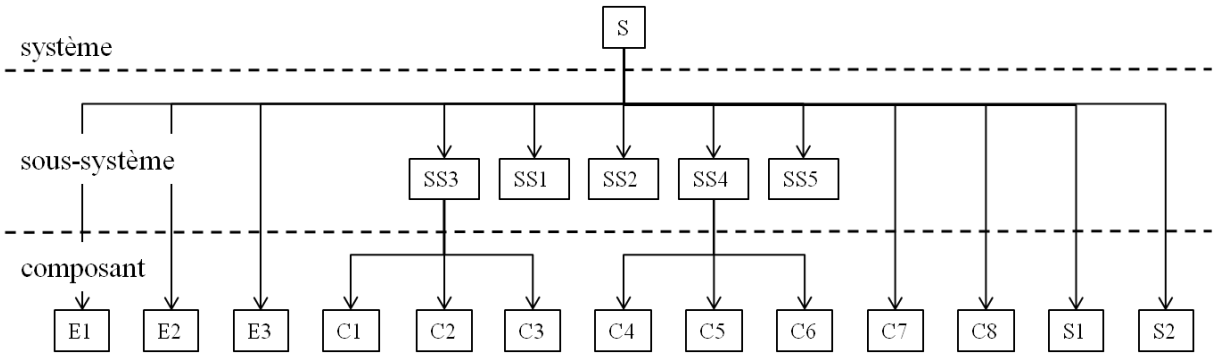


Figure V-2 – Arbre matériel

La décomposition est souvent liée au point de vue de l'expert qui effectue cette décomposition. On remarque qu'elle est souvent guidée par des points de vue fonctionnels ou opérationnels. Par exemple, dans le cas de la mise en œuvre d'une politique de maintenance d'un système, c'est souvent la manière de démonter le système qui guide l'expert. C'est l'approche de décomposition que nous privilégions dans ce travail. En effet, dans ce cadre, la

décomposition d'un système en constituants aboutit à la construction d'un arbre matériel représentant la hiérarchie des constituants du système (Figure V-2).

V.II.2 Le concept de composant logique

Dans ce travail, nous nous sommes plus particulièrement intéressés aux systèmes embarqués communiquant avec leur environnement à l'aide de nombreuses entrées/sorties logiques (systèmes embarqués logiques). Le cœur de ce type de système est souvent constitué par un composant numérique de type microcontrôleur ou FPGA (Figure V-3). Classiquement, le comportement de tels systèmes est représenté par des systèmes états/transitions. Le comportement caractérise la manière dont le système transforme ses entrées pour produire ses sorties.

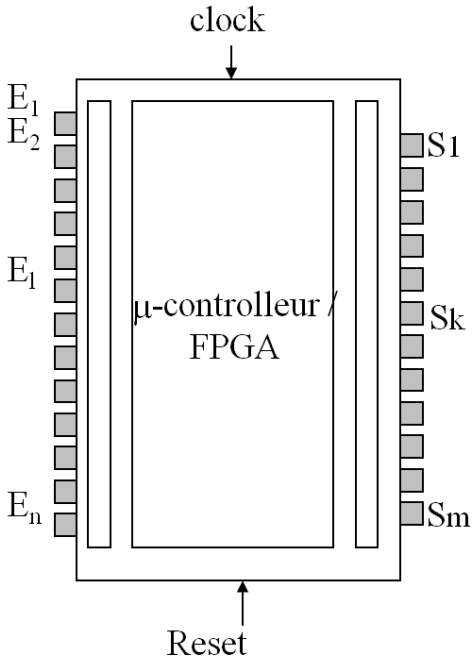


Figure V-3 – Schématisation d'un composant programmable

Mais la présence d'un nombre important d'entrées/sorties entraine un risque d'explosion combinatoire, notamment dans le cadre de formalismes comme les automates à états finis. Pour éviter cet inconvénient, nous proposons de décomposer chaque constituant en un ensemble de composants logiques.

Comme illustré par la Figure V-4, chaque constituant d'un système possède des entrées lui permettant de recevoir un flux issus d'autres constituants et des sorties lui permettant d'initier des flux vers d'autres constituants. Par exemple le constituant SS4 possède deux entrées et deux sorties. Il est en interaction avec SS5 et C8 qui lui envoient des flux d'entrée. Il traite ces flux en utilisant le comportement de ses composants internes (C4, C5 et C6) et génère des flux en sortie. Il émet un flux vers SS1 et un autre flux vers SS2.

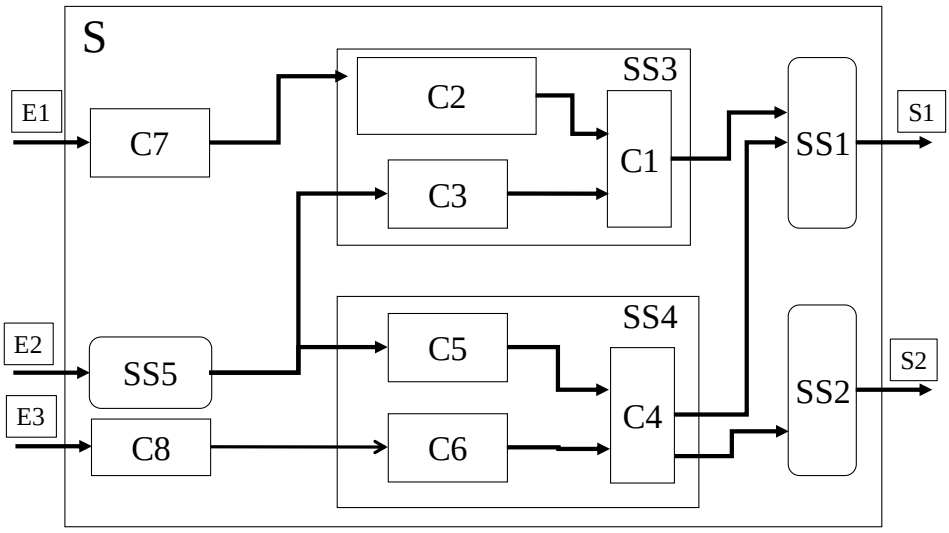


Figure V-4 – Flux entre les différents constituants d'un système logique

Etant donné une sortie de la carte, comme la sortie S2, nous désignons par ligne fonctionnelle l'ensemble des composants utilisés pour fournir le flux de sortie correspondant. Ainsi la ligne fonctionnelle de S2 comprend les constituants S2, SS2, SS4.C4, SS4.C5, SS4.C6, C8, SS5, E3 et E2. En l'absence de redondance (comportement de type OU), la panne d'un des ses constituants conduit à une rupture de flux.

Tout système logique peut être défini par un ensemble de fonctions logiques. Chaque fonction logique définit la relation entre une sortie du système et ses entrées logiques.

Soit II={0,1}, l'ensemble des valeurs logiques. Un système logique séquentiel de n entrées et m sorties peut être défini par une machine de Mealy. Dans ce cas, il est formellement défini par deux applications g (la fonction de succession) et f (la fonction de sortie) :

$g : II^n \otimes II^q \rightarrow II^q$

$(X(t), Z(t)) \rightarrow Z(t+\Delta t)=g(X(t), Z(t))$

et

$f : II^n \otimes II^q \rightarrow II^m$

$(X(t), Z(t)) \rightarrow Y(t)=f(X(t), Z(t))$

Avec X le vecteur d'entrées, Y le vecteur de sortie, Z le vecteur état interne et t le temps.

Les applications g et f peuvent être décomposées en fonctions logiques. Soit y_i, $i \in [1..m]$, une composante de Y. Elle peut être définie en fonction des entrées et de l'état interne du système par la fonction logique f_i :

$f_i : II^n \otimes II^q \rightarrow II$

$(X(t), Z(t)) \rightarrow y_i=f_i(X(t), Z(t))$

Si le système considéré est un système logique combinatoire, les sorties ne dépendent alors que des entrées du système. Dans ce cas, le fonctionnement du système ne dépend plus de temps, et il n'y a plus lieu de définir les notions d'état interne et de fonction succession. Ainsi, chaque composante de sortie est une fonction logique du vecteur d'entrées i.e. $y_i=f_i(X)$.

Nous supposons dans la suite de cette étude que tout constituant d'un système logique est lui-même un système logique. Dans ce cadre, nous donnons notre définition de la notion de composant logique de base.

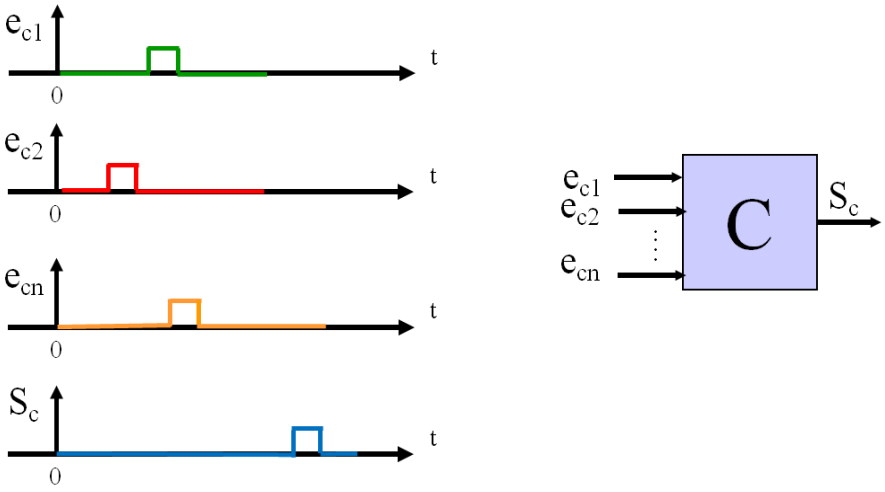


Figure V-5 – Schématisation d'un composant logique de base

Définition V-1 : Composant logique de base

Un composant logique de base est un composant caractérisé par une sortie logique et une ou plusieurs entrées logiques. Il est formellement défini par une fonction logique reliant sa sortie à ses entrées. S'il a un comportement séquentiel, il possède un état interne défini par une fonction de succession.

La Définition V-1 est cohérente avec les définitions habituelles de portes logiques (portes ET, OU et NOT) et de bascules (bascule SR, bascule JK …) qui sont considérées comme les briques élémentaires pour la synthèse des systèmes logiques. La nuance dans ce travail, est que pour nous, dès qu'un système logique possédera une unique sortie (Figure V-5), du point de vue de la surveillance, nous le considérerons comme un composant logique de base. Nous ne serons donc pas obligés de décomposer un système logique jusqu'au niveau porte logique ou bascules. Par exemple, dans la Figure V-4, en fonction des objectifs de maintenance (remplacement complet de la carte correspondante), le sous-système SS3 est assimilable à un composant logique de base caractérisé par deux entrées et une sortie. Cela est possible bien qu'il puisse être lui-même décomposé en composants. Par contre, SS4 n'est pas assimilable à un composant logique de base car il possède deux sorties logiques.

V.III Abstraction de comportement par réduction d'un graphe d'états

Le but de cette abstraction est de réduire la taille de l'observateur final qui sera utilisé pour construire le diagnostiqueur. Nous avons regardé des méthodes existantes dans la littérature que nous présentons dans la première section de cette partie. Ensuite, dans la deuxième section, nous proposerons le concept de graphe d'états à transitions événementielles gardées. Nous utiliserons cet outil dans la troisième section pour proposer une nouvelle méthode de réduction adaptée aux exigences du diagnostic des systèmes logiques.

V.III.1 Méthodes de réduction des systèmes logiques : état de l'art

Modéliser un système en une machine à états devient de plus en plus complexe et couteux dès que la taille de la machine à états est importante. Pour rendre plus simple et pour réduire la complexité du problème, nous avons recours aux méthodes de réduction d'états dans une machine à états. La réduction d'états consiste en la suppression des états qui sont équivalent dans une machine à états. Les trois méthodes principales pour la réduction d'états sont :

- la méthode de ligne correspondante « row matching »,
- la méthode d'implication graphique (d'incidence) « implication charts »,
- et la méthode de partitionnement successif « successive partitioning ».

La méthode de ligne correspondante est la plus simple de ces trois méthodes. Elle fonctionne bien quand on a des tables de transition d'état qui ont un état suivant évident et des équivalences entre les sorties. Cette méthode ne donne pas en général la machine à états la plus optimale d'un point de vue réduction. Néanmoins, comme elle est simple à appliquer et qu'elle donne souvent des résultats correctes, elle est généralement adoptée. La méthode d'implication graphique (d'incidence) emploie une grille graphique pour aider à trouver les équivalences et les implications entre les états. C'est une excellente approche systématique de réduction des machines à états. La méthode du partitionnement successifs est quasiment une méthode hybride entre les deux autres méthodes. Elle utilise la table graphique et l'équivalence d'incidence.

Ces trois méthodes sont détaillées en annexe A. Elles n'aboutissent pas forcément au même modèle réduit. Pour chaque méthode, la qualité de la réduction dépend de la structure du modèle initial. Ces méthodes ont été conçues pour la réduction de nombre de ports logiques nécessaires pour la mise en œuvre d'un système logique. Elles ne sont pas adaptées à notre méthode de diagnostic dans la mesure où l'objectif de réduction n'est pas le même. Nous allons donc proposer dans la suite de ce chapitre une méthode conforme à nos objectifs.

V.III.2 Nouvelle méthode de réduction pour le diagnostic des SEL

V.III.2.A Modélisation du comportement d'un composant logique

La difficulté pour la modélisation de ce type de fonctionnalité réside dans le fait qu'elle dépend du code implémenté dans le composant programmable. Nous chercherons toutefois à proposer une méthode de modélisation qui soit assez générique et qui permette de construire rapidement le modèle sous-jacent en fonction du code implémenté. Nous supposons que le comportement de la fonction peut être abstrait sous la forme d'un modèle états/transitions de type automate. Afin de rendre compréhensible la méthode que nous proposons, nous supposerons que nous avons un composant implémentant le comportement séquentiel modélisé par la machine de Mealy donnée par la Figure V-6. L'intérêt de cet exemple, est qu'il montre que la valeur de la sortie ne dépend pas seulement du vecteur d'entrées mais également de l'état interne du système (conforme au comportement séquentiel du système). Par exemple, on voit que lorsque nous sommes dans l'état 2, le vecteur d'entrées $\overrightarrow{e_i} = (1|1)^T$ donne la valeur 1 en sortie. Par contre le même vecteur d'entrées nous donne 0 lors d'une transition à partir de l'état 5.

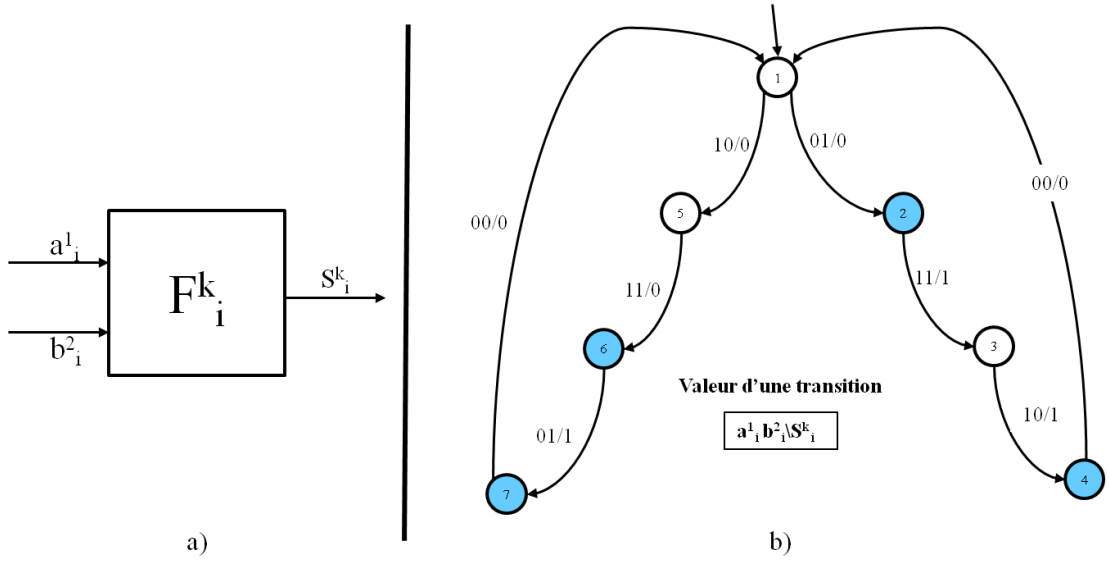


Figure V-6 – Exemple de machine de Mealy (b) pour un composant logique à deux entrées (a)

Cet exemple à deux entrées nous donne un comportement comprenant sept états. Notre objectif est de réduire autant que possible ce modèle de base afin de combattre l'explosion combinatoire qu'induirait l'utilisation des modèles comportementaux bruts. Pour cela, nous allons introduire la notion de graphe d'états à transitions gardées évènementielles.

V.III.2.B Graphe d'états à transitions événementielles gardées

Pour réduire le comportement d'un composant logique, nous proposons la notion de graphe d'états à transitions événementielles gardées.

Dans les modèles de représentation des systèmes logiques séquentiels classiques (Machine de Mealy ou machine de Moore), les états modélisent les différentes valeurs de la sortie d'un composant suite à un changement du vecteur d'entrées et les transitions entre ces états modélisent le changement du vecteur d'entrées permettant le passage d'un état à un autre. Nous transformons une machine de Mealy en un graphe d'états à transitions événementielles gardées. Nous rappelons que l'objectif du diagnostic est de surveiller les évolutions du système suite aux changements des valeurs d'entrée et de vérifier que suite à un changement des valeurs d'entrée, le composant passe à l'état prévu. Dans le cas contraire, le composant est défaillant.

Deux raisons nous amènent à considérer ce type de graphe d'états. La première raison est d'utiliser une méthode proche des méthodes traditionnelles appliquées aux machines à états comme la machine de Mealy. A titre d'exemple, nous rappelons que la méthode de ligne correspondante consiste à regrouper, les états atteints par des séquences de vecteurs d'entrées différentes, et qui ont le même état suivant et la même valeur de sortie. En effet, considérer la séquence de vecteurs d'entrées signifie qu'il est nécessaire d'examiner l'évolution de chaque entrée du composant. En effet, il est traditionnellement admis dans le fonctionnement des systèmes séquentiels asynchrones que des entrées non corrélées ne peuvent pas être synchrones (David et Alla, 1992). C'est-à-dire deux entrées ne peuvent pas changer de valeur en même temps. Ce qui implique que lors d'un passage d'un état à un autre, une seule entrée change. Considérons l'exemple de la Figure V-6 et notamment le vecteur d'entrées $\overrightarrow{e_i} = (1|0)^T$ permettant de faire passer le composant logique de l'état 1 vers l'état 5. Dans l'état 1, les deux entrées sont à l'état bas (cf. sur la Figure V-6 les transitions des états 4 ou 7 vers l'état 1). Cela signifie donc que dans l'état 1, le vecteur d'entrées vaut $\overrightarrow{e_i} = (0|0)^T$. On voit donc que la composante a_i^1 est passée de la valeur 0 à la valeur 1. Cette transition correspond donc à un front montant sur l'entrée a_i^1 (ra_i^1). Notons que cette transition ne modifie pas la valeur de la sortie S_i^k qui reste à 0. Par contre, la transition de l'état 2 vers l'état 3 correspond au même évènement mais cette fois-ci elle fait changer la valeur de la sortie qui passe de 0 à 1. Si on fusionne les états 1 et 2 dans un même macro-état, il faut différentier ces deux transitions car elles ne conduisent pas au même état. Un moyen consiste à prendre en compte la valeur des autres composantes du vecteur d'entrées lors de l'occurrence de l'évènement. Ainsi nous pouvons constater sur la Figure V-7 que cela conduit à prendre en compte que dans la transition de l'état 1 à l'état 5, $b_i^2 = 0$ alors que lors de la transition de 2 vers 3, $b_i^2 = 1$. Nous notons donc les deux transitions respectivement $< 1 > \xrightarrow{b_i^2 = 0 * ra_i^1} < 5 >$ et $< 2 > \xrightarrow{b_i^2 = 1 * ra_i^1} < 3 >$. Par analogie avec les automates temporisés nous qualifions de garde l'équation logique associée à un évènement de transition.

Le résultat obtenu ici peut être généralisé à toute composant logique munit de n entrées. Les conditions d'une transition d'un état vers l'autre peuvent être exprimées à l'aide d'un ET logique entre une garde définie sur les n-1 variables qui ne changent pas de valeurs par rapport à l'état source et un événement par rapport la n^{ème} variable qui change de valeur.

La deuxième raison qui nous amène à utiliser le graphe d'états à transitions événementielles gardées est l'objectif final de construire un diagnostiqueur fondé sur les automates temporisés. Nous nous sommes inspirés du formalisme utilisé par UPPAAL, qui utilise des transitions avec une garde sur les variables et les actions qui sont proches de ce concept d'évènements gardés. Par conséquent, lors de la réécriture en automate temporisé, la relation qui résume le vecteur d'entrées est appelé « événement gardé » dans le reste de ce mémoire.

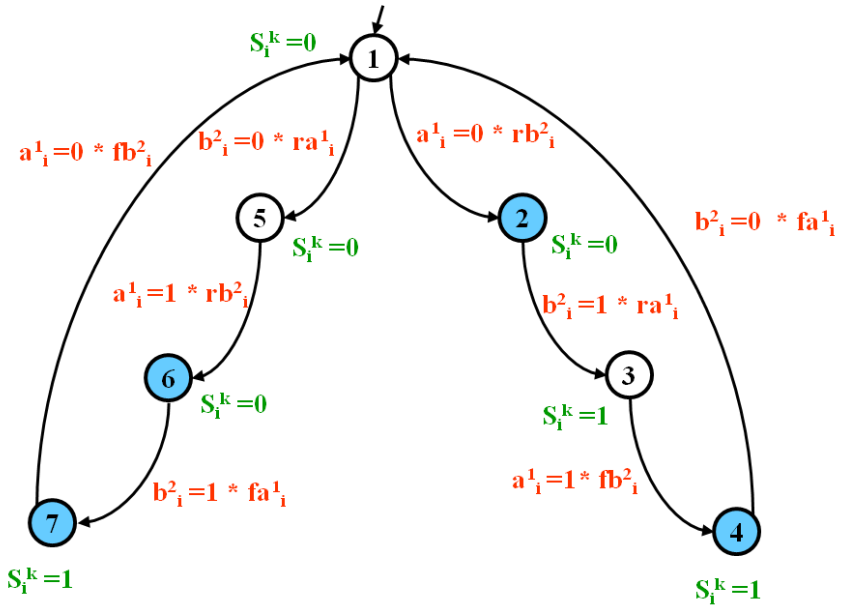


Figure V-7 – Machine de Moore à « événements gardés » équivalente à l'exemple de la Figure V-6

V.III.2.C Méthode de réduction

Cette méthode a pour objectif d'optimiser la réduction tout en respectant les exigences du diagnostic (§V.I). Tout d'abord, il faut identifier les états de référence dans le modèle initial. Nous donnons ci-dessous quelques définitions.

Définition V-2: Transition de valeur

Une transition de valeur est une transition dans le modèle à base d'événements qui modifie la valeur de la sortie. Cette transition est issue d'un état de référence. Un vecteur d'entrées de référence est le vecteur d'entrées associé à une transition de valeur.

Définition V-3: Transition d'évolution

Une transition d'évolution est une transition dans le modèle à base d'événements qui permet d'évoluer dans les états internes, mais qui maintient la valeur de la sortie de l'état précédent.

Définition V-4: Un état de référence

Un état de référence est l'état origine d'une transition de valeur.

A titre d'exemple, dans la Figure V-7, les états 2, 4, 6 et 7 sont des états de référence.

Notre algorithme de réduction est basé sur 7 règles.

Règle V-1 : Partant d'un état, regrouper dans un même macro-état, tous les états qui ont la même valeur et sont accessibles uniquement avec des transitions d'évolution.

Règle V-2 : Pour chaque transition d'évolution, ajouter une transition boucle sur le macro-état.

Règle V-3 : Pour l'état atteint par une transition de valeur, créer un macro-état et appliquer la *Règle V-2*.

Règle V-4: Pour chaque transition de valeur, créer une transition entre le macro-état contenant l'état de référence de la transition et le macro-état contenant l'état atteint par la transition.

Règle V-5 : Si en appliquant la *Règle V-1*, une transition d'évolution lie deux macro-états, alors regrouper ces deux macro-états.

Règle V-6 : Si deux macro-états possèdent la même valeur de sortie et ont les mêmes macro-états précédents et les mêmes macro-états suivants par des transitions de valeur, alors regrouper ces macro-états.

Règle V-7 : Si deux macro-états ont la même valeur de sortie et sont précédés par le même macro-état et reliés à ce dernier par deux transitions de valeur marqués par le même événement gardé, alors fusionner ces deux macro-états.

Pour mieux comprendre ces règles, appliquons les à l'exemple de la Figure V-7. Nous supposons démarrer de l'état 1, l'état initial du modèle comportemental de notre composant logique. Il existe une transition d'évolution de l'état 1 vers l'état 2 et une autre transition d'évolution de l'état 1 vers l'état 5. Par conséquent, ces 3 états sont regroupés dans un même macro-état « S1 » dont la valeur de sortie est égale à 0. Il y a une autre transition d'évolution de l'état 5 vers l'état 6. Donc l'état 6 est ajouté au macro-état « S1 » (Figure V-8 a).

L'application de la Règle V-2 ajoute 3 boucles au macro-état S1, avec les événements gardés des transitions d'évolution correspondantes. Nous voyons bien sur la Figure V-8b les 3 boucles ajoutées à l'état S1. Il existe une transition de valeur de l'état 2 vers l'état 3. D'après la Règle V-3, nous créons un macro-état S2 contenant l'état 3 et dont la valeur de sortie est égale à 1 (Figure V-8c). La Règle V-4 ajoute une transition du macro-état S1 vers le macro-état S2 étiqueté par l'événement gardé $b_i^2 = 1 * ra_i^1$ (Figure V-8d). Après l'application de la Règle V-1, nous ajoutons l'état 4 au macro-état S2 avec une boucle modélisant la transition d'évolution de l'état 3 vers l'état 4. Comme il existe une transition de valeur de l'état 4 vers l'état 1, la Règle V-4 ajoute une transition du macro-état S2 vers le macro-état S1. Le même raisonnement nous permet de créer le macro-état S3 qui contient uniquement l'état 7. Une transition étiquetée par l'événement gardé de la transition de valeur de l'état 6 vers l'état 7 lie

le macro-état S1 au macro-état S3. Ensuite une autre transition est ajoutée de S3 vers S1 pour modéliser la transition de valeur de l'état 7 vers l'état 1. Le modèle obtenu est représenté par la Figure V-9a.

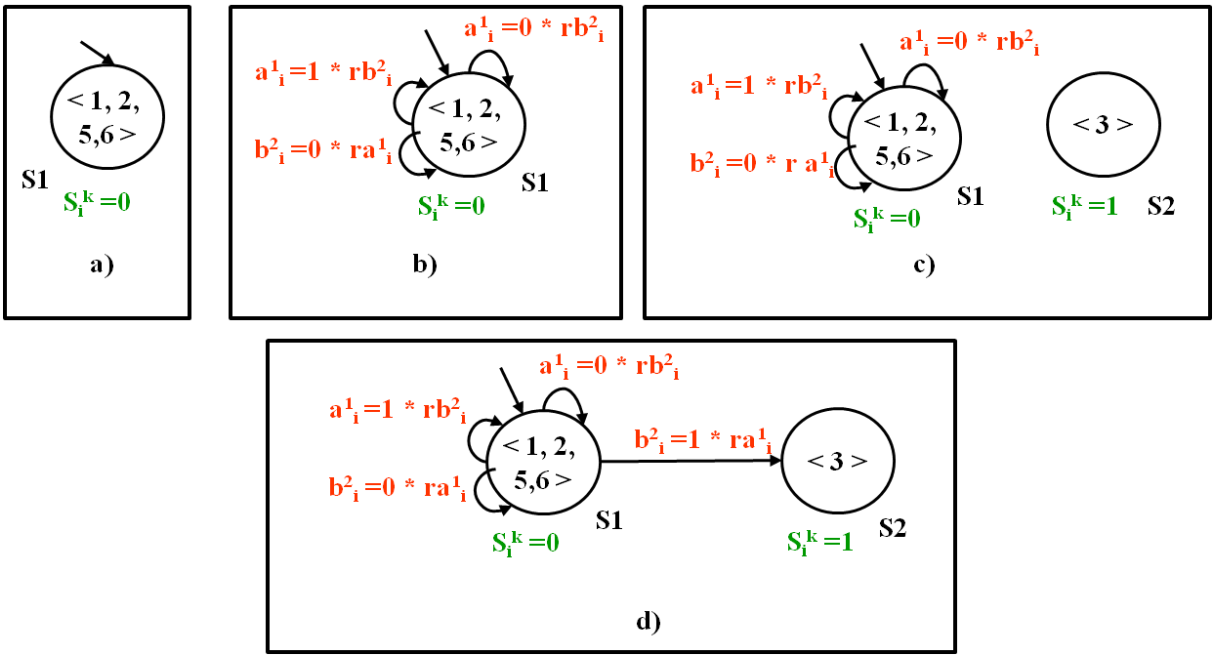


Figure V-8 – Application des quatre premières règles de réduction sur l'exemple de la Figure V-7

La Règle V-5 peut être appliquée dans le cas suivant. Supposons que l'algorithme de réduction commence par l'état 5. Ceci conduirait à regrouper les états 5 et 6 dans un macro-état S1. Ensuite, le macro-état S3 serait créé pour l'état 7. Après, en appliquant la Règle V-3, un macro-état S0 serait créé à la suite de la transition de valeur de l'état 7 vers l'état 1 et parce que l'état 1 n'est pas encore dans un macro-état. Toutefois, comme il y a une transition d'évolution de l'état 1 vers l'état 5, l'application de la Règle V-5 mène à regrouper le macro-état S0 et le macro-état S1 en un seul macro-état S1. Cet exemple nous montre que la Règle V-5 est en effet une conséquence de la Règle V-1.

Considérons la machine à états réduite de la Figure V-9a. Les macro-états S2 et S3 correspondent à la même valeur de sortie 1. Ils ont le même macro-état précédent S1 et le même macro-état suivant S1. Par conséquent, et d'après la Règle V-6 ils peuvent être regroupés en un seul macro-état qui représente une abstraction du comportement du composant lorsque sa valeur de sortie est égale à 1. La Figure V-9b représente le résultat.

La Règle V-7 permet de supprimer l'indéterminisme. Si depuis un macro-état, on peut avoir deux destinations différentes en franchissant deux transitions de valeur avec le même événement gardé, ceci présente un indéterminisme qu'on doit supprimer. Maintenant, pour assurer le déterminisme de la machine à états réduite basé sur les événements, nous donnons une proposition suffisante.

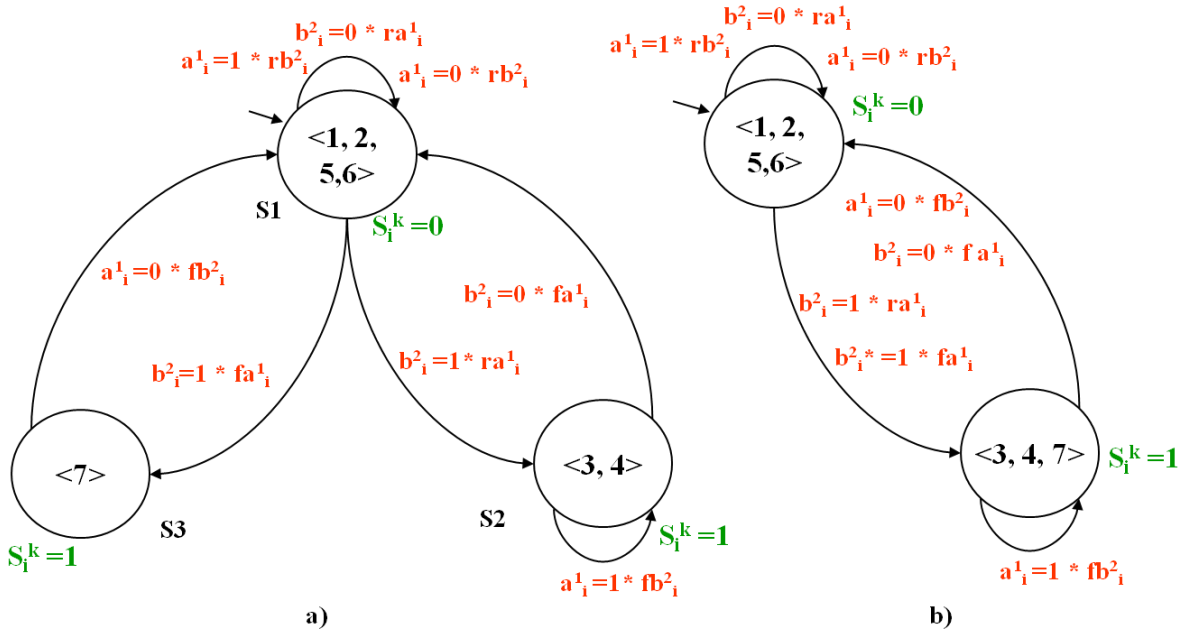


Figure V-9 – Application des règles 5 et 6 sur l'exemple de la Figure V-7

Proposition V-1 : Réduction déterministe

La réduction de la machine à états à base d'événements donne une machine à états déterministe s'il n'y a pas de macro-état qui intègre deux états reliés par une transition d'évolution étiquetée par un événement gardé et un état de référence associé duquel est issu une transition de valeur associé au même événement gardé et permettant de transiter vers un autre macro-état.

Preuve: Un indéterminisme dans le modèle réduit est produit lorsque depuis un macro-état, nous avons deux transitions différentes étiquetées par le même événement gardé vers deux macro-états différents. Deux cas peuvent avoir lieu :

Cas 1: une transition boucle dans un macro-état S_i avec une transition de valeur de S_i vers S_j.

Si nous avons une transition boucle dans S_i, donc il existe deux états e_k et e_n internes dans S_i qui sont reliés par une transition d'évolution étiquetée par l'événement gardé relatif à cette boucle. S'il existe une transition d'évolution de S_i vers S_j, ceci implique qu'il existe un état e_m interne dans S_i relié à un état e_z interne dans S_j par une transition de valeur étiquetée par l'événement gardé.

Cas 2: une transition de valeur du macro-état S_i vers le macro-état S_j et une autre transition de valeur de S_i vers S_k.

Ce cas est résolu par la *Règle V-7* et ne peut donc pas être présent dans le modèle réduit final.

Remarque : Le lecteur trouvera en annexe B un algorithme implémentant ces règles.

V.IV Méthode de modélisation pour le diagnostic des systèmes embarqués logiques

A partir des exigences du diagnostic, notre objectif est de construire un observateur de taille minimale permettant la construction d'un diagnostiqueur répondant à ces exigences.

Dans cette étude, nous proposons une nouvelle méthode qui exploite les avantages d'une machine à états dont les transitions sont basées sur les événements. Cette méthode réduit le comportement normal de la machine à états à un observateur qui doit être déterministe. Le processus de construction est basé sur les étapes suivantes :

Étape 1 : Transformer le comportement normal du composant logique en une machine à états à base d'événements.

Étape 2 : Appliquer l'algorithme de réduction d'états à base d'événements.

Étape 3 : Transformer le modèle précédent en un automate temporisé en considérant l'abstraction temporelle du système.

Étape 4 : Construire le diagnostiqueur à partir de l'automate temporisé.

Nous allons illustrer notre approche sur l'exemple de la carte FerroCOTS.

V.IV.1 Description de la carte FerroCOTS

Afin de simplifier la réalisation des modèles, nous avons limité la carte à trois signaux en entrées et deux signaux en sorties. Sur cette base, nous supposons que la carte est constituée de trois répartiteurs notés R_j, $j \in [1..3]$, trois cartes filles notées DC_i , $i \in [1..3]$ et deux voteurs notés V_k, $k \in [1..2]$ (Figure V-10).

La fonction d'un répartiteur est de répartir un signal d'entrée entre les différentes cartes filles. Ainsi, un répartiteur peur être représenté comme un système logique à une entrée er_j et trois sorties sr_j^i, l'exposant i indiquant la carte fille à laquelle est destinée la sortie (Figure V-11a). Nous faisons l'hypothèse ici qu'une faute d'un répartiteur affecte ses trois sorties.

Chaque carte fille DC_i est conçue autour d'un composant programmable disposant d'entrées/sorties, d'une entrée d'horloge pour la synchronisation de son comportement et d'une entrée « reset » pour son initialisation.

La carte dispose par ailleurs, de ports d'entrées (trois ici, notés e_i^j) et de ports de sorties (deux ici, notés S_i^k) lui permettant de recevoir en entrée les signaux issus des répartiteurs et de fournir ses sorties destinées aux différents voteurs. Nous supposons par ailleurs que les constituants internes du composant programmable mettent en œuvre une fonction $F_i^k =$

$f_k(\vec{e_i})\ avec\ \vec{e_i} = \sum_{j=1}^{n} \vec{e_i}^{j}$ permettant de générer le signal de la sortie S_i^k , avec i l'indice de la carte fille et k l'indice du voteur concerné par la sortie (Figure V-11b). Nous supposons que chaque entrée et chaque sortie de chaque fonction de carte fille peut être affectée par une faute. De même chaque fonction interne F_i^k d'un composant programmable peut être affectée par une faute que nous supposerons ici la même quelque soit le constituant interne cause de la faute.

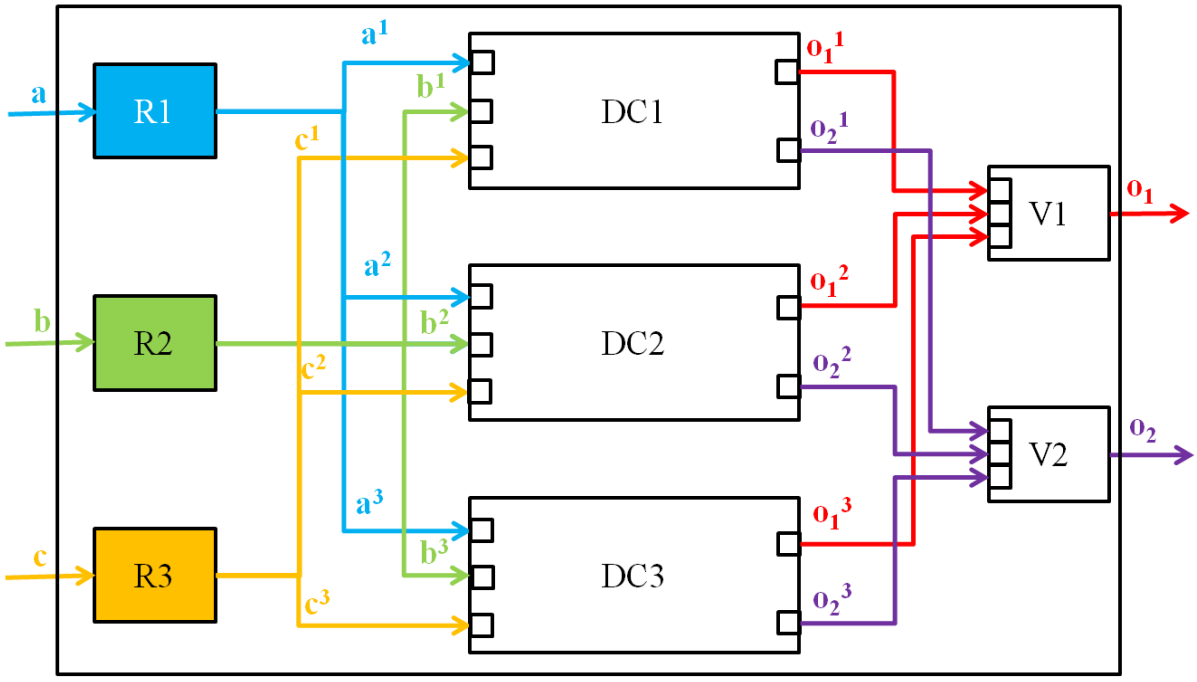


Figure V-10 – Schéma simplifié de la carte FerroCOTS

Remarque : Dans ce travail nous supposons que chaque carte fille dispose de sa propre horloge. Cette contrainte matérielle entraine que les sorties des cartes envoyées vers un voteur ne sont pas synchrones. Cet asynchronisme induit une contrainte de modélisation supplémentaire qu'il faudra prendre en compte.

Le troisième type de constituant de la carte FerroCOTS que nous considérons est le type voteur. Pour notre application, la carte FerroCOTS est sensée mettre en œuvre des votes 2 sur 3 (2oo3). Donc chaque voteur possède trois ports d'entrées ev_k^i avec i l'indice de la carte fille auquel est connecté le port, et un port de sortie sv_k. Chaque port d'entrée d'un voteur peut être en faute indépendamment des autres ports. Le port de sortie peut également être en faute.

La Figure V-12 résume la décomposition structurelle de cette carte dans laquelle les répartiteurs, les cartes filles et les voteurs sont vus comme des sous-systèmes.

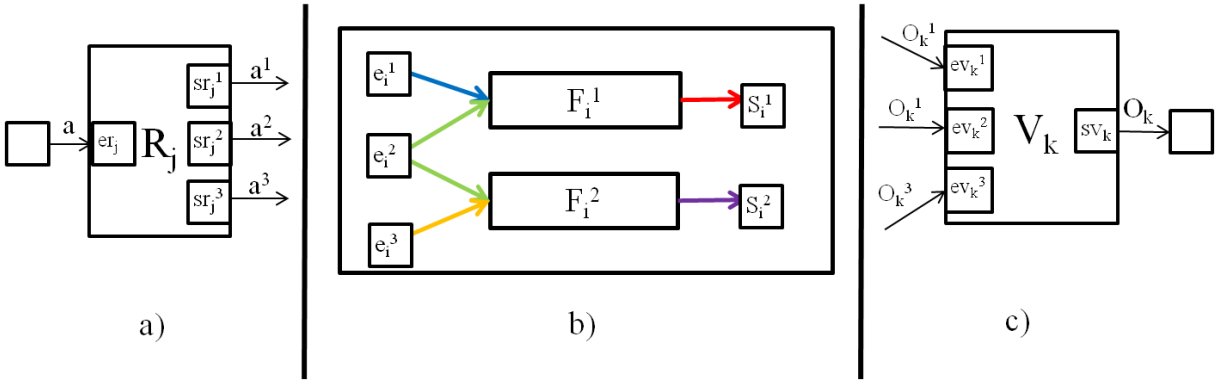


Figure V-11 – Spécification des entrées/sorties de la carte FerroCOTS

Du point de vue d'une carte fille, une ligne fonctionnelle comprend l'ensemble des composants qui permet de mettre la sortie S_i^k à 1 ou à 0 compte tenu du vecteur d'entrées pris à une date donnée. Par exemple, si on se réfère à la Figure V-11, la ligne fonctionnelle de S_i^1 comprend les composants $er_1, er_2,\ sr_1^i, sr_2^i, e_i^1, e_i^2, F_i^1, et\ s_i^1$. Du point de vue du voteur V_1, la ligne fonctionnelle de sa sortie comprend les trois lignes fonctionnelles des trois sorties des cartes filles qui alimentent ce voteur, plus les composants du voteur $sv_1,\ ev_1^1, ev_1^2, ev_1^3$.

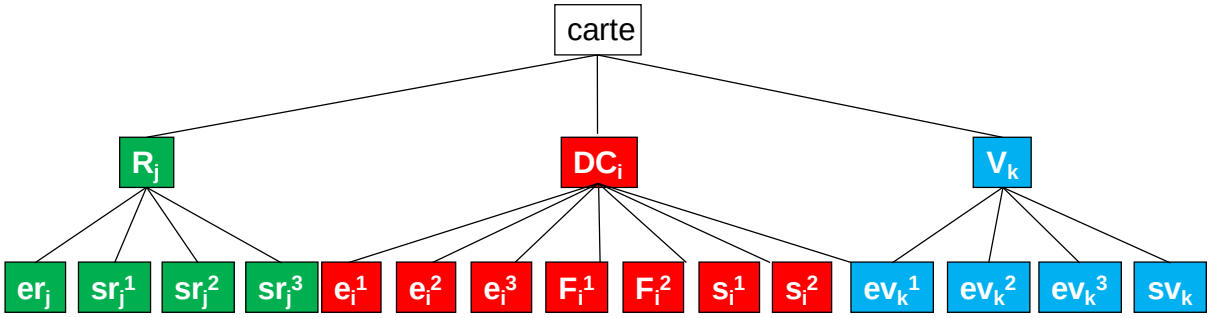


Figure V-12 – Décomposition structurelle de la carte FerroCOTS

L'ensemble de ces composants sont susceptibles d'être en faute. La détection revient à dire si une ligne fonctionnelle est défaillante. Le diagnostic consiste à identifier le composant responsable de la faute.

Remarque : Dans un souci de simplification, nous n'avons pas pris en compte la possibilité de fautes induites par les pistes électronique, ni les défaillances des composants internes des voteurs.

V.IV.2 Modèle générique d'un port d'entrée ou de sortie

Un port de constituant peut être vu comme un système à retard permettant de transmettre un signal d'entrée en un signal de sortie (Figure V-13).

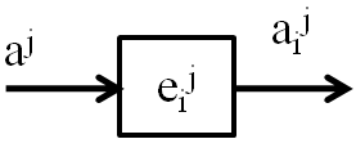


Figure V-13 – Schéma générique d'un port d'E/S

Nous proposons donc d'abstraire le comportement de touts les ports d'entrée ou de sortie du système par le schéma de la Figure V-14a.

La Figure V-14b schématise la manière dont nous allons désormais représenter cette abstraction. Nous noterons ra^j le front montant sur le signal d'entrée a^j et fa^j sont front descendant. Lorsque le signal en entrée du port change d'état (par exemple observation de ra^j), le comportement normal du port consiste à générer un signal en sortie (par exemple ra_i^j) dans un délai compris entre 1 ut et 10 ut. Si au bout de 10 ut de temps, il n'y a pas génération de ce signal, cela correspond à une faute de e_i^j que nous noterons de_i^j.

Remarque : Nous noterons désormais les fautes d'un constituant en mettant la lettre 'd' devant le nom de ce constituant.

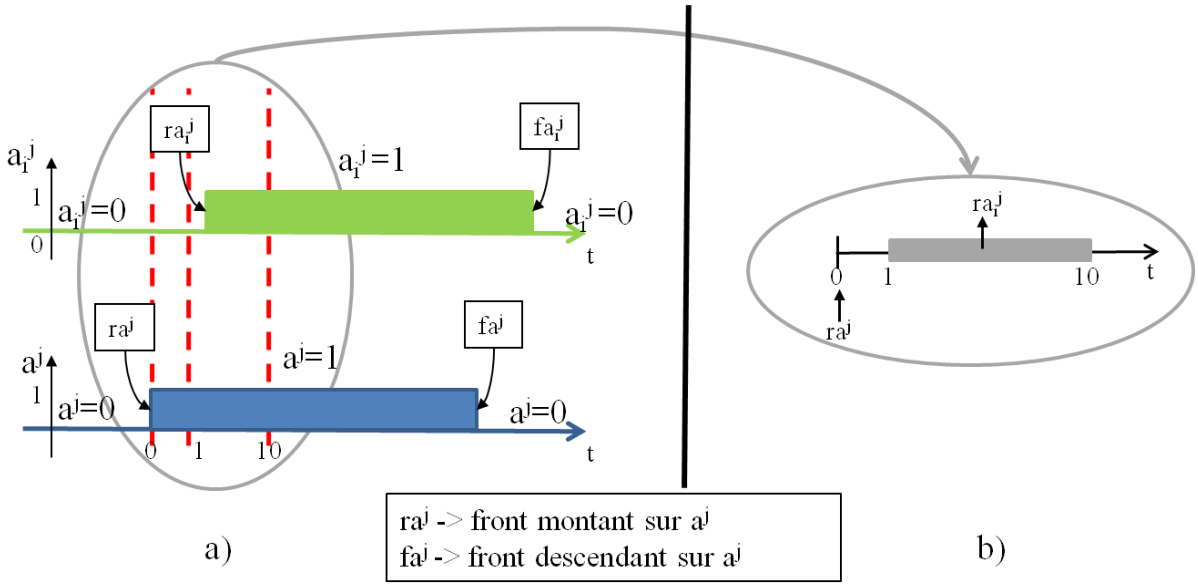


Figure V-14 – Abstraction temporelle du comportement d'un port d'E/S

Compte tenu de l'abstraction temporelle et étant donné que la fonction d'un port logique est de recopier en sortie le signal qu'il a en entrée, nous pouvons proposer un modèle générique pour modéliser son comportement normal (Figure V-15). Ce modèle est un automate temporisé qui modélise le passage d'un signal du niveau bas au niveau haut (transition de l'état 0 vers l'état 2) et inversement le passage d'un état haut vers un état bas (transition de l'état 2 vers l'état 0). Nous avons supposé dans cette modélisation que par défaut le composant est dans l'état bas (i.e. l'état 0 est l'état initial).

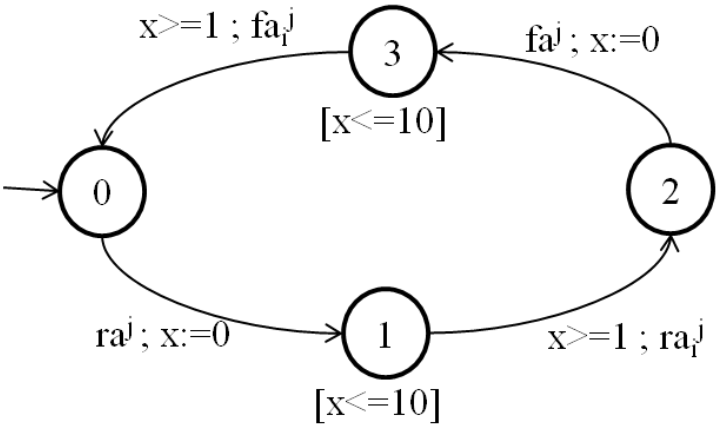


Figure V-15 – Modèle comportemental générique d'un port d'entrée ou de sortie

Dans l'état 0, nous avons $a^j = 0\ et\ a_i^j = 0$. Sur l'occurrence de l'événement ra^j nous commutons vers l'état 1 en initialisant l'horloge x. Si le signal de sortie passe de l'état bas à l'état haut dans un délai de 1 ut à 10 ut, le système commute de l'état 1 vers l'état 2. La borne supérieure du délai de commutation est définie par l'invariant $[x \leq 10]$ associée à l'état 1 et la borne inférieure est modélisée par la garde $[x \geq 1]$ de la transition de l'état 1 vers l'état 2. Le comportement dual pour passer de l'état haut à l'état bas se modélise de manière analogue (Figure V-15).

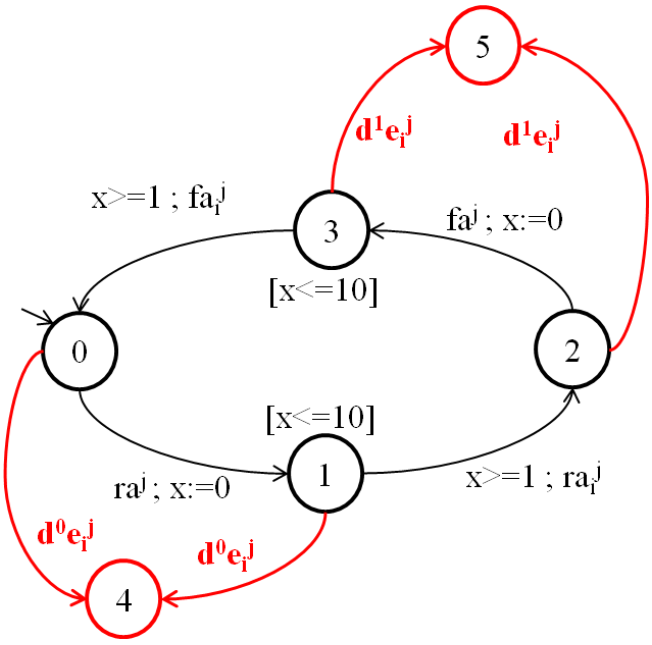


Figure V-16 – Modèle générique du comportement complet (normal et défaillant) d'un port d'entrée ou de sortie

Compte tenu de l'exigence de surveillance sur les sorties du système (cf. §V.I), il est nécessaire de distinguer deux catégories de défaillances pour les ports : le collage à 0 que nous noterons $d^0 e_i^j$ et le collage à 1 que nous noterons $d^1 e_i^j$.

Remarque : $d^0e_i^j$ et $d^1e_i^j$ sont des évènements inobservables.

Considérons le cas du collage à 0. Il peut survenir alors que le signal d'entrée du port est à l'état bas. Donc nous devons modéliser à ce moment le passage du port de l'état 0 à l'état 4 qui modélise cette faute. Même si c'est moins probable, il peut également arriver que le collage à 0 se produise entre le moment où le système reçoit le signal en entrée et le moment où il doit générer sa sortie. C'est la raison pour laquelle nous prévoyons une transition de l'état 1 vers l'état 4 sur l'occurrence de la faute $d^0e_i^j$. La prise en compte du collage à 1 s'effectue de manière analogue et amène à ajouter l'état 5 (Figure V-16).

V.IV.3 Transformation de modèles pour la construction systématique de l'automate temporisé du comportement normal d'un composant logique

Nous souhaitons construire l'automate temporisé qui représente le modèle comportemental temporel du composant logique. Pour cela nous prenons en compte deux abstractions : le graphe d'états à transitions événementielles gardées et une abstraction temporelle du comportement logique.

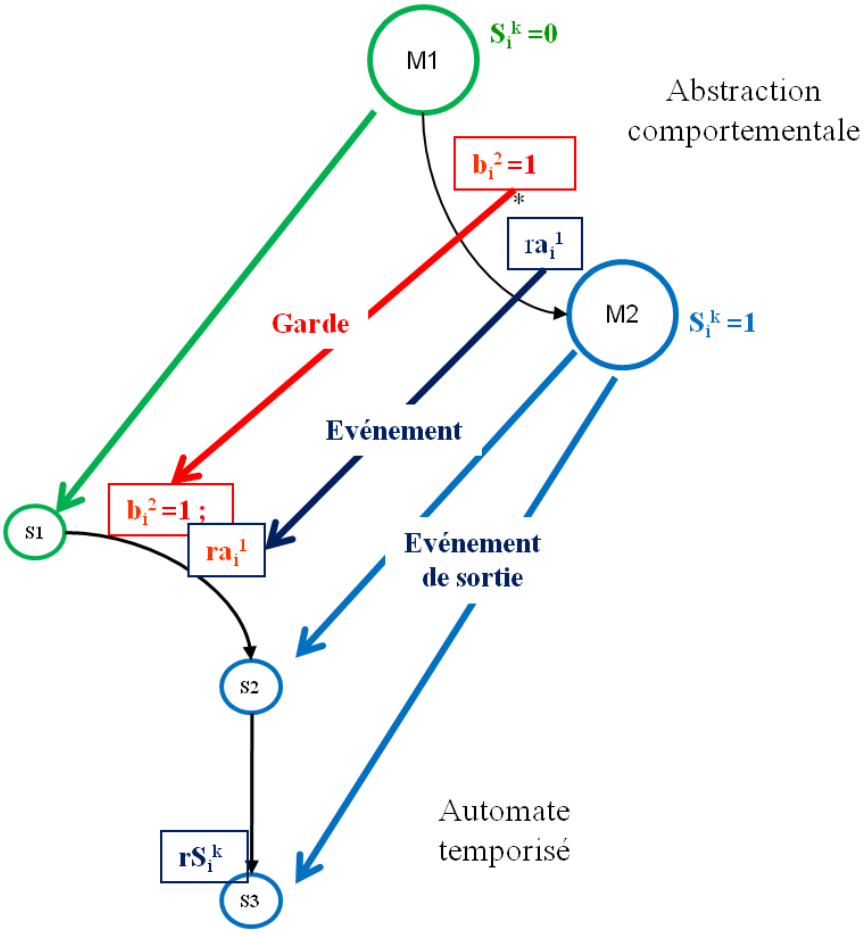


Figure V-17 – Etape 1 de la transformation de modèles

L'abstraction temporelle est un moyen de caractériser la performance du composant logique. Nous supposons ici qu'étant donné un état interne permettant un changement de valeur en sortie d'un composant logique, une modification du vecteur d'entrées entraine ce changement dans un intervalle de temps compris entre 20 et 90 ut. Cette spécification comprend le temps de propagation des signaux d'entrées dans la carte fille, leur traitement par le composant programmable et la propagation du résultat jusqu'à la sortie.

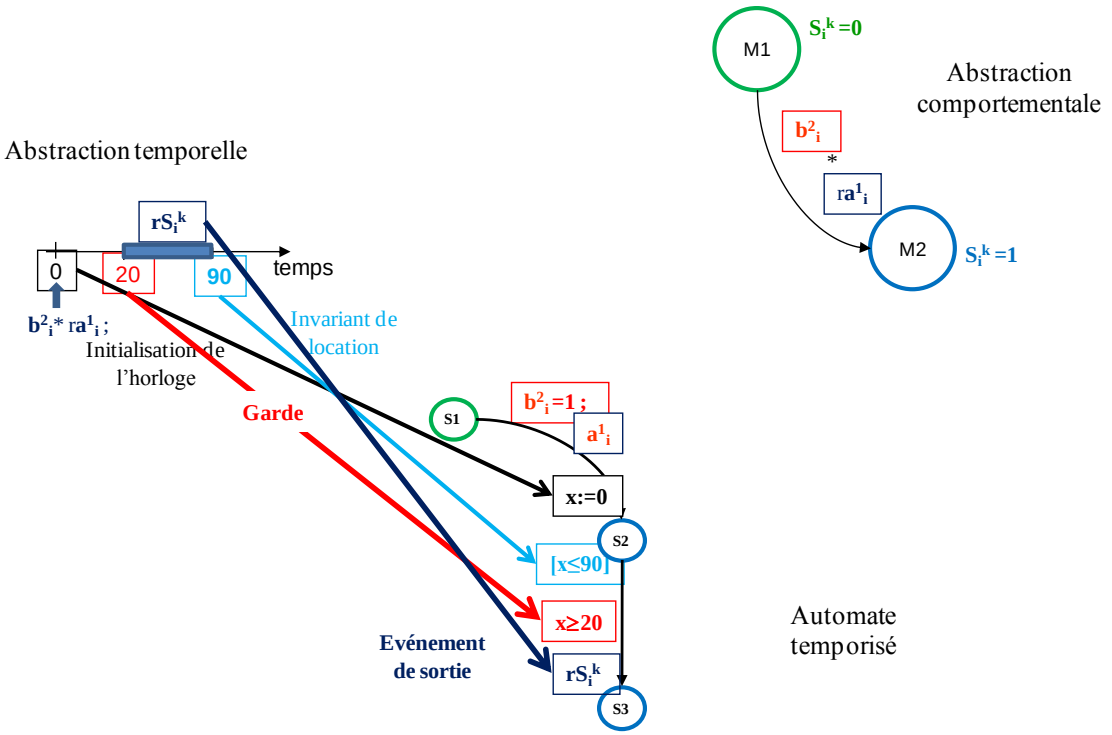


Figure V-18 – Etape 2 de la transformation de modèles

La première étape de la transformation de modèle consiste en la transformation du graphe d'état à transitions gardées en automate temporisé (Figure V-17). Chaque transition de valeur est modélisée par deux transitions. La première transition (comme par exemple la transition $S_1 \rightarrow S_2$) modélise la modification de valeur du vecteur d'entrées de référence et la deuxième transition ($S_2 \rightarrow S_3$) modélise l'occurrence de l'événement de changement de la sortie. En effet, dans le modèle source, le franchissement d'une transition résulte de l'occurrence d'un événement. Suite à l'occurrence de cet événement, la valeur de la sortie change. Donc, deux actions sont produites : l'événement qui entraine la transition et le changement de valeur de la sortie. L'équation logique de la transition de valeur est traduite par une garde dans la première transition et l'événement lui-même est traduit par une action dans cette transition. Ainsi la transition $M_1 \xrightarrow{g.e} M_2$ avec g la garde et e l'événement est transformée en la transition $S_1 \xrightarrow{g\,;\,e} S_2$ avec juste une adaptation à la syntaxe des automates temporisés. L'état destinataire (M_2) dans le modèle source est transformé en deux états S_2 et S_3 dans le modèle cible. Nous créons alors la transition $S_2 \xrightarrow{rS_i^k} S_3$ qui modélise le changement

de la valeur de la sortie en conséquence du changement du vecteur d'entrées dans un état de référence.

La deuxième étape de la transformation consiste à compléter le modèle obtenu lors de l'étape 1 par la prise en compte de l'abstraction temporelle (Figure V-18).

L'abstraction temporelle est prise en compte en initialisant une horloge x dans la première transition pour définir des temps relatifs au changement du vecteur d'entrées. L'hypothèse que la sortie S_i^k doit se produire entre 20 ut et 90 ut après l'occurrence du vecteur d'entrées de référence, est prise en compte par l'invariant de l'état S2 pour la borne maximale et la garde de la transition de S_2 vers S_3 qui est réécrite en $S_2 \xrightarrow{x\geq 20\ ;\ rS_i^k} S_3$.

La transformation du modèle réduit de la Figure V-9b et la prise en compte de l'abstraction temporelle, permettent d'obtenir le modèle de comportement normal d'un composant logique (Figure V-19).

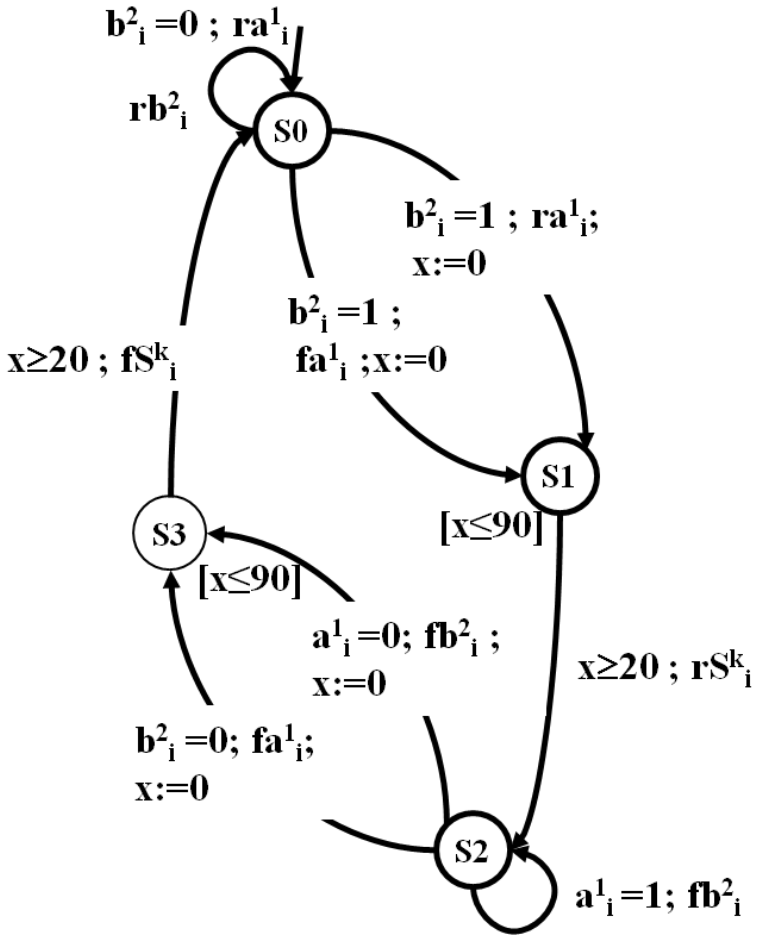


Figure V-19 – Automate temporisé résultant de la transformation de modèles

V.IV.4 Obtention du modèle complet d'un composant logique

Selon les exigences du diagnostic, nous pouvons constater que deux fautes peuvent avoir lieu au sein d'un composant logique affectant sa sortie : $d^1F_i^k$ (respectivement $d^0F_i^k$) qui représente une faute qui empêche le composant de mettre sa sortie à 0 (respectivement à 1). Au modèle de comportement normal nous ajoutons deux états pour représenter les deux états de fautes correspondants : l'état S4 et l'état S5. Ainsi la faute $d^0F_i^k$ peut avoir lieu soit avant l'occurrence du vecteur d'entrées de référence (transition de S0 vers S4), ou bien après son occurrence (transition de S1 vers S4 dans la Figure V-20). Dans les deux cas, la sortie ne passe plus à 1 suite à l'occurrence de cette faute et le composant se retrouve dans l'état S4 (Figure V-20). Un raisonnement similaire permet de prendre en compte le deuxième type de faute entrainant des transitions de S2 ou S3 vers S5.

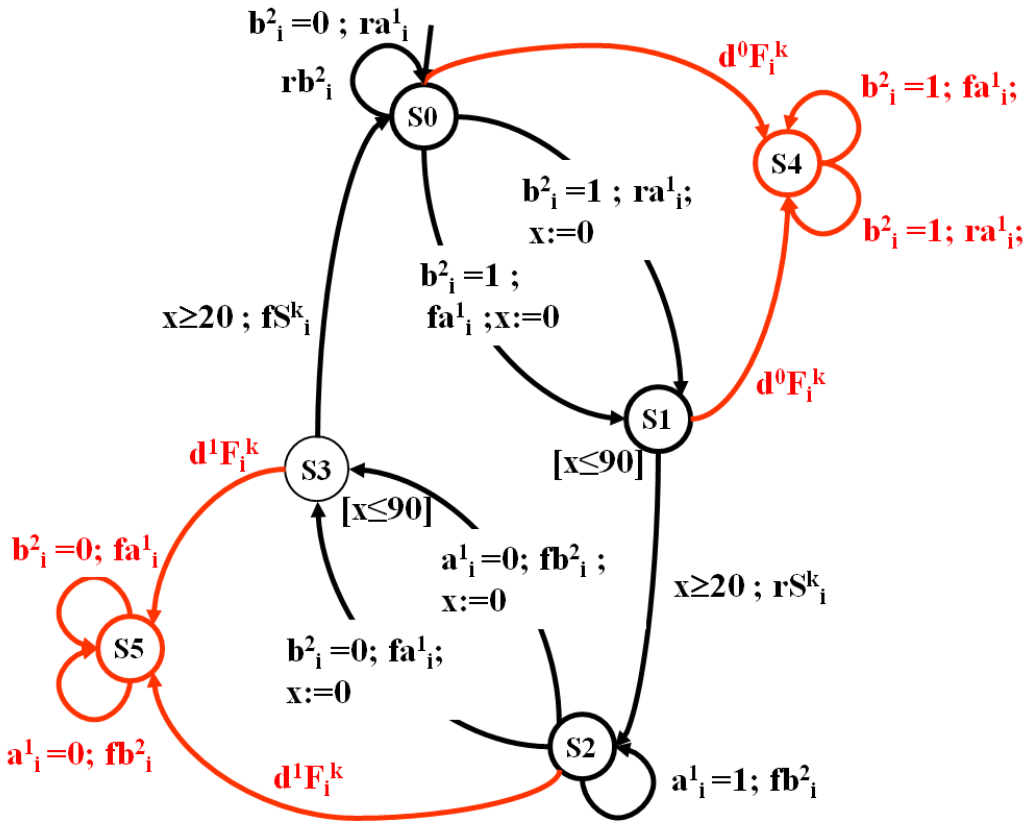


Figure V-20 – Modèle de comportement complet d'un composant logique à deux entrées

V.V Relation entre profondeur d'une modélisation d'un système complexe et explosion combinatoire

L'objectif ici est de montrer que la complexité des systèmes embarqués logiques nécessite une modélisation adaptée pour combattre l'explosion combinatoire. Pour cela, nous allons prendre l'exemple de la carte FerroCOTS et plus précisément le cas de la surveillance d'une ligne fonctionnelle réduite aux éléments d'une carte fille. Si nous supposons que la fonction logique

met en œuvre le comportement donné par la Figure V-6b, une telle ligne fonctionnelle comprend les quatre éléments $e_i^1, e_i^2, F_i^1, et\ s_i^1$. Le lecteur trouvera en annexe C les modèles comportementaux obtenus par instanciation des modèles génériques donnés par la Figure V-16 pour les portes d'entrées ou de sorties, et la Figure V-20 pour la fonction logique F_i^1.

Le Table V-1 donne une synthèse du nombre d'états et de transitions des différents modèles. Nous constatons que le modèle de la ligne fonctionnelle réduit à une carte fille en prenant en compte uniquement deux entrées se traduit par 648 états et 4482 transitions pour identifier 8 fautes. Nous voyons ainsi que la capacité d'identification précise d'une faute par l'approche diagnostiqueur se traduit par une explosion combinatoire notamment quand nous essayons de construire des modèles monolithiques.

Entité	Nombre d'états	Nombre de transitions	Nombre de fautes modélisées
Fik	6	20	2
ei1	6	10	2
ei2	6	10	2
Sik	6	10	2
cartesien(Fik,ei1,ei2,Sik)	1296	20000	8
sync(Fik,ei1,ei2,Sik)	648	4482	8

Table V-1 – Nombre d'états et de transitions pour les différents modèles d'une ligne fonctionnelle

Pour combattre cette explosion combinatoire, nous proposons une approche basée sur la modularité.

V.VI Diagnosticabilité modulaire des automates temporisés

Afin de lutter contre l'explosion combinatoire constatée dans la partie précédente et de déterminer l'observabilité minimale à avoir pour garantir la diagnosticabilité d'un système, nous définissons la diagnosticabilité modulaire pour les automates temporisés. Ce travail est inspiré des travaux sur la diagnosticabilité modulaire des systèmes à événements discrets (SED) (Contant et al., 2006) et des travaux sur le diagnostic des automates temporisés (Tripakis, 2002)(Bouyer, et al., 2005a)(Baier, et al., 2009). Il consiste donc à développer une approche pour mettre en œuvre la diagnosticabilité modulaire dans le cadre des systèmes temporisés.

V.VI.1 Adaptation de la diagnosticabilité modulaire

Conformément aux résultats du chapitre III, étant donné un système T décomposable en sous-systèmes S, eux même décomposables en composants i, nous distinguons 3 types de diagnosticabilité :

- Un composant i est localement diagnosticable si toute défaillance de ce composant est diagnosticable à l'aide uniquement des événements observables de ce composant ;
- Un sous-système S est dit modulairement diagnosticable si au niveau S toute défaillance d'un de ses composants i est diagnosticable à l'aide de mots temporisés des événements observables de S comportant suffisamment d'événements de i ;
- Un sous-système S est dit globalement diagnosticable si toute défaillance du système est diagnosticable à l'aide des événements observables de S.

Soit $\omega \in L(G_t)$, nous notons $Unt(\omega) = a_1 a_2 \dots a_n$ la séquence composée par des événements de G_t.

Définition V-5 : Diagnosticabilité modulaire d'un automate temporisé

Etant donné $T = \{1, \dots, p\}$ *un système temporisé et* $S \subseteq T$ *avec* $G_t^S = ||_{z \in S} G_t^z$. *Le langage* $L(G_t^S)$ *est modulairement diagnosticable par rapport à* $\Sigma^s{}_o$ *et* $\Sigma^s{}_f$ *si* $\forall i \in S$, $\forall f \in \Sigma^i{}_f$, $\forall \omega \in L(G_t^S) \,/\, Unt(\omega) \in \varphi(\Sigma_f^i)$, $\exists \Delta^i \in \mathbb{N} \,/\, \forall \rho \in L(G_t^S)/\omega$, $duree\left(P_o^i(\rho)\right) \geq \Delta^i \Longrightarrow$ $D^i(\omega\rho) = 1$

Avec $D^i(\omega\rho) = 1$ */ si* $\omega' \in (P_o^i)^{-1}\,(P_o^i(\omega\rho)) \Rightarrow f \in \omega', sinon\ D^i(\omega\rho) = 0.$

La Définition V-5 signifie qu'un automate temporisé est modulairement diagnosticable si pour toute faute originaire dans un module, au bout d'un délai fini propre à ce module, au niveau du système, on aura observé suffisamment d'événements de ce module pour diagnostiquer l'occurrence de la faute.

Dans le cas des SED, le test de la diagnosticabilité modulaire est réalisé en recherchant si le diagnostiqueur de G_t^S contient des cycles d'états F^{Mi}-indéterminés. Nous ne pouvons pas reconduire ce type de test dans le cas temporisé pour deux raisons. La première raison est le fait que le diagnostiqueur d'un automate temporisé n'est pas dans le cas général un automate temporisé déterministe. On ne peut donc pas se contenter de rechercher des cycles dans un automate temporisé. La deuxième raison est que l'aspect temporisé entraine que les exécutions infinies ne se traduisent pas forcément pas des cycles dans un modèle. En effet, comme nous l'avons vu au chapitre IV, les exécutions zénons sont des exécutions infinies qui bloquent le temps.

Pour tester la diagnosticabilité des systèmes temporisés, il existe deux méthodes. Etant donné un système G_t la première méthode consiste à rechercher les exécutions non-zénon du modèle $(G_t^1 ||_o G_t^2)^{-f2}$, où G_t^1 et G_t^2 sont des copies de G_t avec un re-nommage des événements inobservables (Tripakis, 2002). La deuxième méthode consiste à vérifier le vide d'un automate de Büchi (Cassez et al., 2008). Cet automate que nous avons appelé D, est obtenu en réalisant le produit synchrone de $(G_t^1 ||_o G_t^2)^{-f2}$ avec l'automate $Div(x)$ (chapitre IV, § IV.I.3, page 126).

Proposition V-2 : Test de la diagnosticabilité modulaire

Etant donné $T = \{1, \dots, p\}$ *un système temporisé et* $S \subseteq T$ *avec* $G_t^S = ||_{z \in S} G_t^z$. *Le langage* $L(G_t^s)$ *est modulairement diagnosticable par rapport à* $\Sigma^s{}_o$ *et* $\Sigma^s{}_f$ *si* $\forall i \in S$, *la projection dans* $\Sigma^i{}_o$ *de toute exécution Fi-fautive de* $(G_t^1 ||_o G_t^2)^{-f2}$ *est zénon.*

Preuve :

Soit $f \in \Sigma_f^i$. Pour toute exécution Fi-fautive de $(G_t^1 ||_o G_t^2)^{-f2}$, si elle est zénon, cela entraine qu'il existe $\Delta_w^i \in \mathbb{N}$ et $w \in (G_t^1 ||_o G_t^2)^{-f2}$ / $duree(w) = duree(\mathrm{P}_o^i(w_1)) = duree(\mathrm{P}_o^i(w_2)) = \Delta_w^i$ et $\mathrm{P}_o^i(w_1) = \mathrm{P}_o^i(w_2)$ avec $w_1 \in L(G_t^1)$ et $w_2 \in L(G_t^2)^{-f2}$. Comme par hypothèse G_t^1 est bien formé, donc on peut trouver $\rho_1 \in L(G_t^1)/w_1$ tel que $D(w_1\rho_1) = 1$ *avec* $duree(\rho_1) \in \mathbb{N}$ et il n'existe pas $\rho_2 \in L(G_t^2)/w_2$ tel que $\mathrm{P}_o^i(\rho_1) \neq \mathrm{P}_o^i(\rho_2)$. Comme toute exécution fautive peut être étendue en une exécution fautive de durée finie pour laquelle on peut décider qu'elle est fautive par rapport à Σ_o^i et Σ_f^i , alors S est modulairement diagnosticable.

V.VI.2 Application au cas de la carte FerroCOTS

Afin de réduire la combinatoire constaté au §V.V, cette fois ci, nous allons considérer comme composant logique la ligne fonctionnelle du point de vue d'une carte fille et nous allons définir uniquement deux fautes : une faute caractérisant toutes les anomalies qui entraveraient le passage de la sortie de l'état bas à l'état haut considéré (notée $\mathrm{d}^0\mathrm{f}_\mathrm{i}^\mathrm{k}$), et la faute duale entravant son passage de l'état haut à l'état bas (notée $\mathrm{d}^1\mathrm{f}_\mathrm{i}^\mathrm{k}$). Compte-tenu de la spécification de ces fautes, et par instanciation du modèle complet d'un composant logique à deux entrées, nous obtenons le modèle de la Figure V-21b. On notera sur ce modèle que l'abstraction temporelle tient compte à la fois des délais induits par les ports d'entrées et de sortie et, celui induit par le composant programmable. Ainsi, sur changement du vecteur d'entrées de référence, la sortie est modifiée dans un délai compris entre 22 et 110 ut.

Le Table V-2 donne la liste des événements du modèle résultant et leur observabilité. Nous supposons ici que les signaux d'entrée et de sortie de la ligne fonctionnelle sont observables. Les fautes ne sont pas observables.

Pour l'analyse de la diagnosticabilité, nous considérons une faute à la fois. La seconde faute dans ce cadre doit être considérée comme un évènement inobservable. Nous analysons la diagnosticabilité par le test proposé dans (Cassez, et al., 2008) et rappelé au §III.I.3. Il s'agit de vérifier si l'état « bad » de l'automate « OBS » est non atteignable.

D'après le test de diagnosticabilité, le modèle de Figure V-21b est non diagnosticable. Nous avons varié la valeur de k à une valeur supérieure à 110 et l'état « bad » est atteignable ce qui signifie que le modèle n'est pas diagnosticable (Figure V-22). Cela s'explique par le fait que l'état S4 de la de la Figure V-21b n'a pas d'invariant. Du coup, dans la Figure V-22, les

exécutions fautives par rapport à $d^0f_i^k$ ne bloque pas le temps. Donc dans l'observateur OBS, on peut atteindre l'état « bad » quelque soit la valeur de la limite k définie pour l'horloge.

Evénement	Description	Observabilité
rai	front montant sur ai	observable
fai	front descendant sur ai	observable
rbi	front montant sur bi	observable
fbi	front descendant sur bi	observable
rOki	front montant sur Oik	observable
fOki	front descendant sur Oik	observable
d0fki	collage sortie à 0	inobservable
d1fki	collage sortie à 1	inobservable

Table V-2 – Observabilité des différents événements

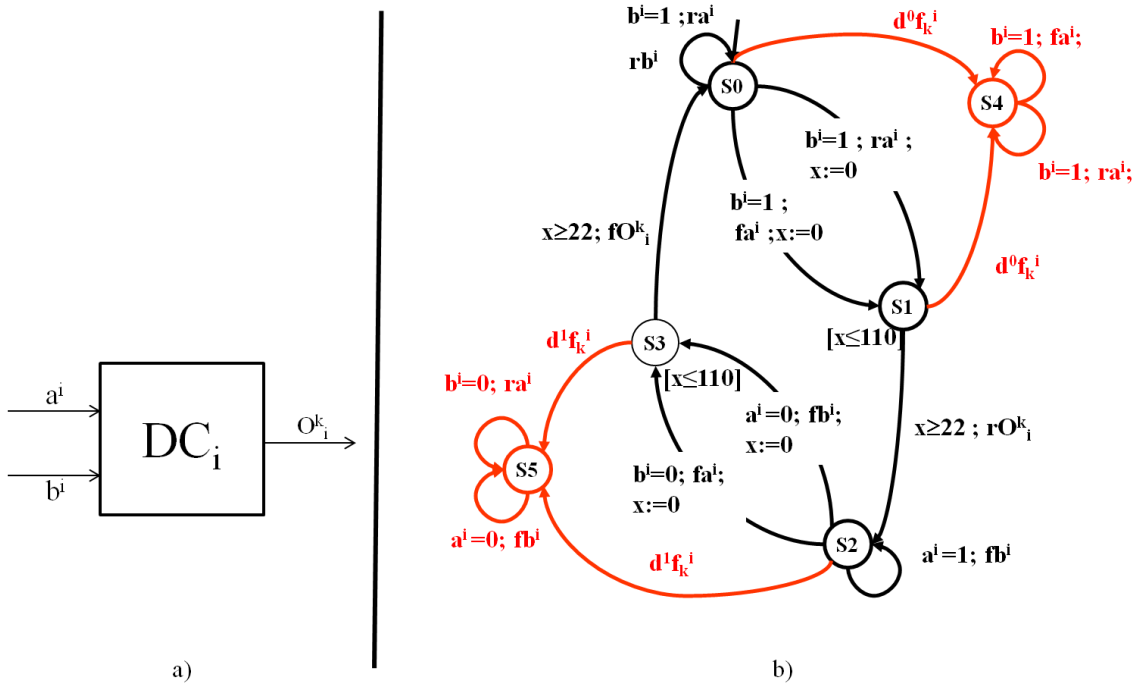
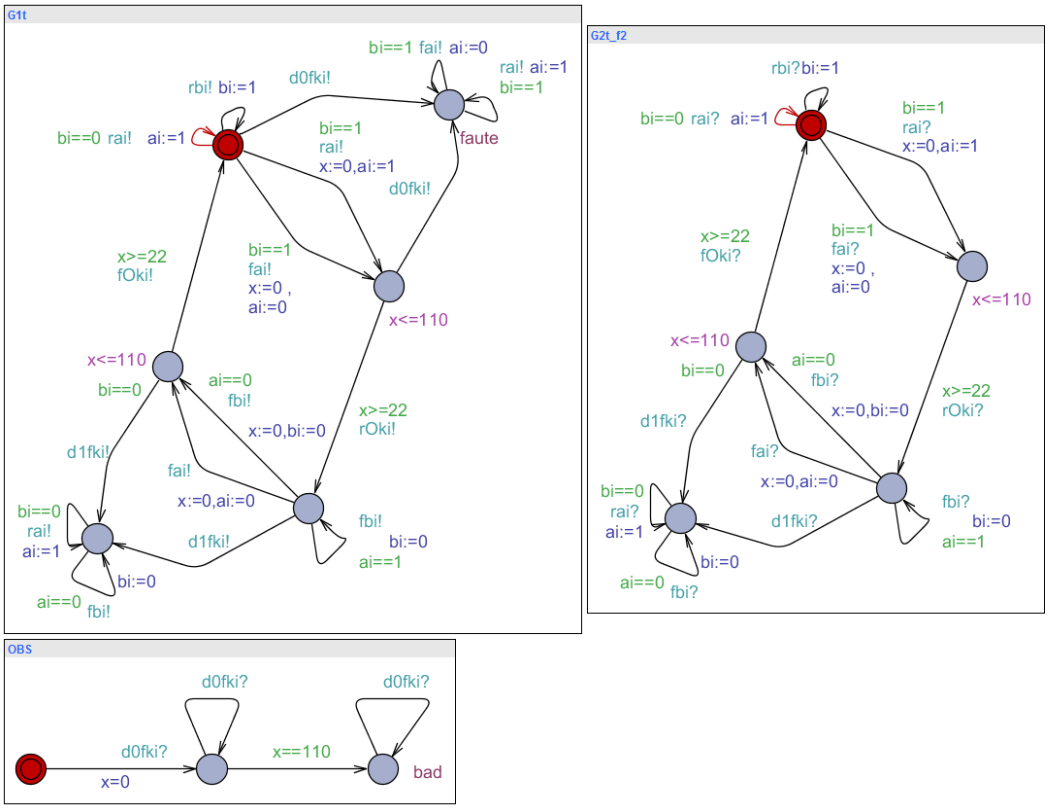


Figure V-21 – Modèle comportemental complet d'une ligne fonctionnelle d'une carte fille

Cet exemple montre une des limites de l'approche diagnostiqueur. Pour diagnostiquer une faute, nous avons besoin après l'occurrence d'une faute, de pouvoir observer un signal qui permet indirectement d'isoler la faute. Pour cela, l'idée dans notre cas d'étude est de considérer la ligne fonctionnelle constituée d'une carte fille et du voteur associé à une sortie donnée de la carte. Nous utilisons ici le principe de la diagnosticabilité modulaire. En effet, la carte fille et le voteur ayant en commun l'événement observable O_i^k. On peut évaluer la diagnosticabilité modulaire de ce sous-système du point de vue des fautes de la carte fille. C'est-à-dire que nous cherchons si ce sous-système est modulairement diagnosticable pour $\Sigma_o^{i,k} = \{ra^i, fa^i, rb^i, fb^i, rO_i^k, fO_i^k, rO_k, fO_k\}$ et les fautes $\Sigma_{uo}^i = \Sigma_f^i = \{d^0f_i^k, d^1f_i^k\}$.



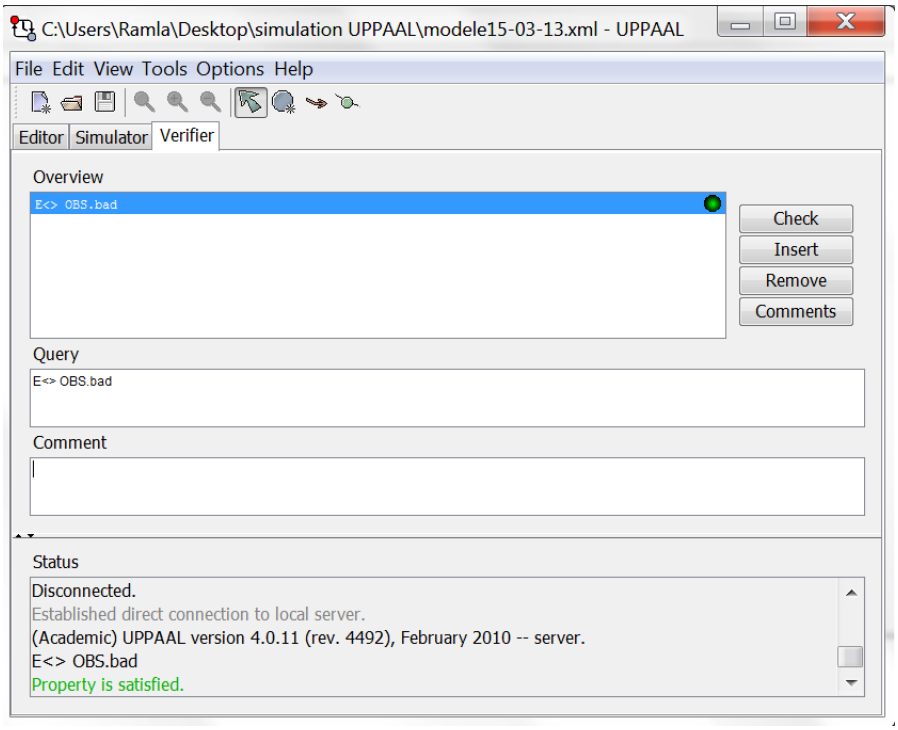


Figure V-22 – Test de diagnosticabilité du modèle de la Figure V-21 avec UPPAAL

Afin de construire le modèle de ce système, nous supposons que le voteur génère sa valeur de sortie dans un délai de 10 à 20 ut après avoir reçu deux des trois signaux pour lequel il effectue le vote.

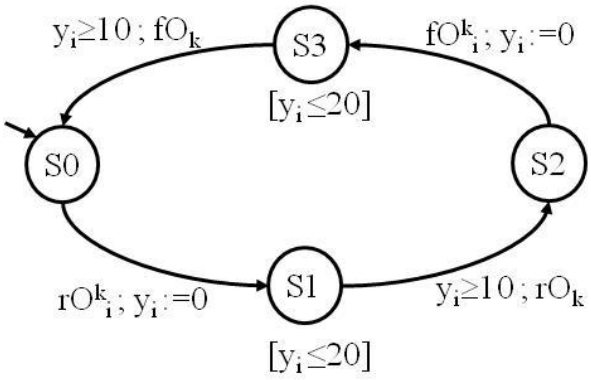


Figure V-23 – Modèle du voteur V_k par rapport à la carte fille i et la sortie k

Remarque : Cette hypothèse permet de prendre en compte l'asynchronisme de fonctionnement des trois cartes filles. Le voteur fonctionne comme un système à retard qui attend que les trois cartes aient eu le temps de lui envoyer leur signal respectif avant de procéder au vote.

Dans l'hypothèse où deux cartes ne peuvent pas être en panne en même temps, cela signifie que le voteur va changer la sortie O_k à une date ultérieure à la date à laquelle O_i^k aurait dû avoir changé d'état. Nous pouvons donc construire le modèle de la Figure V-24. Dans ce modèle, nous pouvons voir qu'en fonctionnement normal, les événements de la sortie O_k du voteur servent à valider ceux de la sortie O_i^k. En cas de faute de la ligne fonctionnelle de la carte, la redondance due aux deux autres cartes permet d'avoir un événement observable qui s'il survient avant celui de O_i^k, entraine des transitions comme $S_6 \xrightarrow{x \geq 32 \,; rO_k} S_7$ qui permet d'observer un événement produit par la carte après l'occurrence d'une faute.

Le modèle de la Figure V-24 réussi le test de la diagnosticabilité pour k=131. Nous pouvons construire par déterminisation le diagnostiqueur donné par la Figure V-25. Ce modèle est obtenu en effectuant le produit synchrone du modèle de la carte DC_i (Figure V-21) avec le modèle du voteur du point de vue de cette carte (Figure V-23). Par soucis de simplicité, nous ne conservons que l'horloge x en réécrivant dans le modèle résultant les invariants et les gardes relatives à y_i. Sur cette figure, nous pouvons constater que cette fois-ci le diagnostiqueur possède un état qui permet de détecter une défaillance de type F1. Comme nous pouvions l'imaginer, l'occurrence de rO_k avant rO_i^k fait transiter le diagnostiqueur dans l'état 7F1 qui correspond au diagnostic d'une faute de type F1 ($d^0O_i^k$). De même lors du passage de l'état haut à l'état bas, nous pouvons diagnostiquer une faute de type F2 ($d^1O_i^k$).

Conclusion du chapitre V

Dans ce chapitre, nous avons cherché à montrer comment nous pouvions appliquer les concepts de diagnosticabilité et diagnostiqueur au diagnostic des systèmes embarqués logiques. Pour cela, nous avons dans un premier temps proposé une méthode d'ingénierie pour la construction systématique des modèles temporisés des systèmes à diagnostiquer. Les

principes de cette méthode sont la décomposition structurelle, l'abstraction de comportement par réduction, l'abstraction temporelle et la transformation de modèles.

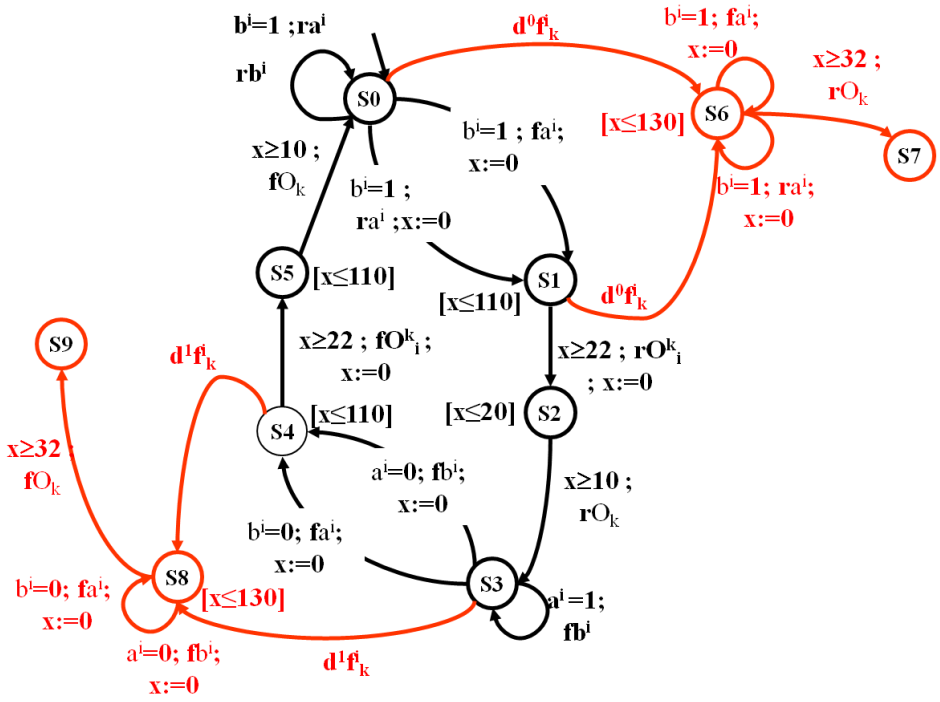


Figure V-24 – Modèle comportemental complet d'une ligne fonctionnelle intégrant la carte fille et le voteur

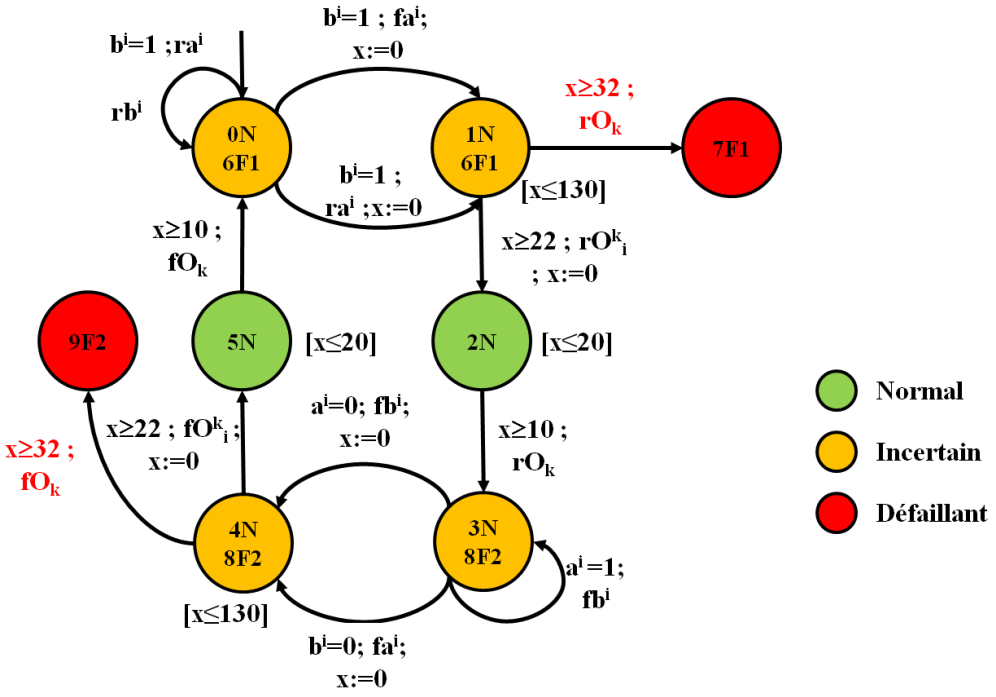


Figure V-25 – Diagnostiqueur du modèle de la Figure V-24

Ensuite, nous avons montré qu'il est possible de modéliser ces systèmes pour permettre l'identification des composants en fautes. Mais, nous avons rapidement été confrontés à l'explosion combinatoire des modèles générés par composition synchrones.

Pour éviter cette explosion combinatoire, nous avons proposé deux solutions : restreindre le diagnostic à la localisation et adapter la diagnosticabilité modulaire au diagnostic des systèmes temporisés.

La diagnosticabilité modulaire nous permet en plus d'envisager une mise en œuvre distribuée du diagnostic. En effet, elle permet de ne pas construire un diagnostiqueur monolithique pour l'ensemble du système, ce qui poserait de nouveau un problème d'explosion combinatoire vu le nombre de composants et d'entrées/sorties. Elle laisse la possibilité au concepteur de construire un diagnostiqueur par ligne fonctionnelle à surveiller.

Mais la principale limite de cette approche est liée à l'hypothèse forte faite sur le fonctionnement de certains composants. Par exemple, dans le cas de la carte FerroCOTS, nous avons supposé que les voteurs calculent leur sortie après avoir reçu toutes leurs entrées. Et donc, en cela, ce sont des systèmes à retards qui attendent la réception de toutes les sorties des cartes filles pour les traiter et produire leur propre sortie. Mais les voteurs simples se contentent de mettre en œuvre un « ou » logique de « et » de leurs entrées prises deux à deux. De ce fait dès que deux entrées sont en accord, le voteur simple génère directement sa sortie sans attendre la 3ème entrée. Avec des voteurs simples, notre approche diagnostiquerait systématiquement en panne la carte fille retardaire.

VI Signatures Temporelles Causales pour le diagnostic des SEL

Introduction du chapitre VI

Dans le chapitre précédent, nous avons vu que même si elle est très intéressante sur le plan théorique, la technique du diagnostiqueur pose encore des problèmes d'application notamment si nous voulons identifier les causes des fautes d'un système. C'est la raison pour laquelle, nous avons cherché à utiliser les STC pour le diagnostic des systèmes embarqués logiques.

Le diagnostic à l'aide des STC consiste à interpréter en ligne l'occurrence d'événements afin d'instancier les motifs qui seront reconnus. Une STC est reconnue lorsque tous ses événements se produisent en respectant leurs contraintes. Dans le chapitre II, nous avons défini et présenté les STC, leur syntaxe générale et l'expression des contraintes temporelles. Rappelons qu'une des limites de l'approche STC, c'est la difficulté dans l'interprétation des événements en entrée du système de diagnostic dans le cadre de l'hypothèse de défaillances multiples. Le problème posé dans ce cadre étant la possibilité qu'une séquence d'événements produits par plusieurs défaillances entraine un surdiagnostic, c'est à dire l'isolation d'une défaillance supplémentaire. En effet, la connaissance codée est généralement une connaissance experte. Dans le meilleur des cas, elle peut être dérivée à partir de connaissances comme les arbres de défaillance (Guerraz, et al., 2004). Toutefois, le problème est que chaque STC ou chronique est construite séparément sans tenir compte des autres règles. Aussi, il demeure nécessaire de vérifier la cohérence d'un ensemble de règles pour assurer la complétude et le déterminisme dans l'hypothèse de défaillances multiples.

Dans ce chapitre, nous présentons trois contributions pour garantir la correction de l'interprétation d'une base de STC.

Tout d'abord, nous expliquons à travers un exemple la problématique de la non cohérence d'une base de STC et les risques de surdiagnostic qui en découle lorsque nous utilisons une technique d'interprétation comme celle des automates dupliqués (Bertrand, 2009). Ensuite, notre première contribution consiste en l'exploitation des RdP T-temporels pour vérifier la cohérence d'une base de STC (Saddem, et al., 2010). Par la suite, dans un souci de cohérence avec le formalisme utilisé dans l'approche diagnostiqueur, la deuxième contribution est la proposition d'une méthode basée sur la technique du model-checking sur des automates temporisés pour vérifier la cohérence d'une base de STC (Saddem, et al., 2011c). Nous

terminons par notre troisième contribution où nous proposons un algorithme d'interprétation basé sur le concept de monde qui garantit la correction du diagnostic (Saddem, et al., 2012).

VI.I Problématiques : cohérence d'une base de STC vs correction de l'algorithme d'interprétation

Soit l'exemple suivant d'une base de STC formée par trois STC :

STC1 : (In, S1, [0,+∞[)*(S1, S2, [1,3])*(S2, S3,[1,2]) => F1
STC2 : (In, S2, [0,+∞[)*(S2, S1, [1,3]) => F2 **(VI-1)**
STC3 : (In, S3, [0,+∞[)*(S3, S1, [1,2]) => F3

Pour comprendre comment s'effectue la reconnaissance des STC, nous introduisons deux concepts : le motif d'une STC et l'instance d'une STC. Un motif de STC est une règle écrite et vérifiée hors ligne de manière à caractériser un fonctionnement d'un système à l'aide d'événements et d'un ensemble de contraintes cohérentes entre ces événements. Une instance de STC est la reconnaissance en ligne d'un motif de STC.

L'algorithme proposé ici est similaire à la technique de l'automate dupliqué (Bertrand, 2009). Il est basé sur les règles suivantes :

Règle VI-1

Si un événement est le premier événement d'un motif de STC, alors créer une instance de cette STC et affecter l'événement à cette instance.

Règle VI-2

Si un événement est attendu dans une instance de STC, alors dupliquer cette instance et l'affecter à l'instance initiale.

La Règle VI-2 permet de tenir compte du fait que l'affectation de l'événement courant peut être une hypothèse erronée qui ne sera pas validée dans le futur. Donc, chaque fois qu'un événement survient, nous effectuons deux hypothèses, l'affecter à l'instance et attendre une nouvelle occurrence de cet événement.

Règle VI-3

Si un événement est affecté à une instance de STC, alors mettre à jour les fenêtres du graphe de contraintes temporelles pour déterminer le prochain événement attendu et la deadline de la STC. La deadline d'une STC étant le délai maximal d'attente d'un événement.

Règle VI-4

Si un événement est attendu dans une instance de STC et ne survient pas avant la deadline, alors supprimer cette instance.

La Règle VI-4 permet de supprimer les instances de STC pour lesquelles nous avons fait des hypothèses d'affectation incohérentes.

Règle VI-5

Si tous les événements d'une STC ont été reconnus, alors intégrer les conséquences de cette STC dans l'ensemble des causes reconnues.

Afin d'illustrer cet algorithme, nous considérons l'ensemble de STC donné par l'équation (VI-1).

Supposons que les symptômes détectés au fil du temps soient la séquence temporelle (S1, 0) * (S2, 2) * (S2, 3) * (S3, 4) * (S1, 5). Notons que dans cette séquence d'entrée, dans le doublet (Si, Ti), « Si » modélise un événement et « Ti » une date absolue. Afin de simplifier l'illustration de cet algorithme de reconnaissance, nous prenons comme référence du temps absolu la date de détection du premier événement (instant initial t0 = 0).

La Figure VI-1 illustre l'application de cet algorithme de reconnaissance à l'exemple donné par l'équation (VI-1). Dans cette figure, nous utilisons la notation suivante « STCi-j (E ; [Sc] ; TOi) ».

- 'STCi-j' est la $j^{éme}$ instance de STCi,
- 'E' est l'ensemble ordonné des événements produits suivant leur ordre d'occurrence (soient 'Sa' le l^{er} événement suivi de 'Sb', E = Sa, Sb),
- [Sc] est le prochain événement attendu,
- TOi est la deadline de STCi-j.

A $t_0 = 0$, S1 se produit. Il ne peut être affecté qu'à STC1. D'après la Règle VI-1, nous créons une instance de STC1 que nous appelons STC1-1 et nous lui affectons S1. D'où dans la Figure VI-1 STC1-1(S1, [S2], 3). S2 est attendu au plus tard après 3 ut (pour valider le triplet (S1, S2, [1,3]) dans STC1).

A $t_1 = 2$ ut, S2 se produit. S2 peut être affecté soit à :

- A l'instance STC1-1 (comme nous attendons un S2). Donc, d'après la Règle VI-2, nous dupliquons STC1-1 en STC1-1 et STC1-2.
    - Nous affectons S2 à STC1-1 et nous mettons à jour cette dernière qui devient STC1-1(S1, S2 ; [S3] ; 2). (Nous attendons l'occurrence de S3 dans un délai de 2 ut (pour valider le triplet (S2, S3, [1,2]) dans STC1)).
    - Pour STC1-2, nous attendons encore 1 ut pour l'occurrence d'un S2.

- Au motif STC2. D'après la Règle VI-1, nous créons une instance de STC2 que nous appelons STC2-1 et nous lui affectons S2. Nous attendons S1 dans un délai de 3 ut (pour valider le triplet (S2, S1, [1,3]) dans STC2)

De la même manière, et pour chaque événement produit, nous appliquons les règles de l'algorithme.

L'algorithme semble bien fonctionner dans la mesure où tous les symptômes sont interprétés au fil du temps. Nous aboutissons au diagnostic de l'ensemble de causes {F1, F2, F3}. Mais dans la réalité, les symptômes en entrée du diagnostic ont été produits par deux causes : la cause F1 a produit les symptômes (S1, 0)*(S2, 2)*(S3, 3) et la cause F2 a produit les symptômes (S2, 3)*(S1, 5). Le diagnostic de F3 correspond donc à une situation de surdiagnostic. Ce problème est dû à un problème de cohérence de la base ou à un problème de correction de l'algorithme d'interprétation. Dans les sections suivantes, nous proposons des solutions à ces deux problèmes.

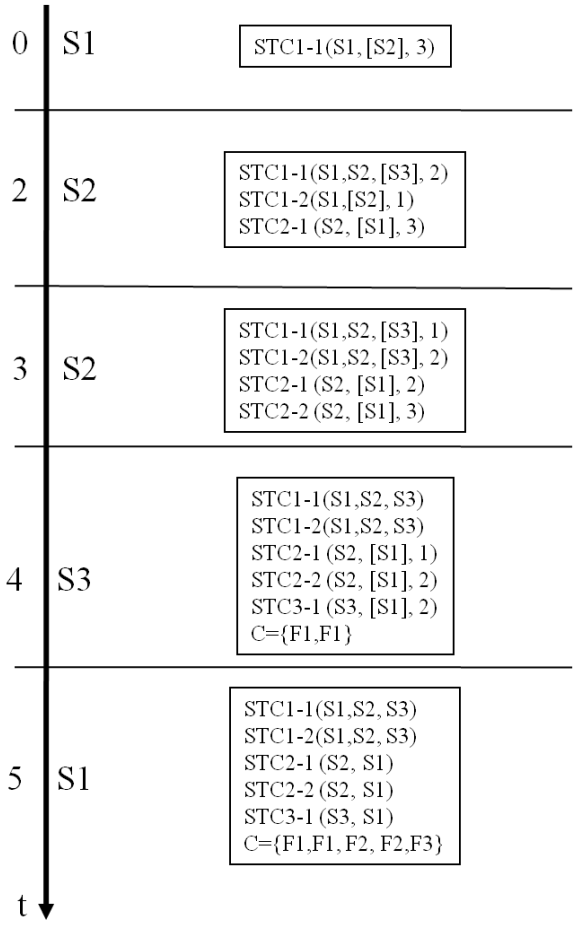


Figure VI-1 – Illustration de l'algorithme basé sur la technique de l'automate dupliqué

VI.II Méthode pour le contrôle de la cohérence d'une base de STC à l'aide des RdP T-temporels

Pour vérifier la cohérence d'une base de STC, nous proposons dans cette partie une réécriture des STC dans le formalisme des RdP T-temporels (§II.I.4.A). Nous rappelons que dans le §II.II.4.B, nous avons dit que pour exprimer les contraintes temporelles dans une STC, nous avons retenu les structures d'intervalles car elles sont conformes à la modélisation temporelle du comportement du procédé pour la détection (Toguyeni, et al., 1997). Une structure d'intervalle nous permet de définir trois types de contraintes temporelles : la période, la date et la durée. Dans la suite, nous proposons une réécriture de STC en un modèle RdP T-temporel (Saddem, et al., 2010).

VI.II.1 Réécriture d'un triplet à contrainte temporelle de type période

Nous proposons ici une modélisation d'un triplet correspondant à une contrainte temporelle de type période entre deux événements. Soit l'équation (VI-2), où 'A' et 'B' sont deux événements et [α, β] est une contrainte temporelle de type période.

(A,B, [α, β]) **(VI-2)**

La Figure VI-2 donne le modèle RdP T-temporel en sémantique forte du triplet de l'équation (VI-2). La place NOP (no operation) modélise que le triplet n'a pas encore été interprété.

Dans ce modèle, nous distinguons deux types de transitions. Les transitions associées à des événements comme la transition nommée 'A' et la transition nommée 'B'. Ce sont des transitions à temporisation $[0, +\infty[$. Elles n'ont donc pas de contraintes temporelles. Leur franchissement modélise l'occurrence de l'événement associé. Le deuxième type de transitions correspond à celles qui modélisent les bornes de la période. La transition $Tmin$ modélise la borne inférieure de la période et la transition $Tmax$ la borne supérieure.

L'occurrence de l'événement 'A' entraine le marquage des places Max, Min_b et A_B. Cela déclenche notamment l'horloge relative à la transition $Tmin$. Au bout de α ut, $Tmin$ est franchie en raison de la sémantique forte. La place Min_a est alors marquée sensibilisant la transition nommée 'B' et la transition $Tmax$. Si l'événement 'B' survient avant le franchissement de $Tmax$, alors la transition correspondante est franchie. Dans ce cas, la place $Succès$ est marquée indiquant la vérification de la relation donnée par l'équation (VI-2). Sinon, le franchissement de $Tmax$ indique un échec de le la reconnaissance. Dans ce modèle, nous rebouclons sur la place NOP ce qui permet de vérifier que le modèle est vivant et borné par construction du graphe des classes.

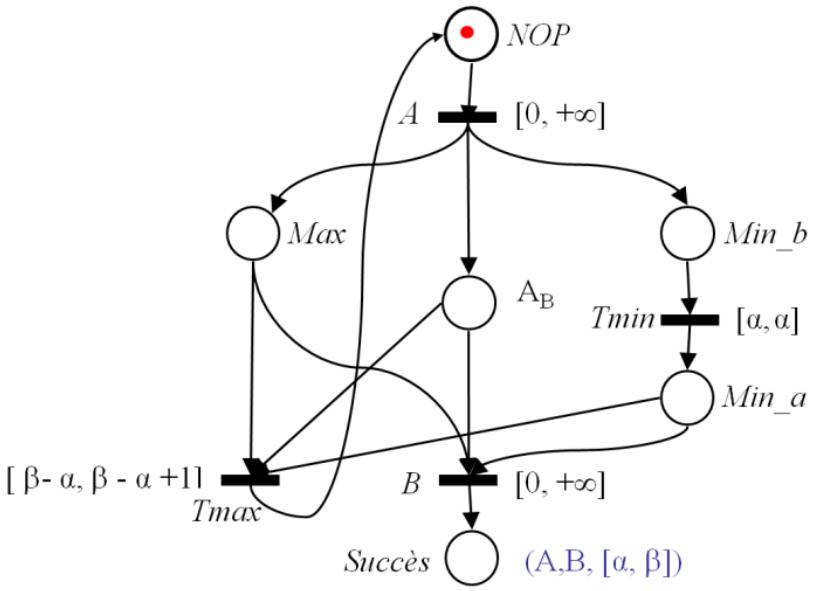


Figure VI-2 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une période

VI.II.2 Réécriture d'un triplet à contrainte temporelle de type date

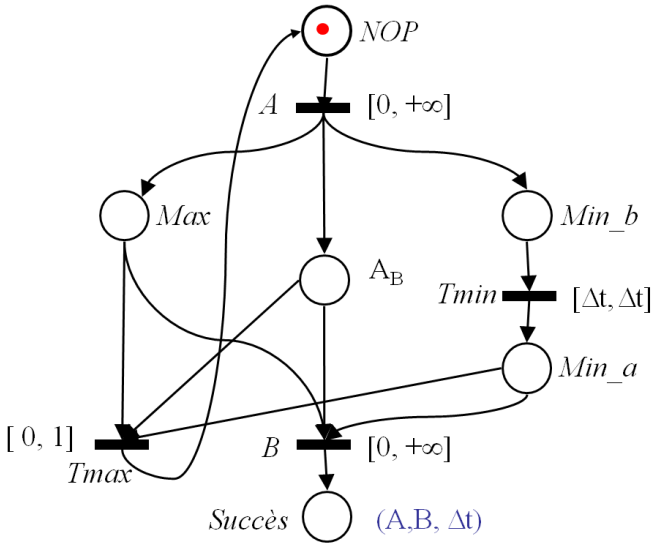


Figure VI-3 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une date

Le cas de la date se déduit directement de la période. Une date Δt se ramène à une période de borne inférieure et supérieure égales ($p = [\Delta t, \Delta t]$). Soit l'équation (VI-3), où 'A' et 'B' sont deux événements et Δt est une contrainte temporelle de type date.

(A,B, Δt) **(VI-3)**

La Figure VI-3 donne le modèle RdP T-temporel en sémantique forte du triplet de l'équation (VI-3).

VI.II.3 Réécriture d'un triplet à contrainte temporelle de type durée

(C, not D, [δ, λ]) **(VI-4)**

L'équation (VI-4) donne un cas d'utilisation d'une durée. Le triplet correspondant est vérifié si l'évènement 'D' ne survient pas de δ ut à λ ut après l'occurrence de 'C'. La sémantique de ce triplet est donnée par la Figure VI-4. Dans ce modèle, seules les transitions 'T1' et 'T2' sont commentées. Les autres places et transitions sont similaires à celles vues dans la Figure VI-2. La transition 'T1' modélise l'occurrence de l'événement 'D' dans les bornes de la contrainte de type période. C'est donc un échec puisque dans le triplet (VI-4), l'événement 'D' est nié. Le franchissement de 'T2' par contre, modélise un succès dans la mesure où cette transition modélise une situation d'occurrence de 'D' avant la date 'δ'. Sinon, s'il n'y a pas d'occurrence de 'D', il y a franchissement de $Tmax$ ce qui correspond à un succès.

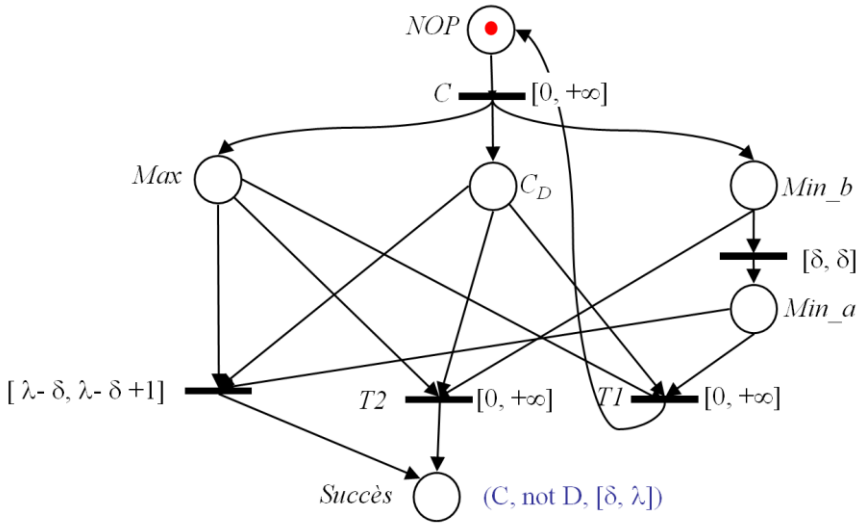


Figure VI-4 – Modèle RdP T-temporel d'un triplet dont la contrainte temporelle est une durée

VI.II.4 Règles de réécriture de STC

Nous considérons les STC de la forme donnée par l'équation (VI-5). Dans cette STC, les mêmes événements apparaissent plusieurs fois. L'objectif à présent est de définir des règles mettant en évidence comment ces événements doivent être interprétés pour traduire la STC en un modèle RdP T-temporel.

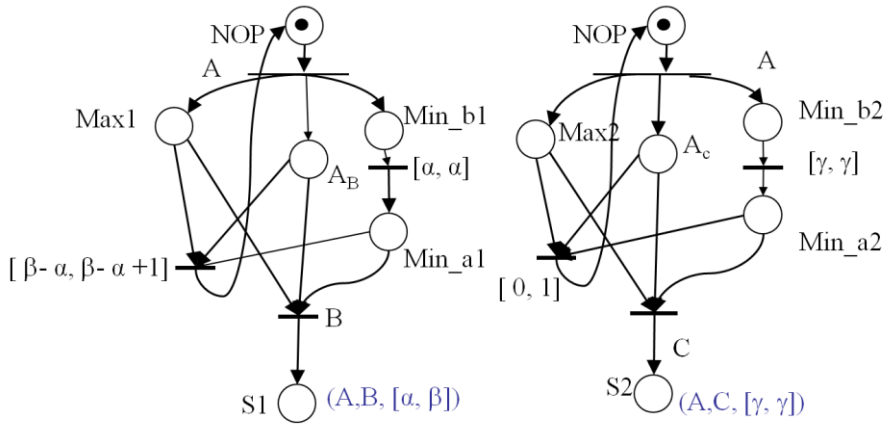
(A,B, [α, β])* (A,C, [γ, δ])*(C, not D, [δ, λ]) -> G **(VI-5)**

Règle VI-6 : Fusion de 2 occurrences d'un événement référent

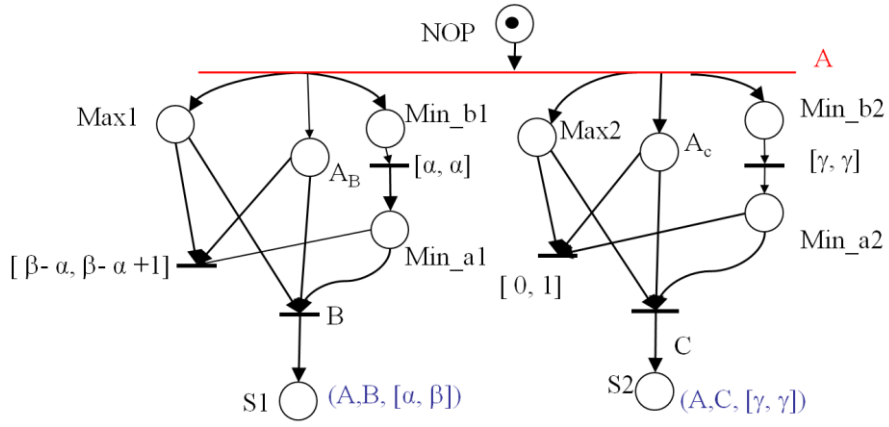
Lorsqu'un événement référent apparait dans plusieurs triplets, il doit être modélisé dans le modèle RdP T-Temporel, par une transition unique.

C'est le cas de l'événement 'A' dans les deux premiers triplets. La Figure VI-5 montre un exemple de l'application de la

Règle VI-6 sur la STC de l'équation (VI-5). La transition 'A' (en rouge) est le résultat de la fusion des deux transitions nommées 'A' dans les modèles des deux premiers triplets.

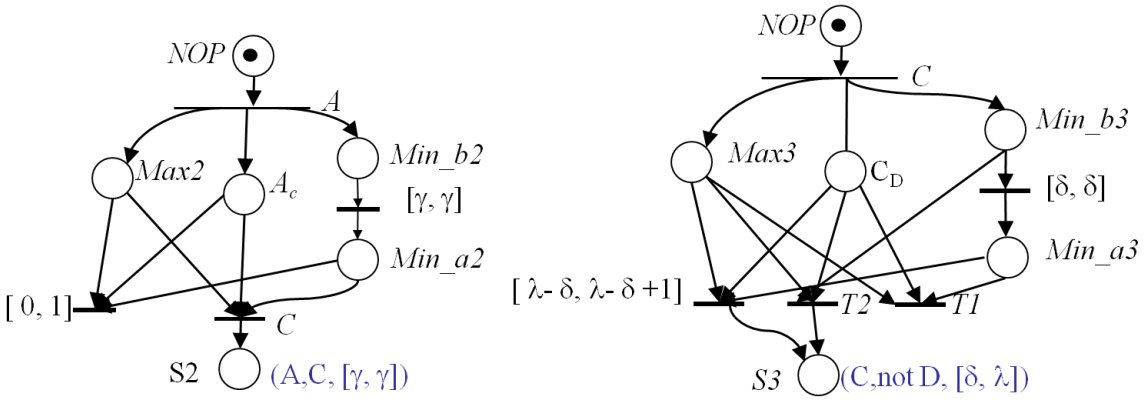


a)

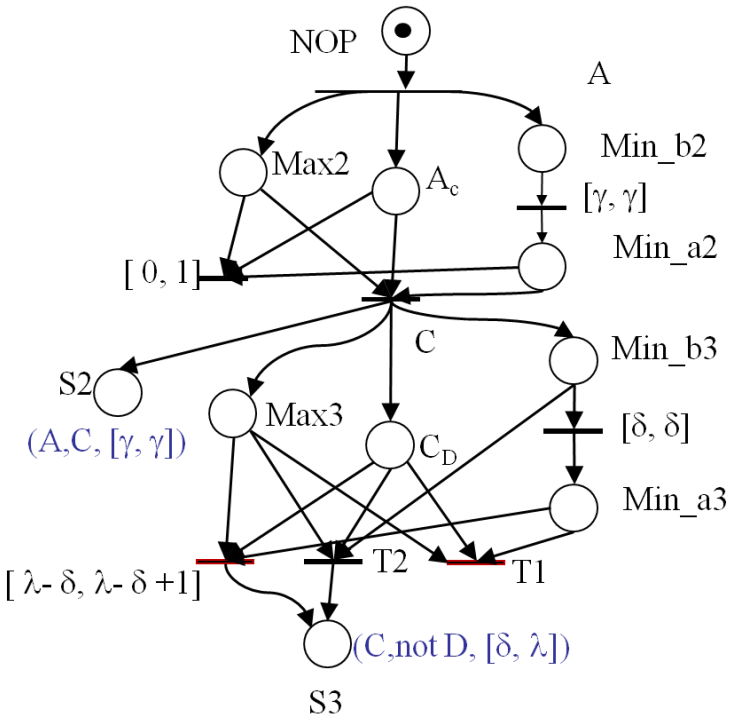


b)

Figure VI-5 – Exemple de fusion de 2 occurrences d'un événement référent



a)



b)

Figure VI-6 – Exemple de fusion de l'occurrence d'un événement référent et d'un événement contraint

Règle VI-7 : Fusion de l'occurrence d'un événement référent et d'un événement contraint

Lorsqu'un événement apparaît en tant qu'événement référent et en tant qu'événement contraint dans une même STC, les triplets correspondants doivent être mis en séquence par la fusion des événements en une transition unique.

C'est le cas de l'évènement 'C' dans les deux derniers triplets de la STC de l'équation (VI-5). La Figure VI-6 montre un exemple d'application de la Règle VI-7 sur la STC de l'équation (VI-5).

Nous spécifions également des règles pour la gestion de l'échec et du succès lors de la reconnaissance d'une STC.

Règle VI-8 : Gestion des échecs

En cas d'échec, nous supprimons les jetons de reconnaissance du triplet par l'utilisation d'une transition puits.

Règle VI-9 : Reconnaissance d'une STC

Nous associons une place à chaque succès de reconnaissance d'un triplet. Quand tous les triplets dans la partie prémisse ont été reconnus, nous déduisons les conséquences par la validation d'une transition unique.

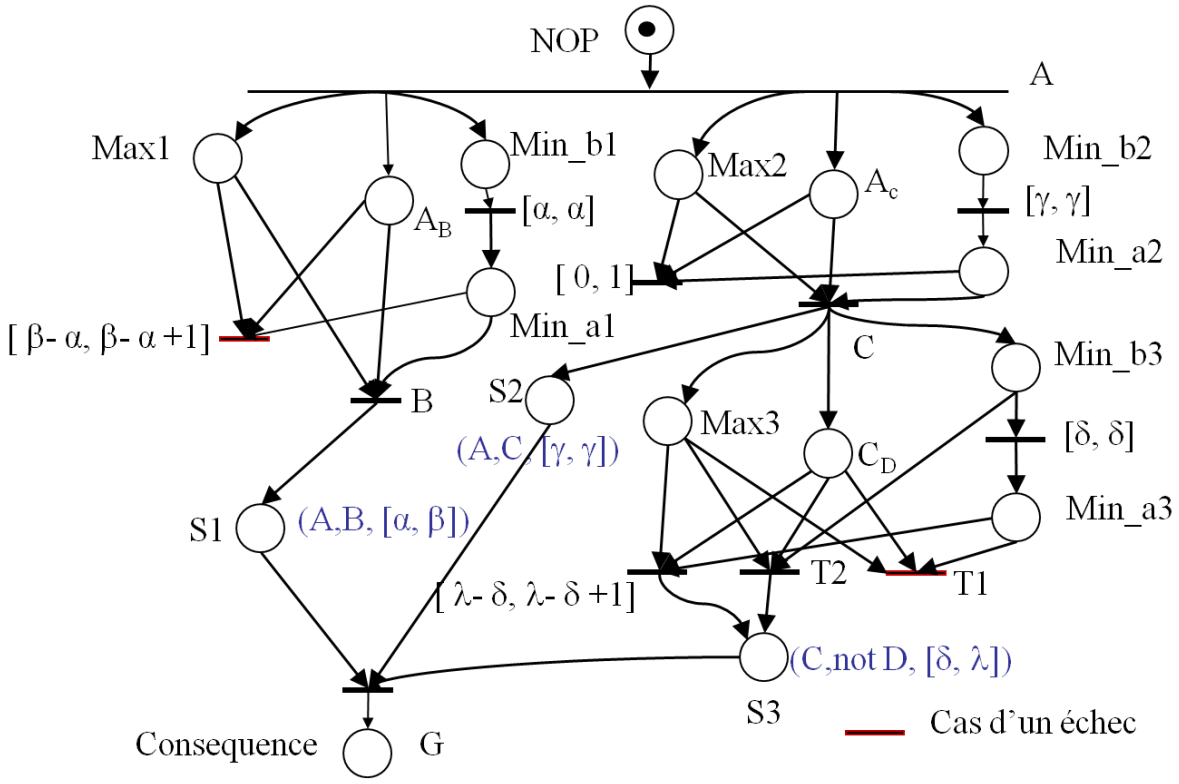


Figure VI-7 – Application des règles de construction sur la STC de l'équation (VI-5)

VI.II.5 Cas d'étude en productique

Nous considérons pour cette étude de cas, l'exemple donné par une partie d'un convoyeur à bande continu illustré par la Figure VI-8. Ce type de convoyeur comprend une bande qui est actionnée en continu par un moteur à courant continu. Dans le cadre de notre étude, le convoyeur comprend quatre postes de chargement/déchargement de produits notés 'Pi' avec i ∈ [1..4]. Chaque poste est structuré de manière identique. Il comprend un capteur d'entrée

noté ‘pie’, un capteur de sortie noté ‘pis’ et une butée notée ‘Bi’, avec i le numéro du poste. Par souci de simplicité, les postes sont tous supposés de capacité unitaire. Deux autres capteurs sont situés sur le parcours : p1vp2 pour détecter le passage d’une pièce lors du transfert de P1 vers P2 et p1vp4 pour détecter le passage d’une pièce lors du transfert de P1 vers P4.

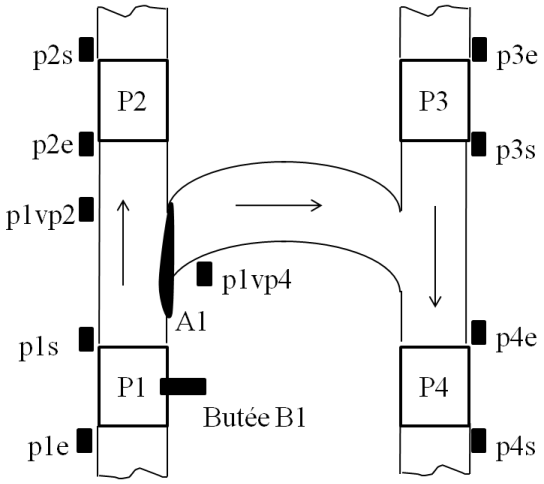


Figure VI-8 – convoyeur à bande

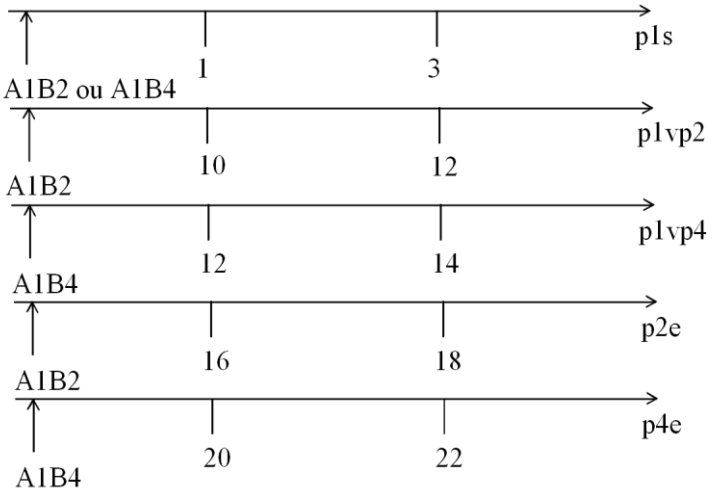


Figure VI-9 – Abstraction du comportement normal d’une partie du convoyeur

Par souci de simplicité, nous nous limitons dans cet exemple, au diagnostic des défaillances de trois composants : le capteur ‘p1s’, l’aiguillage ‘A1’, et la butée ‘B1’. La Figure VI-9 donne l'abstraction du comportement temporel du système pour le transfert d'un produit de P1 vers P2 (correspond à l’ordre ‘A1B2’ ») ou de P1 vers P4 (correspond à l’ordre ‘A1B4’). Par exemple, le premier axe temporel indique que si la partie commande émet l’ordre ‘A1B2’ou ‘A1B4’, le capteur ‘p1s’ doit se déclencher entre 1 et 3 ut en raison de la détection de passage de la pièce transférée.

En cas de comportement anormal d’un capteur « cpi », il peut détecter soit un symptôme de type I (S^1_{cpi}) qui signifie que le capteur ne s’est pas déclenché dans le délai prévu ou un

symptôme de type II (S^2_{cpi}) qui, au contraire, signifie un déclenchement intempestif (Toguyeni, et al., 1990).

Nous supposons qu'un module de détection est placé en filtre entre la partie opérative et la partie commande (Figure VI-10).

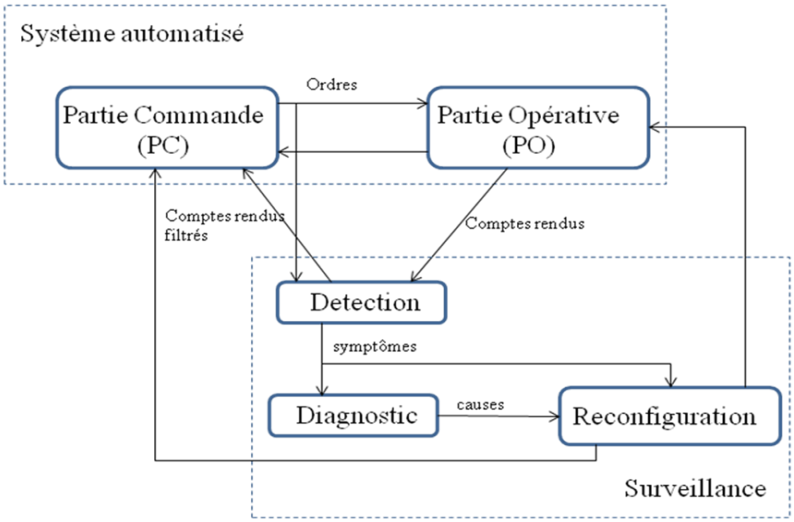


Figure VI-10 – Structure du système de surveillance pour le diagnostic des défaillances

Nous donnons à présent quelques STC relatives à ces défaillances :

STC1: (In, A1B2, [0,+∞[)* (A1B2, S^1_{p1s}, [3,3]) * (A1B2, p1vp2, [10,12]) * (A1B2, p2e, [16,18]) => (collage, p1s, 0)

Le sens de STC1 est le suivant : Si après la commande du transfert d'un produit de 'P1' vers 'P2', le capteur de sortie 'p1s' ne se déclenche pas au bout d'un délai maximum de 3 ut, cela conduit à la détection d'un symptôme de type I (S^1_{p1s}).

En cas de déclenchement des deux autres capteurs situés sur le parcours (respectivement p1vp2 et p2e) dans les délais correspondant au comportement normal (Figure VI-9), nous concluons que la cause de ce symptôme c'est le collage à zéro du capteur 'p1s'. De la même manière, nous écrivons 5 STC supplémentaires :

STC2: (In, A1B4, [0,+∞[) * (A1B4, S^1_{p1s}, [3,3]) * (A1B4, p1vp4, [12,14]) * (A1B4, p4e, [20,22]) => (collage, p1s, 0)

La STC2 est duale de la STC1. Elle sert dans le contexte d'un transfert de 'P1' vers 'P4'.

STC3: (In, A1B2, [0,+∞[) * (A1B2, p1s, [1,3]) * (A1B2, S^1_{p1vp2}, [12,12]) * (A1B2, S^1_{p2e}, [18,18]) => (blocage, A1, P1vP4)

La STC3 sert à diagnostiquer une défaillance de l'aiguillage 'A1' bloqué dans le sens du transfert de 'P1' vers 'P4'. Cette situation peut être détectée lors d'un transfert de 'P1' vers

'P2' quand les capteurs placés le long de la trajectoire ne détectent rien, donnant des symptômes de type I.

STC4: (In, A1B4, [0,+∞[)* (A1B4, p1s, [1,3]) * (A1B4, S^1_{p1vp4}, [14,14]) * (A1B4, S^1_{p4e}, [22,22]) => (blocage, A1, P1vP2)

La STC4 est duale de la STC3.

STC5: (In, A1B2, [0,+∞[)* (A1B2, S^1_{p1s}, [3,3]) * (A1B2, S^1_{p1vp2}, [12,12])) * (A1B2, S^1_{p2e}, [18,18]) => (blocage, B1, sortie)

La STC5 sert à diagnostiquer la défaillance de la butée 'B1', bloquée en position sortie, dans le cas d'un transfert de 'P1' vers 'P2'. Dans ce cas, tout au long de la trajectoire, aucun capteur ne détecte le passage de pièce ce qui conduit à la génération en séquence de 3 symptôme de type I.

STC6: (In, A1B4,[0,+∞[)* (A1B4, S^1_{p1s}, [3,3]) * (A1B4, S^1_{p1vp4}, [14,14])) * (A1B4, S^1_{p4e}, [22,22]) => (blocage, B1, sortie)

STC6 est duale de STC5 pour le contexte d'un transfert de 'P1' vers 'P4'.

En fait, les 6 règles peuvent être groupées en deux groupes de règles : le groupe des règles commençant par l'événement 'A1B2' (STC1, STC3, STC5) et le groupe des règles commençant par l'événement 'A1B4'. Puisque 'A1B2' et 'A1B4' sont deux ordres opposés, les deux classes de règles ne peuvent pas être interprétées simultanément. Par conséquent, la vérification de la cohérence peut être effectuée classe par classe. Ici, nous illustrons notre méthode avec la classe initiée par 'A1B2'.

La transcription de STC1, STC3 et STC5 en RdP T-temporels et l'application des règles du §VI.II.4 nous donne le modèle RdP de la Figure VI-11. Nous avons utilisé l'outil ROMEO (Gardey, et al., 2005) pour construire ce modèle. En analysant le modèle résultant, nous concluons que cet ensemble de STC n'est pas cohérent car il peut nous conduire à des faux diagnostics. Par exemple, le collage à zéro de 'p1s' et le blocage de 'A1' en position transfert vers 'P4' entraîne l'obtention de la séquence d'entrée S^1_{p1s} * S^1_{p1vp2} * S^1_{p2e}. Cette séquence conduit à diagnostiquer le blocage de 'B1' en position sortie, au lieu de diagnostiquer les défaillances de 'p1s' et de 'A1'. Cela s'explique par le fait que les STC ont été implicitement écrites dans le cadre de l'hypothèse de défaillance unique. En effet, le triplet (A1B2, p1vp2, [10,12]), écrit en supposant les autres capteurs opérationnels, ne permet pas de reconnaître la STC1. De même le triplet (A1B2, p1s, [1,3]) ne permet pas de reconnaitre la STC3.

Cet exemple montre bien l'intérêt de la recherche de cohérence. En effet, cela conduit à reconsidérer la base de STC et à les réécrire de manière à les rendre interprétable quelque soit la situation. Prenons l'exemple de la STC3. Cette règle peut être réécrite en supprimant le premier triplet qui est très spécifique à un contexte de fonctionnement du système et en

caractérisant la défaillance à la fois avec les symptômes de type I et les symptômes de type II produits par la défaillance. Soit la nouvelle STC3 (NEWSTC3) donnée par l'équation (VI-6).

NEWSTC3 : (A1B2, S^1_{p1vp2}, [12,12]) * (A1B2, S^2_{p1vP4}, [12,14]) * (A1B2, S^1_{p2e}, [18,18]) * (A1B2, S^2_{p4e}, [20,22]) -> (blocage, A1, P1vP4) **(VI-6)**

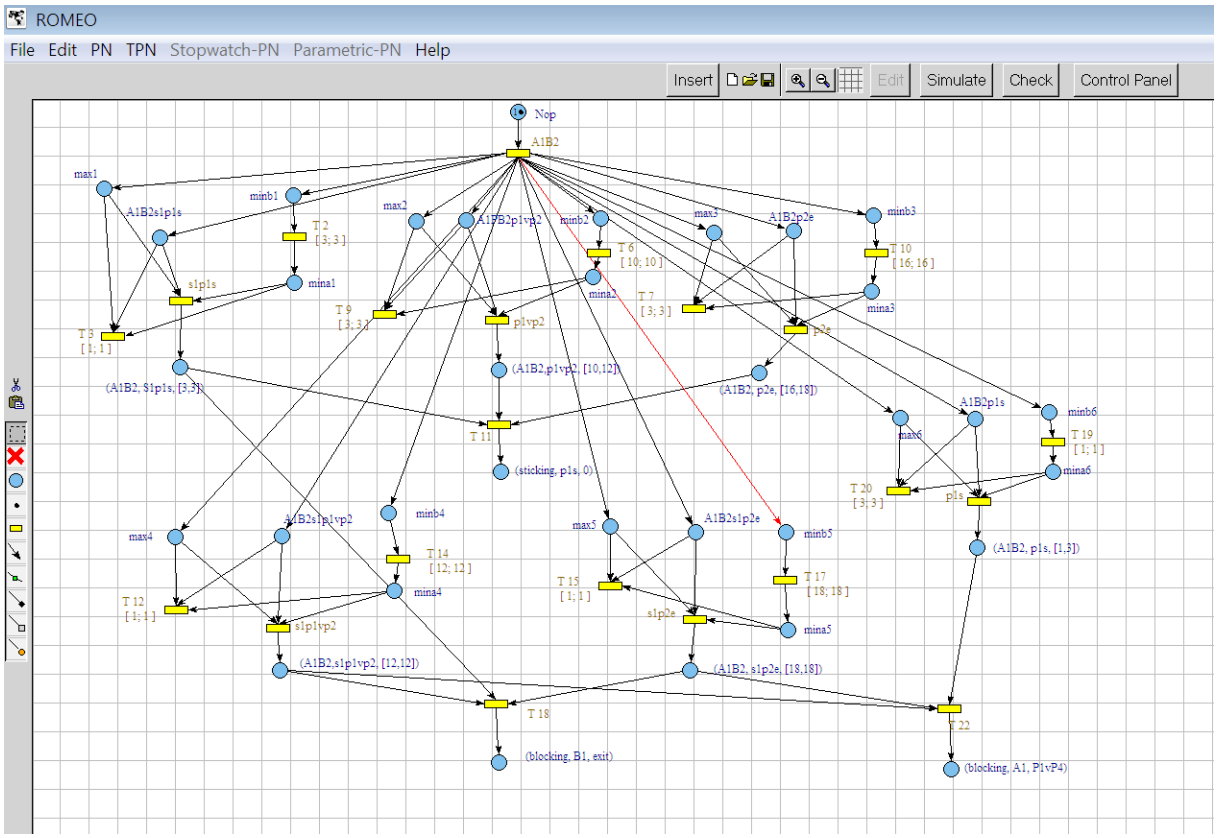


Figure VI-11 – Evaluation avec ROMEO de l'ensemble de STC1, STC3 et STC5

VI.III Méthode pour la vérification de la cohérence d'une base de STC par le model-checking

VI.III.1 Présentation de la méthode

Nous proposons une méthode pour vérifier la cohérence d'un ensemble de STC basée sur le model-checking. La présentation des STC faite dans le §II.II.4 montre qu'elles correspondent à un système de transitions temporisées. Nous choisissons d'utiliser dans cette partie, le formalisme des automates temporisés pour formaliser la modélisation de STC.

Nous avons choisi l'outil KRONOS §IV.V.3 pour son aptitude à extraire des séquences satisfaisant une propriété. En effet, pour vérifier la cohérence d'un ensemble de STC, l'idée consiste à vérifier s'il existe des séquences temporisées en entrée du diagnostic conduisant à des situations de surdiagnostic. Pour identifier ces situations, il suffit de vérifier si certaines séquences de symptômes en entrée pourraient participer à l'interprétation de plusieurs STC. Il

suffit pour cela de rechercher dans le produit synchrone des automates temporisés écrits avec les différentes règles, les séquences d'entrée conduisant à la reconnaissance de plusieurs STC. S'il y a moins de symptômes dans une séquence d'entrée que la somme des nombres de symptômes caractérisant chaque STC considérée, alors on est en présence d'un surdiagnostic.

La méthode de vérification comprend 4 étapes.

Etape VI-1 :

Elle consiste à coder chaque STC dans le formalisme des automates temporisés.

Etape VI-2 :

Elle consiste à construire un automate temporisé synthèse de tous les automates qui formalisent les différentes STC. Cet automate est construit par le produit synchrone des automates de STC synchronisés sur leurs événements communs. En effet, dans §VI.I, nous avons vu qu'une des causes du surdiagnostic est que le même événement est affecté à plusieurs STC. Pour construire l'automate de synthèse, nous pouvons utiliser sous KRONOS la commande suivante (VI-7) :

« kronos -out synthese.tg stci.tg stcj.tg » **(VI-7)**

Avec « synthese.tg » l'automate résultant, et « stci.tg » et « stcj.tg » correspondant respectivement aux automates de STCi et STCj. Pour obtenir le produit synchrone par rapport aux événements communs, nous énumérons dans chaque fichier « stci.tg » les événements qui sont synchronisés (Utiliser le champ « #sync» dans les fichiers « stci.tg »).

Etape VI-3 :

Elle consiste à extraire toutes les séquences temporisées qui permettent d'atteindre les états qui correspondent à plusieurs défaillances. La commande suivante de KRONOS donne ces séquences :

« kronos –allpaths –FULLDFS –forw synthese.tg defaillances.tctl » **(VI-8)**

Avec « defaillances.tctl » définissant un état atteint correspondant à des défaillances multiples. En fait, ici nous utilisons les capacités d'analyse d'accessibilité d'états de KRONOS.

Etape VI-4 :

Elle consiste à analyser les séquences obtenues pour voir si elles correspondent ou pas à un cas de surdiagnostic. Il y a surdiagnostic quand un symptôme est affecté à plusieurs défaillances.

L'application de cette analyse à l'exemple des STC proposées par l'équation (VI-1) nous permet de constater qu'une séquence temporisée telle que (S1 ; 0) * (S3 ; 2) * (S2 ; 3) * (S1 ; 4) * (S3 ; 5) permet de diagnostiquer les 3 défaillances sans aucune violation de contraintes.

Ce résultat n'est pas correct, car il est nécessaire de détecter à des dates différentes 3 symptômes « S1 », 2 symptômes « S2 » et 2 symptômes « S3 » (donc, au total, 7 symptômes). La séquence temporisée considérée ici a seulement 5 symptômes. Cela signifie par exemple que le dernier symptôme « S1 » est utilisé pour caractériser à la fois la défaillance F2 et la défaillance F3. Donc la base de STC considérée n'est pas cohérente.

VI.III.2 Application au diagnostic des défaillances d'un convoyeur à bande

Nous reprenons le même exemple du convoyeur à bande de la Figure VI-8.

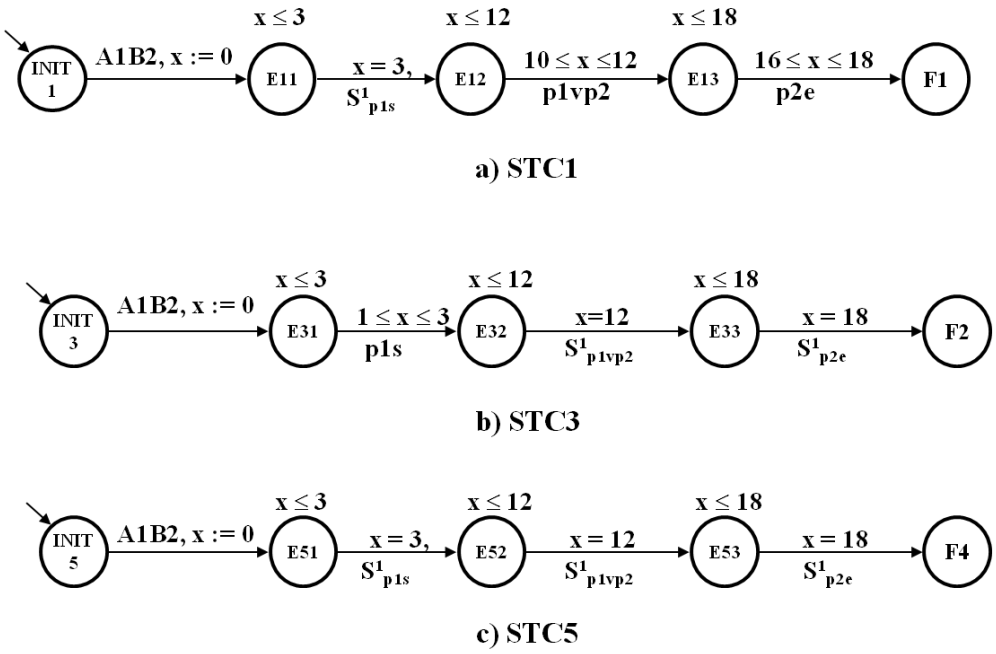


Figure VI-12 – Réécriture des STC en AT

La Figure VI-12 illustre la réécriture de l'automate temporisée correspondant. Nous effectuons d'abord, le produit synchrone de STC3 et STC5 en synchronisant les événements 'A1B2', 'S^1_{p1vp2}' et 'S^1_{p2e}'. Nous effectuons ensuite la composition parallèle de l'automate obtenu avec STC1 en synchronisant les événements 'A1B2' et 'S^1_{p1s}'. L'automate final est représenté par la Figure VI-13.

Comme présenté dans le paragraphe précédent (§VI.III.1), nous pouvons repérer les séquences temporisées qui pourraient conduire à un surdiagnostic avec un algorithme comme celui basé sur la duplication d'automate. A titre d'exemple, la séquence A1B2 * S^1_{p1s} * p1vp2 * p2e * p1s * S^1_{p1vp2} * S^1_{p2e} conduit à diagnostiquer les trois fautes (collage, p1s, 0), (blocage, A1, P1vP4) et (blocage, B1, sortie) alors qu'en réalité seulement les deux premières ont eu lieu. En fait, l'état 14 modélise l'occurrence des trois fautes et nous pouvons chercher toutes les séquences qui atteignent cet état.

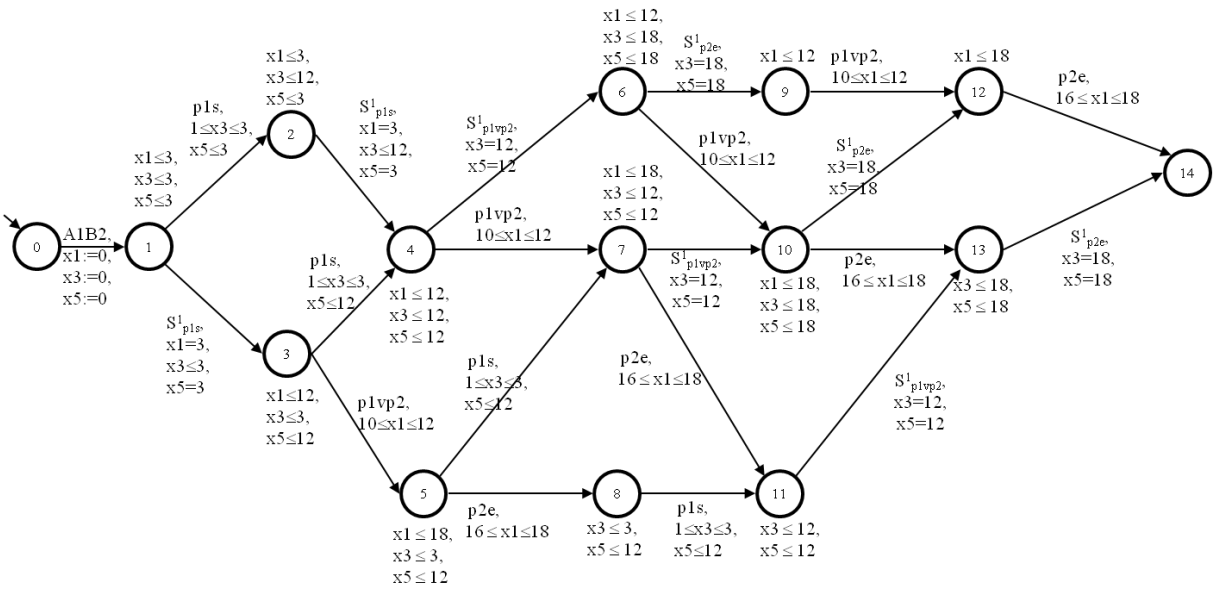


Figure VI-13 – Composition parallèle des AT de STC1, STC3 et STC5

VI.IV Algorithme d'interprétation basé sur la notion de monde

Les algorithmes de reconnaissance des chroniques sont basés sur la technique de duplication d'automates (Bertrand, 2009). L'algorithme de reconnaissance de STC proposé ici est basé sur le concept de monde. Dans notre approche, un monde représente un ensemble d'hypothèses cohérentes d'affectations d'événements reçus par la tâche de diagnostic à une et une seule instance de STC.

En pratique, un monde 'W_i' (avec i l'indice de ce monde) est caractérisé par :

- 'Evt_i' : l'ensemble des événements reçus et affectés,
- 'L_i' : l'ensemble des STC instanciées,
- '$Causes_i$' : l'ensemble des causes inférées,
- 'NE_i' : le prochain événement attendu,
- 'MAX_TO_i' : le timeout donné par 'NE_i' calculé en prenant le minimum des fenêtres temporelles obtenues en propageant la date d'occurrence de chaque événement au précédent compte tenu des contraintes temporelles les reliant à 'NE_i',
- 'MIN_TO_i' : le maximum des minimums des contraintes temporelles propagées à partir des événements précédents.

Notre algorithme est basé sur les règles suivantes :

Règle VI-10 : Hypothèse du monde Fermé

Cette règle signifie que si un événement n'est pas explicitement reçu par la tâche de diagnostic, il est supposé être inexistant (donc la négation est vraie).

Un événement reçu en entrée est supposé vrai jusqu'à ce qu'il ait contribué à la reconnaissance complète d'une STC.

Règle VI-11 : Une seule affectation par symptôme

Si un symptôme survient, il est affecté à une et une seule STC dans un monde.

Cette règle est cohérente avec l'hypothèse classique des systèmes asynchrones qui considère que deux événements externes non corrélées ne peuvent pas être simultanés (David, et al., 1992). En d'autres termes, pour nous, deux défaillances indépendantes ne produisent pas simultanément les mêmes symptômes. Dans l'écriture de nos STC, nous supposons que les défaillances sont indépendantes.

Règle VI-12 : Duplication des mondes

Si plusieurs STC sont candidates pour l'affectation d'un symptôme, dupliquer le monde en cours en autant de mondes que d'hypothèses d'affectation cohérentes.

En fait, chaque possibilité correspond à une hypothèse d'affectation qui est cohérente avec les hypothèses précédentes qui ont été faites auparavant pour les symptômes reçus ayant été affectés à une seule STC dans ce monde. Cette règle est donc une conséquence de la Règle VI-11.

Règle VI-13 : Suppression du monde fondé sur un ensemble d'hypothèses incohérentes

Si un événement n'est affectable à aucune des STC du monde considéré (il n'était pas prévu ou il n'est pas compris entre le MIN_TO_i et le MAX_TO_i), ou si un événement attendu n'est pas survenu à MAX_TO_i, cela signifie que toutes les affectations antérieures sont incohérentes. Alors, le monde considéré est supprimé.

Cette règle est fondée sur l'hypothèse que la base des STC est complète.

Règle VI-14 : Fusion des deux mondes

Soient deux mondes 'W_i' et 'W_j'. Si tous leurs attributs caractéristiques sont égaux ('$Causes_i$' = '$Causes_j$' ; 'L_i' = 'L_j' ; 'NE_i' = 'NE_j' ; 'MIN_TO_i' = 'MIN_TO_j' ; 'MAX_TO_i' = 'MAX_TO_j'), alors garder 'W_i' et supprimer 'W_j'.

Règle VI-15 : Utilisation des causes inférées

Si un seul monde subsiste à une date et l'ensemble de ses causes n'est pas vide, les causes identifiées correspondent au diagnostic. Si 'L_i' est vide, alors réinitialiser tous les attributs du monde.

La Figure VI-14 représente un exemple d'arbre de résolution. Cet exemple est basé sur l'ensemble de STC données par l'équation (VI-1).

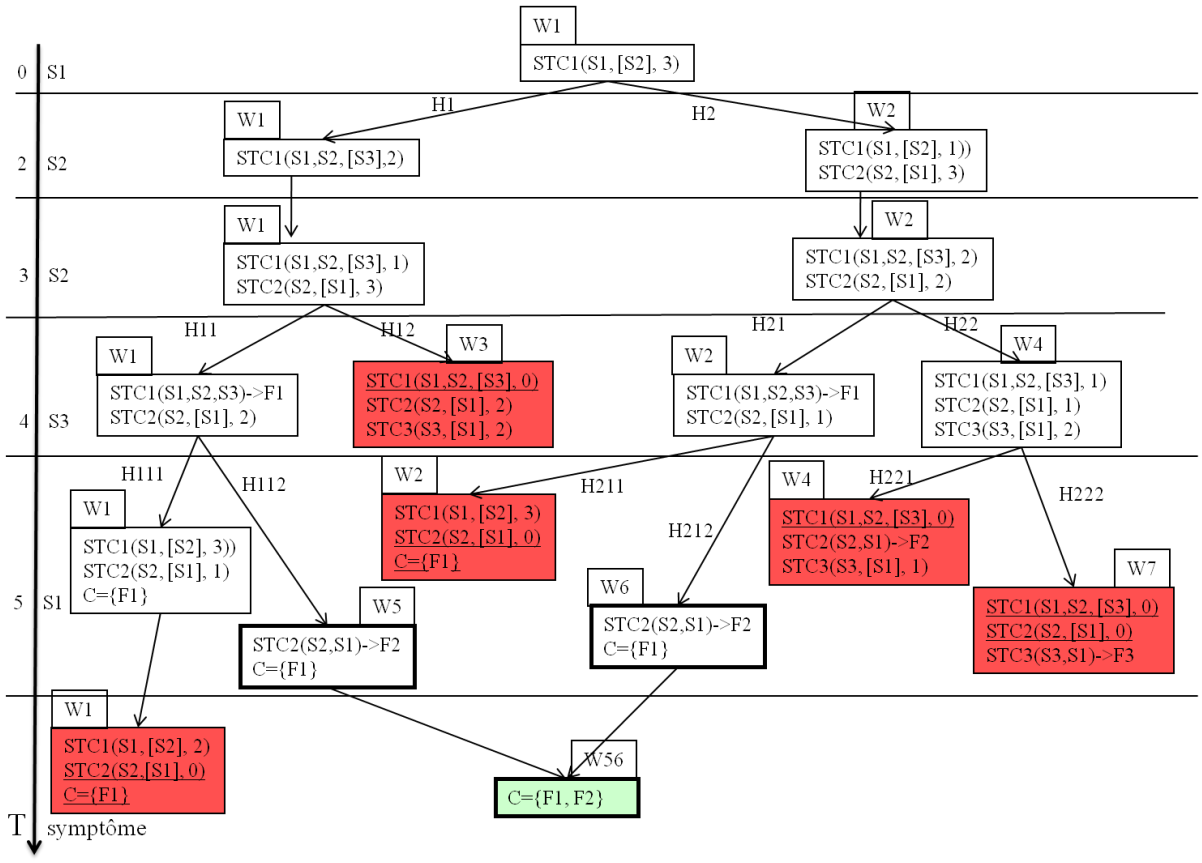


Figure VI-14 – Arbre de résolution de l'algorithme des mondes

Lorsque le module de diagnostic reçoit le premier 'S1', ce symptôme ne peut être affecté qu'à la STC1. Mais quand 'S2' est détecté à l'instant t = 2, d'après la Règle VI-11, STC1 et STC2 sont en concurrence. Par conséquence, le monde précédent 'W1' est dupliqué. Nous obtenons deux mondes 'W1' et 'W2' (duplication de 'W1' avant instanciation de STC2) qui correspondent à deux hypothèses d'affectation (respectivement H1 et H2). Ainsi dans 'W1', 'S2' est attribué à la STC1. Donc, cette hypothèse sera confirmée si 'S3' se produit dans un délai de 2 ut après 'S2', soit avant la date t=4 après cette affectation. L'hypothèse H2 affecte 'S2' à STC2. Par conséquent, dans 'W2', STC1 continue à attendre 'S2' pendant encore un délai de 1 ut et, STC2 attend 'S1' dans un délai de 3 ut Donc W2.MAX_TO (deadline du monde W2) est égale à 1 ut, ce qui signifie que l'hypothèse H2 serait fausse si ce timeout expire. Dans l'arbre de résolution, les mondes comme 'W3' à la date t = 4, 'W2', 'W4' et 'W7' à t = 5 et 'W1' à t = 6 sont supprimés en raison de l'expiration de leur timeout. A t = 5, les deux mondes 'W5' et 'W6' sont équivalents donc ils sont fusionnés d'après la Règle VI-14 (En pratique, l'un d'eux est supprimé pour ne conserver qu'un seul monde).

Ce sont la Règle VI-11, la Règle VI-12 et la Règle VI-13 qui garantissent la correction de cet algorithme. En effet, la Règle VI-11 garantit qu'un symptôme n'est affecté qu'à une seule STC dans un monde. En cas de compétition d'affectation, la Règle VI-12 garantit que chacun des mondes dupliqués correspond à une hypothèse concurrente d'affectation d'un symptôme. Bien entendu, une seule des hypothèses d'affectation est vraie. Les hypothèses fausses sont révélées par le fait qu'un événement ne peut pas être interprété ou que la deadline d'une STC

du monde a été atteinte. Dans ce cas, les mondes incohérents sont détruits par la Règle VI-13 et ne subsiste que le monde correspondant aux hypothèses cohérentes.

VI.V Application au cas de la carte FerroCOTS

Dans le chapitre précédent, nous nous sommes intéressés à la construction d'un diagnostiqueur d'une carte fille de la carte FerroCOTS. Le diagnostiqueur permet de donner l'état dans lequel la carte est. Nous avons modélisé l'occurrence d'une faute, comme par exemple le collage à 0 d'une sortie, par l'occurrence d'un événement inobservable que nous avons nommé $d^0f_k^i$. L'occurrence de $d^0f_k^i$ amène la carte vers un état fautif. En effet, nous pouvons modéliser autrement le système sans avoir recours à définir les fautes mais, en utilisant des timeout pour modéliser qu'une sortie n'a pas eu lieu dans l'intervalle prévu ou bien utiliser une garde pour une transition modélisant qu'une sortie par exemple a eu lieu avant la date prévue. Le modèle comportemental d'une ligne fonctionnelle d'une carte fille sera celui présenté dans la Figure VI-15.

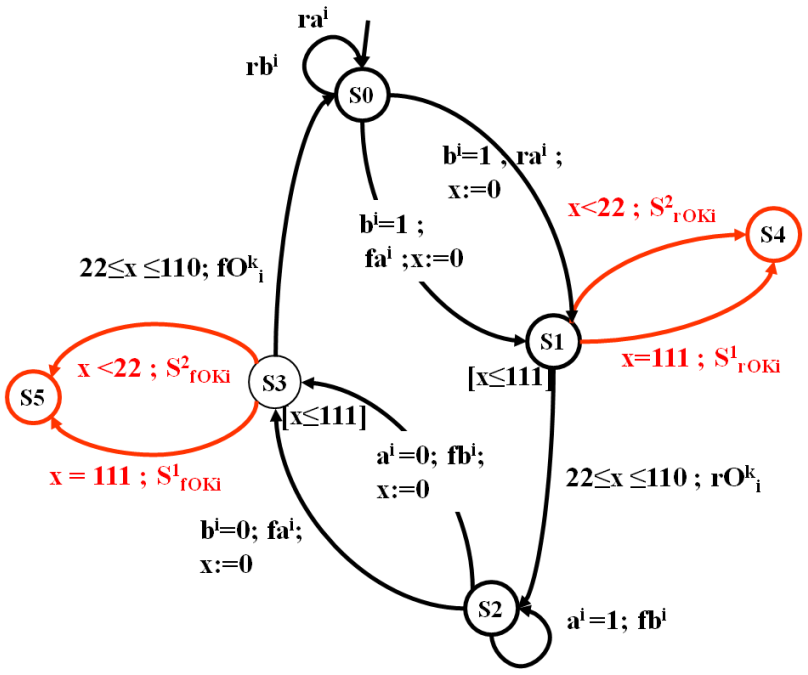


Figure VI-15 – Modèle comportemental d'une ligne fonctionnelle d'une carte fille

A partir de ce modèle, Nous pouvons définir le symptôme S^i_k qui signifie que la sortie O^i_k n'a pas changé de valeur dans le délai prévu. Ainsi, nous pouvons extraire l'ensemble suivant de STC :

STC1: (In, rb^i, [0,+∞[)* (rb^i, ra^i, [0,+∞[)* (ra^i, S^2_{rOki}, [0,22]) => (faute passage O^k_i à 1)

STC2: (In, rb^i, [0,+∞[)* (rb^i, fa^i, [0,+∞[)* (fa^i, S^2_{rOki}, [0,22]) => (faute passage O^k_i 1)

STC3: (In, ra^i, [0,+∞[)* (ra^i, S^1_{rOki}, [111,111]) => (faute maintient O^k_i 0)

STC4: (In, fa^i, [0,+∞[)* (fa^i, S^1_{rOki}, [111,111]) => (faute maintient O^k_i 0)

STC5: (In, fa^i, [0,+∞[)* (fa^i, fb^i, [0, +∞]) * (fb^i, S^2_{fOki}, [0,22] => (faute passage O^k_i 0)

STC6: (In, fb^i, [0,+∞[)* (fb^i, fa^i, [0, +∞]) * (fa^i, S^2_{fOki}, [0,22] => (faute passage O^k_i 0)

STC7: (In, fa^i, [0,+∞[)* (fa^i, fb^i, [0, +∞]) * (fb^i, S^1_{fOki}, [111,111] => (faute maintient O^k_i 1)

STC8: (In, fb^i, [0,+∞[)* (fb^i, fa^i, [0, +∞]) * (fa^i, S^1_{fOki}, [111,111] => (faute maintient O^k_i 1)

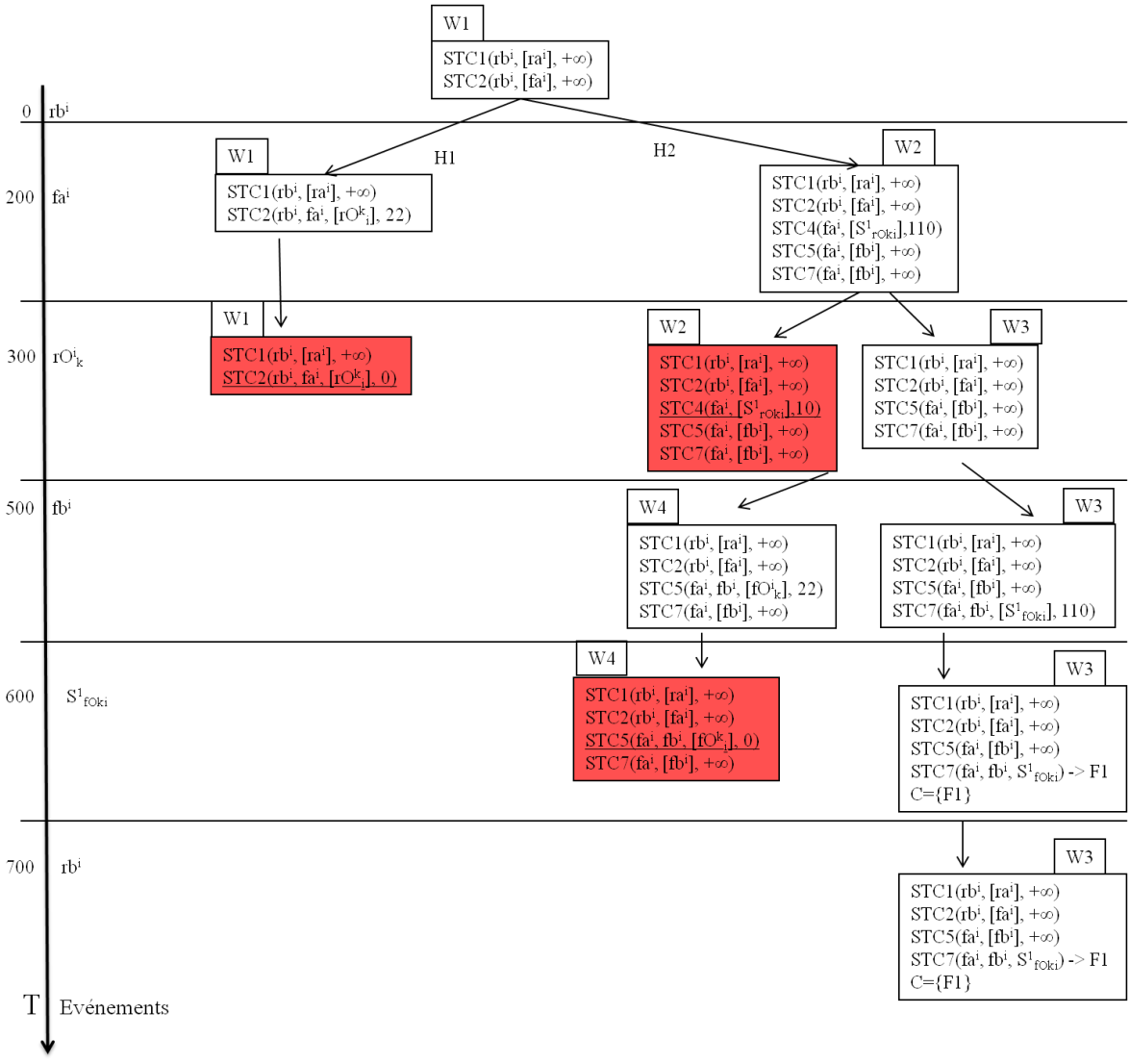


Figure VI-16 – Arbre de résolution de l'algorithme de reconnaissance sur l'exemple d'une carte fille de la carte FerroCOTS

Les 8 STC constituent notre base de STC pour le diagnostic d'une sortie d'une carte fille.

Nous pouvons appliquer ensuite notre algorithme de reconnaissance pour définir l'état du système. Supposons que les événements observés au fil du temps soient la séquence temporelle (rb^i, 0) * (fa^i, 200) * (rO^i_k, 300) * (fb^i, 500) * (S^i_k, 610)* (rb^i, 700). La Figure VI-16 montre l'arbre de résolution de notre algorithme de reconnaissance de STC sur l'exemple d'une carte fille de la carte FerroCOTS. Le diagnostic indique que le système est défaillant et que la sortie O^i_k de la carte fille est maintenue à 1.

Cet exemple montre que l'approche STC est une solution possible pour le diagnostic des systèmes embarqués logiques.

Conclusion du chapitre VI

Dans ce chapitre, nous avons cherché à proposer une alternative à l'approche diagnostiqueur pour le diagnostic des systèmes embarqués logiques. Nous avons d'abord montré que l'approche STC devait être améliorée afin de garantir la cohérence d'une base de STC. Dans ce cadre, nous avons proposé deux approches, l'une basées sur une formalisation de STC en RdP T-temporels (Saddem, et al., 2010) et l'autre basée sur les automates temporisées (Saddem, et al., 2011c). Ensuite, nous avons proposé un algorithme d'interprétation basé sur le concept de monde qui garantit la correction du diagnostic (Saddem, et al., 2012). Enfin, nous avons appliqué cet algorithme sur l'exemple de la carte FerroCOTS pour montrer que nous pouvons utiliser les STC pour le diagnostic de la carte au lieu de construire le diagnostiqueur. Cet exemple nous permet de déduire deux autres conclusions. Pour garantir la cohérence d'une base de STC, il est préférable de modéliser le comportement d'un système sous forme d'automate temporisé basé sur des événements observables et ensuite utiliser une model-checker comme KRONOS pour extraire des séquences d'événements caractérisant des fautes. Ces séquences représentent alors une base de STC cohérente. La deuxième conclusion, est que nous pouvons donc implémenter un ensemble de diagnostiqueurs sous forme de STC. Notons que KRONOS à la capacité à extraire à la volée des séquences d'un ensemble de modèles sans avoir à construire le modèle global.

Conclusion générale

Dans ce mémoire de thèse, nous avons présenté des travaux ayant pour objectif de contribuer au diagnostic des systèmes embarqués logiques. Un système embarqué logique est un système embarqué construit autour d'un ou plusieurs composants programmables (ou circuits intégrés) et possédant de nombreuses entrées/sorties tout-ou-rien. Aujourd'hui les industriels pour des questions de coûts de développement et pour faciliter les évolutions, ils s'intéressent à l'intégration de ces systèmes dans leurs architectures de contrôle-commande. La problématique industrielle est que ces systèmes ne sont pas qualifiés par les normes industrielles pour des raisons de sécurité (safety) des systèmes qu'ils contrôlent. L'approche industrielle naturelle consiste alors à utiliser des techniques de redondance pour rendre ces systèmes embarqués plus robustes. La redondance ne suffit toutefois pas. Il faut forcément intégrer du diagnostic pour identifier les composants défaillants avant une mission ou en cours de mission. Nous sommes donc confrontés à une problématique de diagnostic passif en ligne.

Dans ce travail de thèse, nous avons proposé d'approcher ce problème par deux techniques : l'approche diagnostiqueur et l'approche STC.

L'approche diagnostiqueur est celle sur laquelle nous avons porté l'essentiel de notre travail. En effet, nous avons profité de cette problématique industrielle pour identifier les solutions permettant de repousser le verrou principal de cette technique : l'explosion combinatoire.

Les principales solutions que nous avons proposées sont d'abord l'utilisation d'un formalisme plus expressif que celui couramment utilisé (les automates à états finis). Nous avons choisi d'utiliser les automates temporisés entre autre parce que nous voulions également exploiter les techniques de model-checking qui proposent des outils efficaces pour faire de la vérification formelle.

Au-delà du formalisme, nous avons proposé une approche d'ingénierie par les modèles pour la formalisation du comportement d'un système embarqué logique. Cette approche est d'abord basée sur une réduction du modèle comportemental initial du système. Dans ce cadre, nous avons proposé un algorithme de réduction adapté aux exigences de la surveillance de ce type de système. Cela nous a amené à proposer le concept de graphe d'état à transitions événementielles gardées. Nous avons ensuite montré qu'il est possible de mettre en œuvre les techniques de transformation de modèle pour généré un automate temporisé du comportement normal d'un composant logique à partir du graphe d'état à transitions événementielles gardées et d'une abstraction temporelle de son comportement. Le modèle est ensuite complété avec les événements caractérisant les fautes du composant.

Dans un deuxième temps, nous avons cherché à proposer une alternative à l'approche diagnostiqueur pour le diagnostic des systèmes embarqués logiques. Nous avons d'abord montré que l'approche STC devait être améliorée afin de garantir la cohérence d'une base de STC. Dans ce cadre, nous avons proposé deux approches, l'une basées sur une formalisation de STC en RdP T-temporels et l'autre basée sur les automates temporisées. Ensuite, nous avons proposé un algorithme d'interprétation basé sur le concept de monde qui garantit la correction du diagnostic. Nous devons néanmoins indiquer ici que l'approche de résolution basée sur les modes amène à dupliquer les instances des motifs de STC dans les différents mondes en cours de résolution. C'est donc une approche plus coûteuse en mémoire que la résolution basée sur la technique de l'automate dupliqué. De ce fait, elle est plus indiqué pour une interprétation par un calculateur possédant beaucoup de mémoire vive que embarqué dans un composant numérique.

Nous pouvons envisager de très nombreuses perspectives pour ce travail.

La première perspective concerne les outils permettant de manipuler les différents modèles. Dans ce travail de thèse, nous avons utilisé des outils de model-checking comme KRONOS et UPPAAL qui sont puissants mais exigent que le concepteur connaissent les techniques permettant de test de la diagnosticabilité ou les techniques de synthèse d'un diagnostiqueur à partir de modèles déterministe ou indéterministe. Il est clair que ce genre d'approche passera d'autant plus facilement dans les approches d'ingénieries des systèmes si elle repose sur ses propres outils. Dans le cas des automates à états finis, on a des outils comme DESUMA/UMDES qui permettent de construire assez simplement les diagnostiqueurs. Nous pensons développer un outil semblable qui facilitera le travail de la conception d'un système de diagnostic basé sur les automates temporisés.

Dans cette thèse nous avons fait le choix du formalisme des automates temporisés car pour le développement de notre méthode, l'étude dont nous avons donné une synthèse au chapitre II et à la fin du chapitre IV, nous a montré que nous disposions pour ce formalisme d'outils puissants permettant de construire des modèles par composition ou de vérifier des propriétés à la volée. Il n'en demeure pas moins que malgré les hypothèses simplificatrices permettant de réduire la taille de nos modèles par abstraction et la méthode de réduction proposée, l'approche diagnostiqueur explose rapidement si nous voulons l'utiliser pour faire de l'identification des défaillances. Aussi, il nous semble important de poursuivre ce travail dans deux directions. La première direction consiste à étudier l'exploitation de la diagnosticabilité modulaire des automates temporisés dans le cadre d'une approche fonctionnelle. La deuxième direction est le développement d'une approche diagnostiqueur basée sur les réseaux de Petri temporels.

Dans (Toguyeni, 1992) et (Toguyèni, 2001), il a été proposé une méthode de modélisation fonctionnelle des systèmes complexes basée sur le Graphe Fonctionnel. Le Graphe Fonctionnel est un modèle qui représente les redondances fonctionnelles d'un système. D'autre part, il modélise les relations de dépendances fonctionnelles entre les fonctions du

niveau système et les fonctions du niveau composants. L'idée dans ce cadre et donc d'utiliser le Graphe Fonctionnel pour structurer un réseau de diagnostiqueurs. On pourrait associer un diagnostiqueur pour le diagnostic de l'état de certaines fonctions du modèle disposant d'une observabilité suffisante. Chaque diagnostiqueur pris séparément, il ferait de la localisation d'un sous ensemble fonctionnel défaillant. L'identification des causes de défaillances seraient alors obtenue par un raisonnement hypothétique mené en propageant des hypothèses de défaillances en exploitant les relations de dépendance entre fonctions. Pour garantir l'identification de toute défaillance, il faudrait que les fonctions directement observables permettent d'obtenir une observabilité indirecte pour l'ensemble des fonctions du Graphe Fonctionnel. Cela permettrait même d'envisager de construire un réseau d'agents de diagnostic mixant aussi bien pour certains agents l'approche diagnostiqueur et pour d'autres l'approches STC.

La troisième perspective que nous évoquerons ici concerne le développement des diagnostiqueurs basés sur les Réseaux de Petri Temporels. Au chapitre II, nous avons montré que ce type de RdP a une expressivité comparable à celle des automates temporisés. D'autre part, les réseaux de Petri modélisent plus naturellement le parallélisme que les automates temporisés. Nous pouvons donc espérer développer des diagnostiqueurs modulaires basés sur ce formalisme sans avoir la même combinatoire qu'avec les formalismes d'automates. Des travaux sont en cours dans notre équipe sur ce thème. Ils sont basés sur une extension de l'équation d'état des Réseaux de Petri pour caractériser pour chaque marquage atteint le vecteur d'événements observés et le vecteur de fautes probables. Cela permet de définir des classes d'états caractérisant l'ensemble des états atteints à partir de l'état initial et par une séquence de transition se terminant par le franchissement d'une transition associée au même événement observable. Des travaux préliminaires ont été effectués basés sur les RdP synchronisés (Liu, et al., 2012). Ils sont en cours d'extension aux RdP T-temporels.

La dernière perspective que nous évoquerons ici est une mise en œuvre sécurisée des modèles de diagnostique que nous proposons. L'idée est d'avoir la même approche pour la mise en œuvre du diagnostic que l'approche redondante utilisée pour la commande. Il s'agit de répartir les modèles de diagnostic des différents composants sur plusieurs composants programmables. Par exemple, le diagnostiqueur relatif à une carte fille serait implémenté sur les composants programmables des trois cartes filles. Nous pourrions alors mettre en œuvre au niveau de chaque carte fille un mécanisme de diagnostic distribué comme inspiré par les travaux de Sengupta (Sengupta, 1998).

Bibliographie

Abreu, R., P. Zoeteweij, et A. J. C. van Gemund. «An evaluation of similarity coefficients for software fault localization.» Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC'06). California, USA: IEEE Computer Society Washington, DC, USA, 2006, pp 39–46.

Abreu, R. «Spectrum-based Fault Localization in Embedded Software.» Thèse de l'Université de Delft, Pays Bas, 2009.

Alur, R., et T. Henzinger. «Logics and models of real time : a survey.» REX workshop, Real-Time: Theory in practice. Mook, The Netherlands, 3-7 juin 1991, pp 74-106.

Alur, R., et D. L. Dill. «A theory of timed automata.» Theoretical Computer Science, Volume 126, n° 2, 1994, pp 183-235.

Alur, R., L. Fix, et T. A. Henzinger. «A Determinizable Class of Timed Automata.» 6 th International Conference on Computer Aided Verification (CAV'94). Stanford, California, USA, 21–23 juin 1994, Lecture Notes in Computer Science, pp 1-13.

Alur, R., L. Fix, et T. A. Henzinger. «Event-Clock Automata: A Determinizable Class of Timed Automata.» Theoretical Computer Science. Volume 211, n° 1-2, Janvier 1999, pp 253–273.

Antola, A., V. Piuri, et M. Sami. «On-line Diagnosis and Reconfiguration of FPGA Systems.» First IEEE International Workshop on Electronic Design, Test and Applications DELTA, Christchurch, New Zealand, 29-31 janvier 2002, pp 291–296.

Baier, C., N. Bertrand, P. Bouyer, et T. Brihave. «When are Timed Automata Determinizable.» 36th International Colloquium on Automata, Languages and Programming, ICALP. Rhodes, Greece, 2009, pp 43-54.

Behrmann, G., A. David, et K. G. Larsen. «A Tutorial on Uppaal.» 4th International School on Formal Methods for the Design of Computer, Communication and Software Systems : Real Time (SFM-04 :RT). Bertinoro, Italie,13-18 Septembre 2004, pp 200–236.

Bérard, B. «Model checking temporisé.» Approches formelles des systèmes embarqués communicants. Hermès, 2008, pp 75-103.

Berthomieu, B., et M. Menasche. «An Enumerative Approach for Analyzing Time Petri Nets.» IFIP Congress. Paris, Septembre 1983, pp 19-23.

Berthomieu, B., et F. Vernadat. «State class constructions for branching analysis of time Petri nets.» 9th International Conference on Tools and Algorithms for the construction and Analysis of Systems, TACAS. Springer Verlag LNCS 2619. Warsaw, Poland, avril 2003, pp 7-11.

Berthomieu, B., P. O. Ribet, et F. Vernadat. «The tool TINA – Construction of Abstract State Spaces for Petri Nets and Time Petri Nets.» International Journal of Production Research. Volume 42, n° 14, juillet 2004, pp 2741-2756.

Bertrand, O. «Détection d'activités par un système de reconnaissance de chroniques et application au cas des simulations distribuées HLA.» Thèse de docteur de l'université, Paris 13, 2009.

Boulanger, J. L. «Principles.» Safety of Computer Architectures, ISTE – WILEY, 2010, pp 1-46.

Bouyer, P., F. Chevalier, et D. D'Souza. «Fault diagnosis using timed automata.» International Conference on Foundations of Software Science and Computational Structures (FoSSaCS'05), Lecture Notes in Computer Science, Edinburgh, UK, 4-8 Avril 2005. pp 219-233.

Bouyer, P., F. Chevalier, M. Krichen, et S. Tripakis. «Observation partielle des systèmes temporisés.» Journal européen des systèmes automatisés (JESA), Lavoisier, Octobre 2005, pp 381-393.

Bozga, M., C. Daws, O. Maler, A. Olivero, S. Tripakis, et S. Yovine. «KRONOS: A Model-Checking Tool for Real-Time Systems.» International Symposium on Formal Techniques in Real Time and Fault Tolerant systems (FTRTFT). Springer. Lyngby, Denmark. 14-18 Septembre 1998, pp 298-302.

Brams, G. W. Réseaux de Petri: théorie et pratique. Masson, Paris , 1982.

Cassandras, C. G., et S. Lafortune. Introduction to Discrete Event Systems, 2nd edition, XXIV, 772p. Springer, 2008.

Cassez, F., et S. Tripakis. Approches formelles des systèmes embarqués communicants. Hermès, 2008.

CEI, Norme. «Commission Electronique Internationale, Liste des termes de base, définitions et mathématiques applicable à la fiabilité, Référence 271.» 1974.

CENELEC. «NF EN 50126, Railway applications – the specification and demonstration of reliability, availability, maintenability and safety.» 2000.

CENELEC. «NF EN 50128, Railway applications – communications, signaling and processing systems – software for railway control and protection systems.» 2001.

CENELEC. «NF EN 50129, European standard. Railway applications – communications, signaling and processing systems – safety-related communications in transmission systems.» 2003.

Chamseddine, N. «Analyse quantitative paramétrée d'automates temporisés probabilistes.» Thèse de Doctorat de l'École Normale Supérieure de Cachan, 2009.

Clarke, E. M., et E. A. Emerson. Design and Synthesis of Skeltons Using Branching Time Temporal Logic. Proceeding of the IBM Workshop on logic of programs,Volume 131. Lectures Notes in Computer Science. Yorktown Heights, New York, USA, Mai 1981, pp 52-71.

Clermidy, J. «Amélioration de la fiabilité des calculateurs parallèles SIMD par test et tolérances aux fautes structurelles.» Thèse de Doctorat de l'Institut National Polythechnique de Grenoble, 1999.

Contant, O., S. Lafortune, et D. Teneketzis. «Diagnosability of Discrete Event Systems with Modular Structure.» Discrete Event Dynamic Systems. Volume 16, n° 1, 2006, pp 9-37.

Cordier, M. O., et C. Dousson. «Alarm Driven Monitoring Based on Chronicles.» 4th Symposium on Fault Detection, Supervision and Safety for Technical (Safeprocess), Budapest, Hongrie, 14-16 Juin 2000, pp 286-291.

Cordier, M. O., X. Le Guillou, S. Robin, L. Rozé, et T. Vidal. «Distributed Chronicles for On-line Diagnosis of Web Services.» The 18th International Workshop on Principles of Diagnosis (DX'07), Nashville, TN, USA, 29-31 Mai 2007, pp 37-44.

David, A., G. Behrmann, K. G. Larsen, P. Pettersson, J. I. Rasmussen, W. Yi, D. Lime, M. Magnin, O. H. Roux. «Outils de Model Checking.» Chapitre 7 dans Approches formelles des systèmes embarqués communicants, édité par O. Roux et C. Jard, 199-244. Hermès, 2008.

David, R., et H. Alla. Du Grafcet aux réseaux de Petri. Paris: Hermès, 1992.

Davis, R. Diagnostic reasoning based on structure and behavior. Volume 24, N°1-3, Artificial Intelligence, Décembre 1984, pp 347-410.

Daws, C., A. Olivero, et S. Yovine. «Verifying ETLOTOS programs with KRONOS.» Édité par Chapman et Hall. 7th. IFIPWG G.1 International Conference of Formal Description Techniques, FORTE'94. Bern, Switzerland, 1994, pp 227-242.

de Frutos Escrig, D., V. Valero Ruiz, et O. Marroquin Alonso. «Decidability of properties of timed-arc petri nets.» 21st International Conference on Application and Theory of Petri Nets (ICATPN'00). Aarhus, Denmark, Lecture Notes in Computer Science 1825, 26-30 Juin 2000, pp 187-206.

de Kleer, J., et J. S. Brown. «Mental Models of Physical Mecanisms and their Acquisition.» Dans Cognitive and Instructional Sciences Group, Hill Road Palo Alto, 1980, pp 285-309.

de Kleer, J., et J. S. Brown. A Qualitative Physical Based on Confluences. Artificial Intelligence, Volume 24, n° 1-3, Decembre 1984, pp 7-83.

de Kleer, J., et B. C. Williams. Diagnosing Multiple Faults. Volume 32, n° 1, Artificial Inteligence, Avril 1987, pp 97-130.

Debouk, R., S. Lafortune, et D. Teneketzis. «Coordinated decentralized protocols for failure diagnosis of discrete events systems.» Discret Event Dynamic Systems : Theory and Application (Kluwer Academic Publishers), Volume 10, 2000, pp 33-86.

Debouk, R., R. Malik, et B. Brandin. «A modular architecture for diagnosis of discrete event systems.» IEEE Conference on Decision and Control. Warren, MI, USA , 2002, pp 417–422.

Debouk, R. «Diagnosis of discrete event systems: a modular approach.» IEEE International Conference on Systems, Man and Cybernetics, Washington DC USA, 5-8 Octobre 2003, pp 306 - 311.

Diaz, M. Les réseaux de Petri : modèles fondamentaux. Hermès, 2001.

Dousson, C., P. Gaborit, et M. Ghallab. «Situation Recognition : Representation and Algorithms.» International Joint Conference on Artificial Intelligence (IJCAI'93). Chambéry, France, 1993, pp 166-172.

Finkel, O. «On decision problem for timed automata.» Bulletin of the European Association for Theoretical Computer Science, 2005, pp 185-190.

Froger, E. «Diagnostic de systèmes complexes : modélisation des raisonnements, architecture multi-expret et validation.» Thèse de Doctorat, Nantes, 1990.

Gardey, G., D. Lime, M. Magnin, et O. Roux. «Roméo: A tool for analyzing time Petri nets.» 17th International Conference on Computer Aided Verification (CAV'05), Lecture Notes in Computer Science LNCS 3576, Edinburg, Scotland, UK, 6-10 juillet 2005.

Genc, S., et S. Lafortune. «Distributed Diagnosis of Discrete-Event Systems Using Petri Nets.» 24th International Conference on Applications and theory of Petri Nets ICATPN, Eindhoven, Netherlands, Lecture Notes in Computer Science LNCS 2679, 23-27 Juin 2003, pp 316-336.

Genesereth, M. R. The Use of Design Descriptions in Automated Diagnosis. Volume 24, N° 1-3, Artificial Intelligence, Décembre 1984, pp 411-436.

Grisel, R., et N. Abouchi. «Les systèmes embarqués : une introduction.» http://richard.grisel.free.fr/Master_OSM/2_Introduction_Embedded_systems.pdf (date d'accès le 21 octobre, 2012).

Guerraz, B., et C. Dousson. «Chronicles construction starting from the fault model of the system to diagnose.» The 15th International Workshop on Principles of Diagnosis (DX'04), Carcasonne, France, 23-25 Juin 2004, pp 51–56.

Henzinger, T. A., X. Nicollin, J. Sifakis, et S. Yo. Symbolic model checking for real-time systems. Volume 111, n° 2, Information and Computation, 1994, pp 193-244.

Holloway, L., et N. Pandalai. «Template Languages for Fault Monitoring of Timed Discrete Event Processes.» IEEE Transactions on Automatic Control. Volume 45, Issue 5,2000, pp 868-882.

Holzmann, G. J. «Software model checking with SPIN.» Advances in Computers. Volume 65, 2005, pp 78-109.

Jéron, T., H. Marchand, S. Pinchinat, et M. O. Cordier. «Supervision Patterns in Discrete Event Systems Diagnosis.» Workshop on Discrete Event Systems, WODES'06. Ann-Arbor , MI, USA, 2006.

Jiang, S., Z. Hiang, V. Chandra, et R. Kumar. «A polynomial Algorithm for Testing Diagnosability of Discrete Event Systems.» IEEE Transactions on Automatic Control. Volume 46, n° 8, 2001, pp 1318 – 1321.

Kadionik, P. «Les systèmes embarqués : une introduction.» décembre 2005. http://www.enseirb-matmeca.fr/cosynux/grand_public/embedded_intro_2006.pdf (accès le 22/10/2012).

Khansa, W. «Réseaux de Petri P-temporels : Contribution à l'étude des systèmes à événements discrets.» Thèse de Doctorat, Université de Savoie, Annecy, 1997.

Lafortune, S., Y. Wang, et T. S. Yoo. «Diagnostic Decentralise Des Systemes A Evenements Discrets.» Compte-rendu du Colloque International sur la Modélisation des Systèmes Réactifs (MSR'05), Grenoble, France, 5-7 Octobre, 2005, Journal Europeen des Systèmes Automatisés (JESA), Lavoisier, 2005, pp. 95-110.

Larsen, K. G., P. Pettersson, et W. Yi. «Uppaal in a Nutshell.» Journal of Software Tools for Technology Transfer. Volume 1, n° 1–2, 1997, pp 134-152.

Lin, F., et W. M. Wonham. «On observability of discrete event systems.» International Journal on Information Sciences - Robotics and Automation. Volume 44, 1988, pp 173 - 198.

Lin, F. Diagnosability of disrete event systems and its applications. Discrete Event Dynamic Systems, Volume 4, Springer, 1994, pp 197-212.

Liu, B., M. Ghazel, et A. A. K. Toguyeni. «K-diagnosability of labeled Petri nets.» MajecSTIC , 29-31 octobre 2012, Lille, France.

Maler, O., et S. Yovine. «Hardware timing verification using KRONOS.» 7th Israelian Conference on Computer Systems and Software Engineering. Herzliya, Israel, 1996.

Manfoumbi, V., et H. Achigui. «Détection, Diagnostic des fautes sur des FPGA.» 2003. http://www.cours.polymtl.ca/ele6306/data/proj/r20031212_14.pdf.

Merlin, P. «A study of the recoverability of computer system.» Thèse de Doctorat, Université de Californie, Californie, 1974.

Ozveren, C. M., et A. S. Willsky. «Observability of Discrete Event Dynamic Systems.» IEEE Transactions on Automatic Control. Volume 35, Issue 7, 1990, pp 797–806.

Ozveren, C. M., et A. S. Willsky. Invertibility of Descret Event Dynamic system. Volume 5, Mathematics of Control, Signals, and Systems, Springer, 1992, pp 365-390.

Pnueli, A. «The temporal Logic of Programs.» 18ème IEEE Symposium Foundations of Computer Science (FOCS'77). Providence, Rhode Island, USA, 31 octobre – 1 novembre 1977, pp 46-57.

Ramadge, P. J., et W. M. Wonham. «On the Supremal Controllable Language of a Given Language.» SIAM Journal of Control and Optimization. Volume 25, n° 3, 1987, pp 637-659.

Ramadge, P. J., et W. M. Wonham. «Supervisory Control of a Class of Discrete Event Processes.» SIAM Journal of Control and Optimization. Volume 25, n°1, 1987, pp 206-230.

Ramchandani, C. «Analysis of asynchronous concurrent systems by timed Petri nets.» Thèse de Doctorat, MIT Cambridge, 1974.

Reiter, R. "A Theory of Diagnosis from First principles". Artificial Intelligence. Volume 32, n° 1, 1987, pp 57-95.

Saddem, R., A. K. A. Toguyeni, et M. Tagina. «Consistency's checking of chronicles' set Using Time Petri Nets.» 18th Mediterranean Conference on Control & Automation (IEEE MED'10). Marrakech, Maroc, 2010, pp 1520 - 1525.

Saddem, R., A. K. A. Toguyeni, et M. Tagina. «Diagnostic des systèmes embarqués critiques : Application à la carte de commande du système de freinage d'un train.» Journal Européen des Systèmes Automatisés. Volume 45, n° 1-3, 2011, pp 205-220.

Saddem, R., A. K. A. Toguyeni, et M. Tagina. «Diagnosis of Critical Embedded Systems: Application to the Control Card of a Railway Vehicle Braking Systems.» 7th International IEEE Conference on Automation Science and Engineering, CASE'2011. Trieste, Italia, 2011, pp. 163-168.

Saddem, R., A. K. A. Toguyeni, et M. Tagina. «A model-checking approach for checking the Consistency of a set of Causal Temporal Signatures.» Proceeding of 9th European Workshop on Advanced Control and Diagnosis, ACD 2011, Budapest, Hongrie Novembre 2011.

Saddem, R., A. K. A. Toguyeni, et M. Tagina. «Algorithme d'interprétation d'une base de signatures temporelles causales pour le diagnostic en ligne des Systèmes à événements Discrets.» 9ème Conférence Internationale de Modélisation, Optimisation et SIMulation, MOSIM 2012. Bordeaux, France, 2012.

Sampath, M., R. Sengupta, S. Lafortune, et K. Sinnamoh. «Diagnosability of Discrete-Event Systems.» IEEE Transactions on Automatic Control. Volume 40, Issue 9, 1995, pp 1555 – 1575.

Sampath, M., R. Sengupta, K. Sinnamohideen, S. Lafortune, et D. Teneketzis. «Failure diagnosis using discrete event systems.» IEEE Transaction on Control System Technology. Volume 4, Issue 2, 1996, pp 105–124.

Sava, A. T. «Sur la synthèse de la commande des systèmes à évènements Discrets temporisés.» Thèse de Doctorat, Institut National Polytechnique de Grenoble, 2001.

Sengupta, R. «Diagnosis And Communication In Distributed Systems.» 4th International Workshop on Discrete Event Systems WODES'98, Cagliari, Italie, Aout 1998, pp 26-28.

Sifakis, J. "Use of Petri Nets for performance evaluation", Measuring Modeling and evaluating Computer Systems. 3rd International Symposium on Modeling and Evaluation, IFIP. North Holland, 1977, pp 75-93.

Stainer, A. «Test d'automates temporisés.» Master informatique, INRIA, Renne, 2010.

Stankovic, J. A. «Misconceptions about real-time computing.» IEEE Computer. Volume 21, n° 10, octobre 1988, pp 10-19.

Staroswiecki, M., et J. P. Cassar. «Approche structurelle pour la conception des systèmes de surveillance.» Ecole d'Eté d'Automatique. Grenoble, France, 1996.

Staroswiecki, M., J. P. Cassar, et P. Declerck. «A structural framework for the design of FDI system in large scale industrial plants.» Issues of Fault Diagnosis for Dynamic Systems. Springer Verlag, 2000, pp 245–283.

Stroud, C., S. Konala, P. Chen, et M. Abramovici. «Built-In Self-Test for Programmable Logic Blocks in FPGAs (Finally, A Free Lunch: BIST Without Overhead!).» IEEE VLSI Test Symposium VTS'96. Princeton, NJ, USA, 28 Avril - 1 Mai 1996, pp 387 - 392.

Stroud, C., E. Lee, S. Konala, et M. Abramovici. «Using ILA Testing for BIST in FPGAs.» International Test Conference, ITC'96. Washington DC, USA, 20-25 octobre 1996, pp 68-75.

Stroud, C., E. Lee, et M. Abramovici. «BIST-Based Diaignostics of FPGA Logic Bllocks.» International Test Conference, ITC'97. Washington DC, USA, 3-5 novembre 1997, pp 539 – 547.

Suman, P. V., P. K. Pandya, et S. Naray. «Timed Automata with Integer Resets: Language Inclusion and Expressiveness.» Technical Report TIFR-SPKM-GM-2008/1, Juillet 2008.

Toguyeni, A. K. A., E. Craye, et J. C. Gentina. « A method of Temporal Analysis to perform online Diagnosis in the context of Flexible Manufacturing System.» 16th Annual conference of IEEE Industrial Electronics Society (IECON'90), Vomume 1, Pacific Grove, Californie, USA, 27-30 Novembre 1990, pp 445 -450.

Toguyeni, A. K. A. «Surveillance et Diagnostic en ligne dans les ateliers flexibles de l'industrie manufacturière.» Thèse de Doctorat, Université de Lille1, 1992.

Toguyeni, A. K. A., E. Craye, et J. C. Gentina. «Time and reasoning for on-line diagnosis of failures in flexible manufacturing systems.» 15th IMACS world congress on scientific computation, modeling, and applied mathematics, Berlin, Allemagne, 24-29 Aout 1997, pp 709-714.

Toguyeni, A. K. A. «Contribution à la tolérance aux fautes des systèmes flexibles de production manufacturière.» Habilitation à diriger des recherches, Université de Lille1, 2001.

Tomoo, I., M. Satoshi, et F. Hideo. «Universal Fault Diagnosis for Lookup Table FPGAs.» IEEE Design & Test of Computers, Volume 15, n° 1, janvier-mars 1998, pp 39-44.

Tripakis, S. «Verifying progress in timed systems.», Formal Methods for Real-Time and Probabilistic Systems, 5th International AMAST Workshop, ARTS'99, LNCS 1601, Bamberg, Allemagne, 26-28 Mai 1999, pp 299-314.

Tripakis, S. «Fault Diagnosis for Timed Automata.» 7th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, (FTRTFT), LNCS 2469, Oldenburg, Allemagne, 9-12 Septembre 2002, pp 205-224.

Tripakis, S. «Folk theorems on the determinization and minimization of timed automata.» Information Processing Letters, Volume 99, n°6, Septembre 2006, pp 222-226.

Tripakis, S. « Checking timed Büchi automata emptiness on simulation graphs.» ACM Transactions on Computational Logic (TOCL), Volume 10, n°3, Article No. 15, Avril 2009, New York, NY, USA, pp 222-226.

van Gemund, A. J.C., , J. Pietersma, et A. Bos. «A Model-based Approach to Fault Diagnosis of Embedded System.» 10th Advanced School for Computing and Imaging, ASCI'04. Ouddorp, Netherlands, 2-4 Juin 2004, pp 189-196.

Villemeur, A. Sûreté de fonctionnement des systèmes industriels. Eyrolles. Paris, 1988.

Wang, Y., T. S. Yoo, et S. Lafortune. «Decentralized Diagnosis of Discrete Event Systems Using Conditional and Unconditional Decisions.» 44th IEEE Conférence on Decision and Control, et European Control Conference (DCD-ECC'05), Sevile, Spain, 12-15 Décembre 2005, pp 6298-6304.

Wang, Y., T. S. Yoo, et S. Lafortune. «Diagnosis of Discrete Event Systems Using Decentralized Architectures.» Journal of Discrete Event Dynamic Systems. Volume 17, n° 2, Juin 2007, pp 233-263.

Yoo, T., et S. Lafortune. «Polynomial-Time Verification of Diagnosability of Partially-Observed Discrete-Event Systems.» IEEE Transactions on Automatic Control, Volume 47, n°9, 2002, pp 1491 –1495.

Yovine, S. «KRONOS: A Verification Tool for Real-Time Systems.» Journal STTT. Volume 1, n°1-2, 1997, pp 123-133.

Zoeteweij, P., J. Pietersma, R. Abreu, A. Feldman, et A. J. C. van Gemund. «Automated Fault Diagnosis in Embedded Systems.» 2008 Second International Conference on Secure System Integration and Reliability Improvement (SSIRI '08). Yokohama, Japan, 14-17 Juillet 2008, pp 103-110.

Zwingelstein, G. «Diagnostic des défaillances : théorie et pratique pour les systèmes industriels.» Traité des Nouvelles Technologies : Série Diagnostic et Maintenance. 601 pages, Hermès, Paris, 1995.

Annexe A. Méthodes de réduction des systèmes logiques

Modéliser un système en une machine à état devient de plus en plus complexe et couteux dès que la taille de la machine à états devient importante. Pour rendre plus simple et pour réduire la complexité du problème, nous avons eu recours aux méthodes de réduction d'états dans une machine à états. La réduction d'états consiste en la suppression des états qui sont équivalents dans une machine à états. Les trois méthodes principales pour la réduction d'états sont : la méthode de ligne correspondante « row matching », la méthode d'implication graphique « implication charts », et la méthode de partitionnement successif « successive partitioning ». La méthode de ligne correspondante, la plus simple des trois, marche bien quand on a des tables de transition d'état qui ont un état suivant évident et des équivalences entre les sorties. Cette méthode ne donne pas en général la machine à états la plus optimale d'un point de vue réduction. Néanmoins, comme elle est simple à appliquer et elle donne souvent des résultats correctes, elle est généralement adoptée. La méthode d'implication graphique emploie une grille graphique pour aider à trouver les équivalences et les implications entre les états. C'est une excellente approche systématique de réduction des machines à états. La méthode du partitionnement successifs est quasiment un métissage entre les deux autres méthodes dans laquelle on utilise la table graphique et l'équivalence d'incidence. Souvent, chaque méthode réduit la taille et le nombre d'états d'une machine à états. En outre, une méthode peut donner un résultat plus optimal qu'une autre méthode.

La méthode de ligne correspondante

Elle utilise le théorème d'équivalence d'états: Soient Si et Sj deux états. Si = Sj si et seulement si, pour chaque combinaison d'entrée, Si et Sj ont la même sortie et des états suivants équivalents. Toutes les combinaisons d'entrée doivent être envisagées, mais l'information sur l'état interne du système peut être ignorée. En appliquant ce théorème, on considère la sortie et l'état suivant de chaque état. Toutefois, seules les entrées simples plutôt que les séquences d'entrée sont prises en compte. Les étapes de la méthode de ligne correspondante sont :

1. Construire la table de transition d'état
2. Identifier les états ayant les mêmes sorties
3. S'ils ont le même état suivant, ils sont équivalents
4. Fusionner-les en un seul nouvel état et renommer le
5. Répétez jusqu'à ce qu'il n'y ait plus de nouvel état

Soit l'exemple d'une machine à état de 15 états, S0.. S14, qui a une entrée sur un bit et une sortie sur un bit. La table de transition est présentée par la Table A-2. D'abord, on note que les états S10 et S12 ont la même sortie et les mêmes états suivants. Donc on regroupe ces deux états en un seul état qu'on appelle S10'. De même, on regroupe les états S7, S8, S9, S11, S13 et S14 en un état qu'on appelle S7'. La Table A-1 se transforme en Table A-2. On répète le même traitement aux états S3 et S6 qui ont la même sortie et les mêmes états suivants et on les regroupe en un seul état S3'. De même, on regroupe les états S4 et S5 en S4'. On obtient la Table A-3. En fin, nous avons réduit le modèle de 15 états en un modèle de 7 états.

Etat	Etat suivant		Sortie	
	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	0	0
S1	S3	S4	0	0
S2	S5	S6	0	0
S3	S7	S8	0	0
S4	S9	S10	0	0
S5	S11	S12	0	0
S6	S13	S14	0	0
S7	S0	S0	0	0
S8	S0	S0	0	0
S9	S0	S0	0	0
S10	S0	S0	1	0
S11	S0	S0	0	0
S12	S0	S0	1	0
S13	S0	S0	0	0
S14	S0	S0	0	0

Table A-1 - Exemple de réduction avec la méthode de ligne correspondante

Etat	Etat suivant		Sortie	
	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	0	0
S1	S3	S4	0	0
S2	S5	S6	0	0
S3	S7'	S7'	0	0
S4	S7'	S10'	0	0
S5	S7'	S10'	0	0
S6	S7'	S7'	0	0
S7'	S0	S0	0	0
S10'	S0	S0	1	0

Table A-2 - Transformation de la Table A-2

Etat	Etat suivant		Sortie	
	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	0	0
S1	S3’	S4'	0	0
S2	S4’	S3’	0	0
S3’	S7’	S7’	0	0
S4’	S7’	S10’	0	0
S7’	S0	S0	0	0
S10’	S0	S0	1	0

Table A-3 - Transformation de la Table A-2

La méthode d’implication graphique

La méthode d’implication graphique utilise une grille graphique de trie de façon à trouver systématique les équivalences entre les états. Le tableau d’implication aide à garder une trace de toutes les conséquences. Les étapes de la méthode graphique d’implication sont :

1. Construire le graphe d’implication, un carré pour chaque combinaison de deux états.
2. Pour le carré de la paire d’états (Si, Sj), si les sorties sont différentes, alors le carré devient un « X ». Sinon, écrire dans le carré les paires d’états impliqués pour toutes les combinaisons d'entrée.
3. Progresser à travers le graphique du haut en bas et de gauche à droite. Si le carré de la paire d’états (Si, Sj) contient la paire d’états (Sm, Sn) comme états suivants et le carré de cette paire est déjà étiqueté par « X », alors étiqueter le carré de la paire (Si, Sj) par « X ».
4. Continuer l'exécution de la 3ème étape jusqu'à ce qu'il n'y ait plus de nouveau carré marqué par « X ».
5. Pour tout carré de paire (Si, Sj) non marqué par « X », conclure que Si et Sj sont équivalents.

Etat	Etat suivant		Sortie	
	X = 0	X = 1	X = 0	X = 1
S0	S1	S2	0	0
S1	S3	S4	0	0
S2	S5	S6	0	0
S3	S0	S0	0	0
S4	S0	S0	1	0
S5	S0	S0	0	0
S6	S0	S0	1	0

Table A-4 - Exemple de réduction avec la méthode d’implication graphique

Soit l'exemple de la Table A-4. On construit le graphe d'implication suivant les étapes de la méthode de réduction (Figure A-1). On obtient à la fin le résultat suivant : les états S3 et S5 sont équivalents et les états S4 et S6 sont équivalents. On conclue ainsi que les états S1 et S2 sont équivalents. Cette méthode appliquée à l'exemple de la Table A-4 réduit le modèle de 7 états à 4 états.

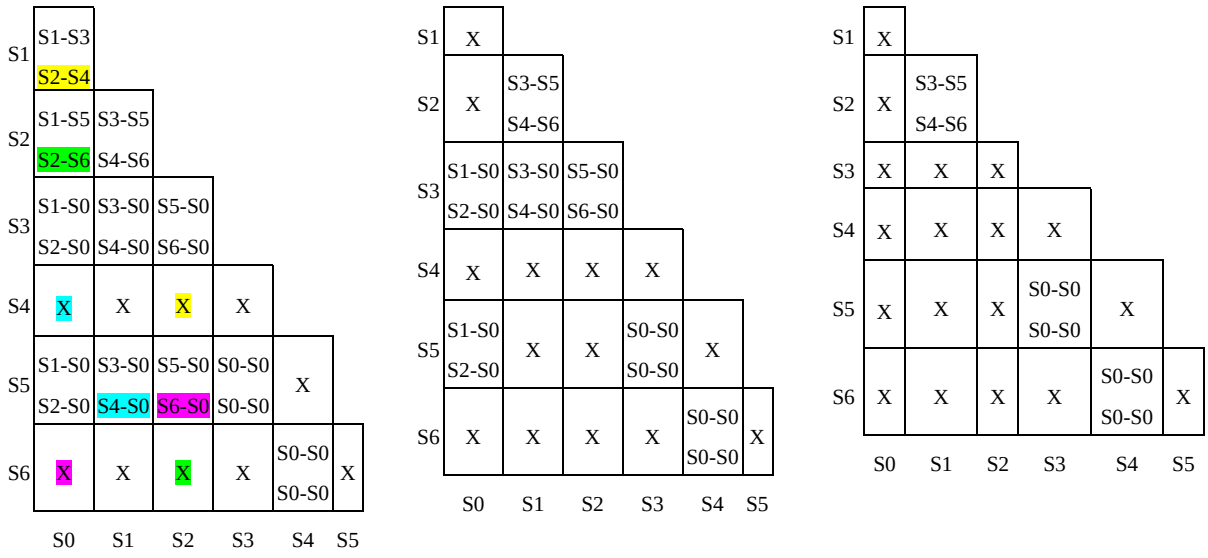


Figure A-1 : Graphe d'implication de la Table A-4

La méthode de partitionnement successif

Cette méthode est un mixage entre les deux autres méthodes. Il s'agit de partitionner l'ensemble des états selon une procédure simple pour déterminer l'équivalence entre les états quelque soit la complexité du modèle. Les étapes de partitionnement successives produisent des partitions de taille de plus en plus petites. Le processus de partitionnement termine lorsque l'étape suivante ne donne plus de partitions plus petites. Tous les états d'une même partition, après la $k^{ème}$ étape sont k équivalents. Tous les états d'une même partition qu'on ne peut plus la décomposer sont équivalents. Les états qui ne sont pas dans la même partition finale ne sont pas équivalents.

Annexe B. Algorithme de réduction d'un graphe d'états à transitions événementielles gardées

Notations :

$\Sigma_{nontraité}$: ensemble des états du modèle non traités

Set_ME : ensemble des macro-états du modèle

ME : Macro-état

ME_C : Macro-état courant

ME_i : Macro-état initial

E : état

E_C : Etat courant

E_i : Etat initial

F: file pour contenir les états du modèle à traiter

Algorithme reduction_MAEG

Début algorithme

Mettre l'ensemble des états du modèle dans $\Sigma_{nontraité}$

Etape 1 : (* Macro-état initial *)

- Créer le macro-état initial (ME_i),
- $E_C = E_i$;
- Ajouter E_i dans ME_i ; (* valeur de sortie de ME_i = valeur de sortie de E_i *)
- ME_C <- ME_i ;
- Ajouter ME_C dans Set_ME

Etape 2 : (* Traitement de tous les états d'un macro-état*)

- **Etape 2-1** : **Si** la valeur de sortie de $ME_C \neq$ la valeur de sortie de E_C **alors**

- ME_C <- ME dont la valeur de sortie = la valeur de sortie de E_C

  (*il existe forcement car si l'état a été déjà enfilé dans la file F c'est que le macro-état a été déjà créé *)

- **Etape 2-2** : **Pour** chaque transition d'évolution t de l'état courant E_C vers l'état suivant E **faire** : (* une boucle pour car on connait le nombre de transitions d'évolution de chaque état E *)

  - **Si** E ∈ ME_C **alors**
    - Créer une transition de ME_C sur lui-même correspondant à la transition t (* self-loop *).
  - **sinon**
    - **Si** il existe ME / $E \in ME$ alors
      - Créer une transition de ME_C vers ME

      sinon debut

      - Ajouter E à ME_C
      - Enfiler l'état E dans F
      - Créer une transition de ME_C sur lui-même correspondant à la transition $t(* \text{ self} - \text{loop } *)$.

      fin sinon
  - **fin pour**

- **Etape 2-3** : **Pour** chaque transition de valeur t de l'état courant E_C vers l'état suivant E faire : (* une boucle car on connait le nombre d'itérations*)

  - **Si** il existe un macro-état dont la valeur de sortie = valeur de sortie de E, soit ME ce macro-état **alors**
    - **Si** E ∈ ME
      - Créer une transition de ME_C vers ME correspondant à la transition t

      sinon debut (* Cas ou E n'est pas déjà dans un macro-état *)

      - Créer ME'
      - Ajouter ME' dans Set_ME
      - Ajouter E à ME'
      - Enfiler l'état E dans F
      - Créer une transition du ME_C vers ME' correspondant à la transition t
      - **fin sinon**

sinon debut (* Cas ou il n'existe pas de macro-état ayant la valeur de sortie de E *)

- Créer un nouveau macro-état ME dont la valeur de sortie = valeur de sortie de E
- Ajouter ME dans Set_ME
- Ajouter E à ME
- Enfiler l'état E dans F
- Créer une transition du ME_C vers ME correspondant à la transition t.
- **fin sinon**

- **fin pour**

- **Etape 2-4** : Retirer E_C de $\Sigma_{nontraité}$
- **Etape 2-5** : **Si** F n'est pas vide et $\Sigma_{nontraité} \neq \varnothing$ **alors**
  - E_C <- Défiler F $\cap \Sigma_{nontraité}$ (* le nouvel état courant devient le prochain état dans la file qui est non traité *)
  - **Aller** à l'étape 2

Etape 3 : (* Règles d'agrégation *)

TQ non vide (Set_ME) **faire**

debut

ME_C <- Retirer ME de Set_ME

- **Etape 3-1** : **Si il** existe ME dans Set_ME tel que MEc possède une transition d'évolution vers un état de ME **alors**

  debut

  - Retirer ME de Set_ME
  - ME_C <- ME_C U ME

  fin

- **Etape 3-2** : **Si** il existe ME appartenant à Set_ME tel que la (valeur_sortie(ME)= la valeur_sortie(ME_C)) (predecesseur(ME)=predecesseur(ME_C)) et ((successeur(ME)=successeur(ME_C)) **alors**

  debut

- Retirer ME de Set_ME
- ME_C <- ME_C U ME

fin

- **Etape 3-3** : **Si** il existe ME appartenant à Set_ME tel que la (valeur_sortie(ME)= la valeur_sortie(ME_C)) (predecesseur(ME)=predecesseur(ME_C)) et (event_pred(ME)= event_pred(ME_C)) et garde_pred(ME)= garde_pred(ME_C)) **alors**

  debut

  - Retirer ME de Set_ME
  - ME_C <- ME_C U ME

  fin

fin (* étape 3*)

Fin algorithme

Remarque :

predecesseur(ME) -> donne ME'un élément de Set_ME qui est tel qu'il existe une transition de ME' vers ME

successeur(ME) -> donne ME'un élément de Set_ME qui est tel qu'il existe une transition de ME vers ME'

valeur_sortie(ME) -> donne la valeur de la sortie correspondant de ME

event_pred(ME) -> donne l'événement associé à une transition arrivant sur ME

garde_pred(ME) -> donne la garde associée à une transition arrivant sur ME

Annexe C. Modèles comportementaux d'une fonction logique

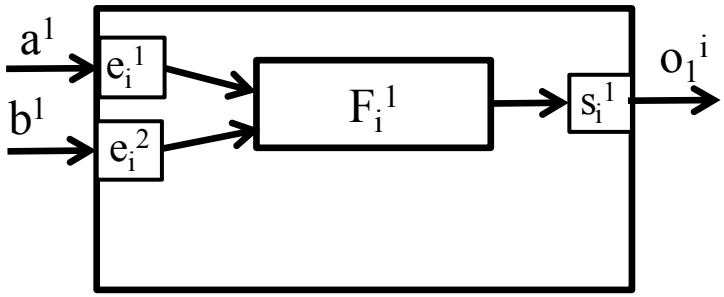


Figure C-1 - Ligne fonctionnelle réduite aux éléments d'une carte fille

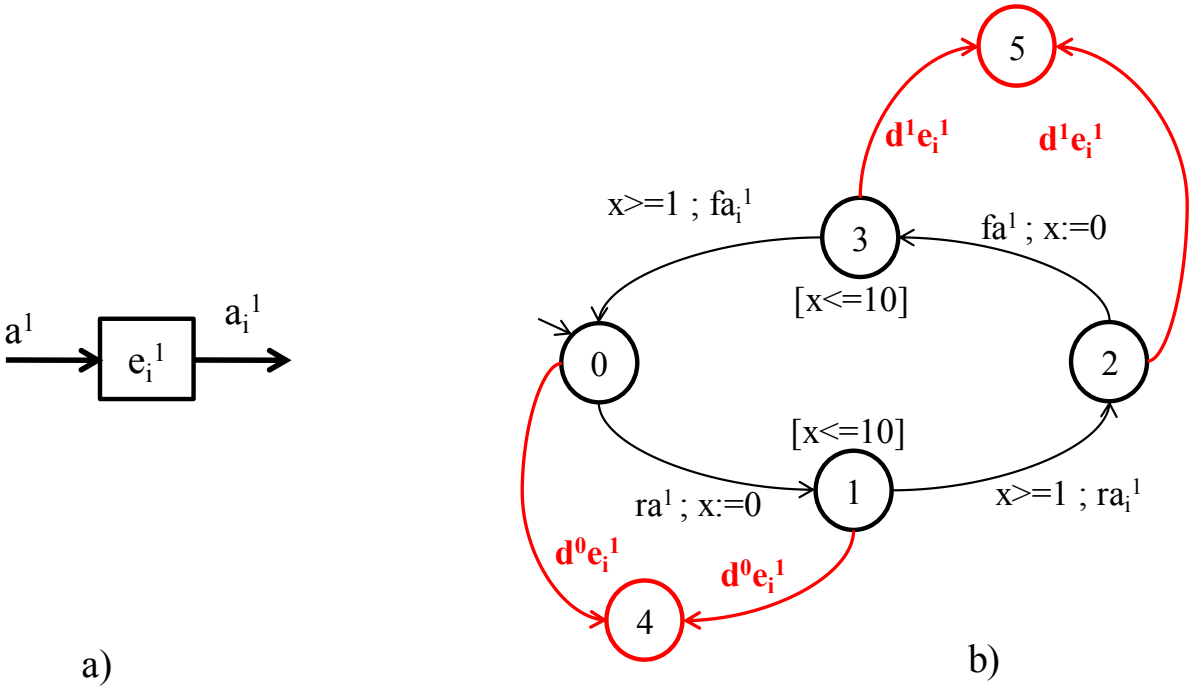


Figure C-2 : Modèle Comportemental du port d'entrée e_i^1

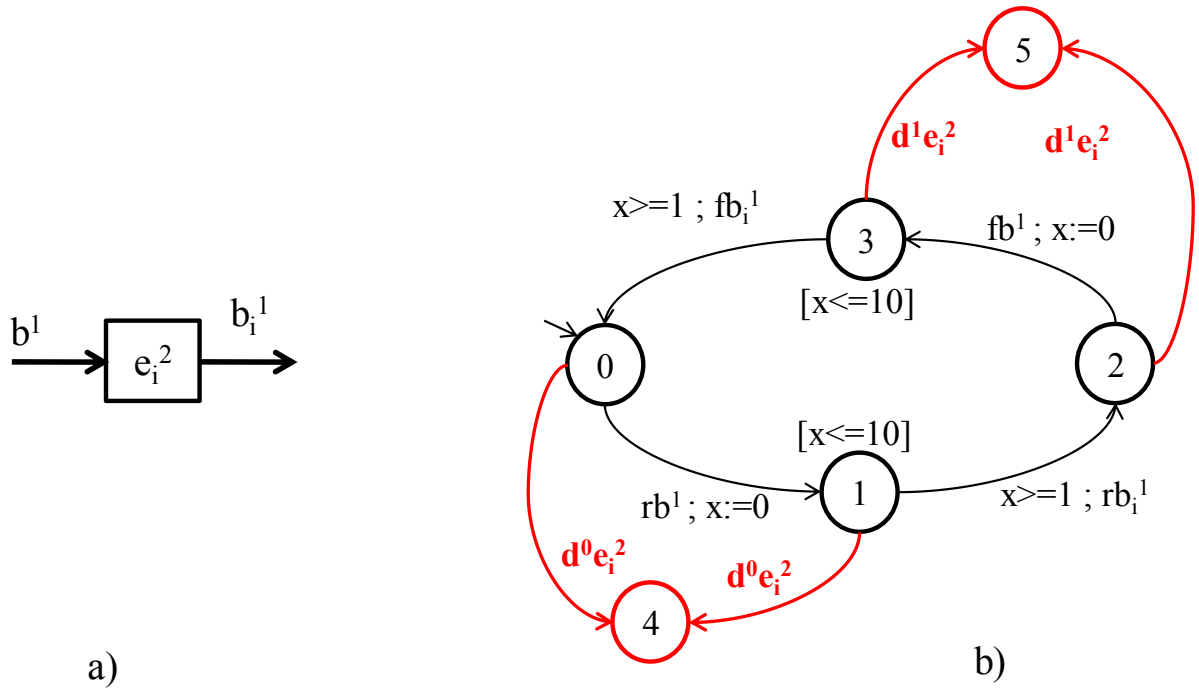


Figure C-3 : Modèle Comportemental du port d'entrée e_i^2

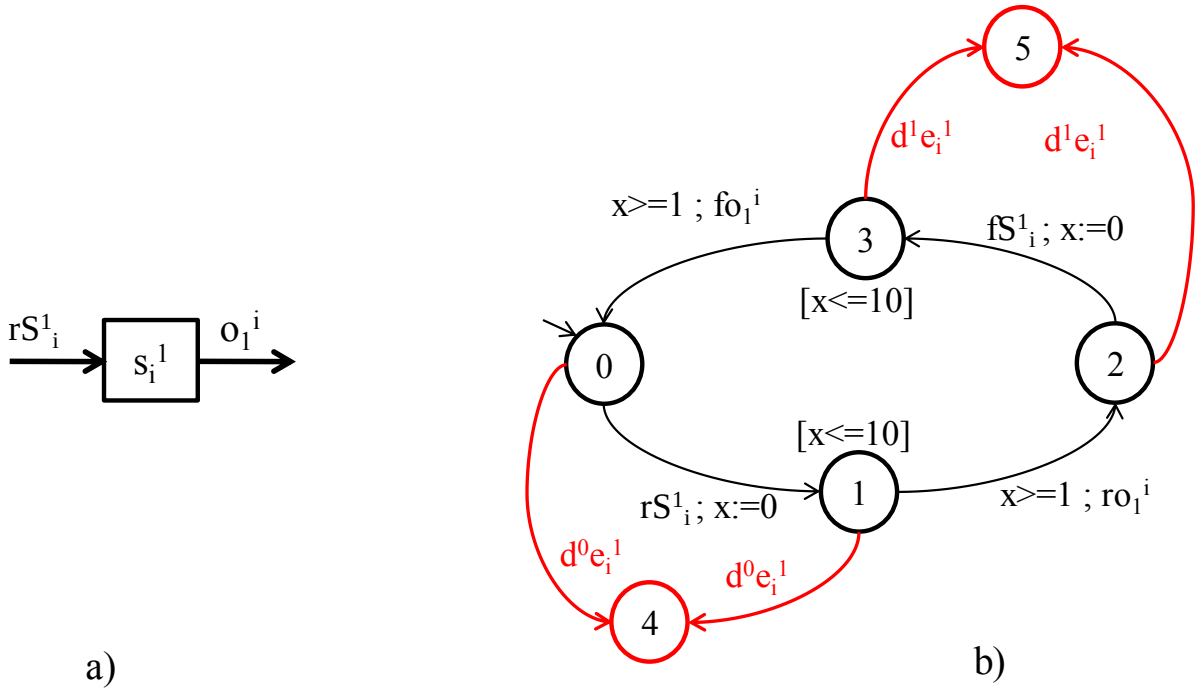


Figure C-4 : Modèle Comportemental du port d'entrée s_i^1

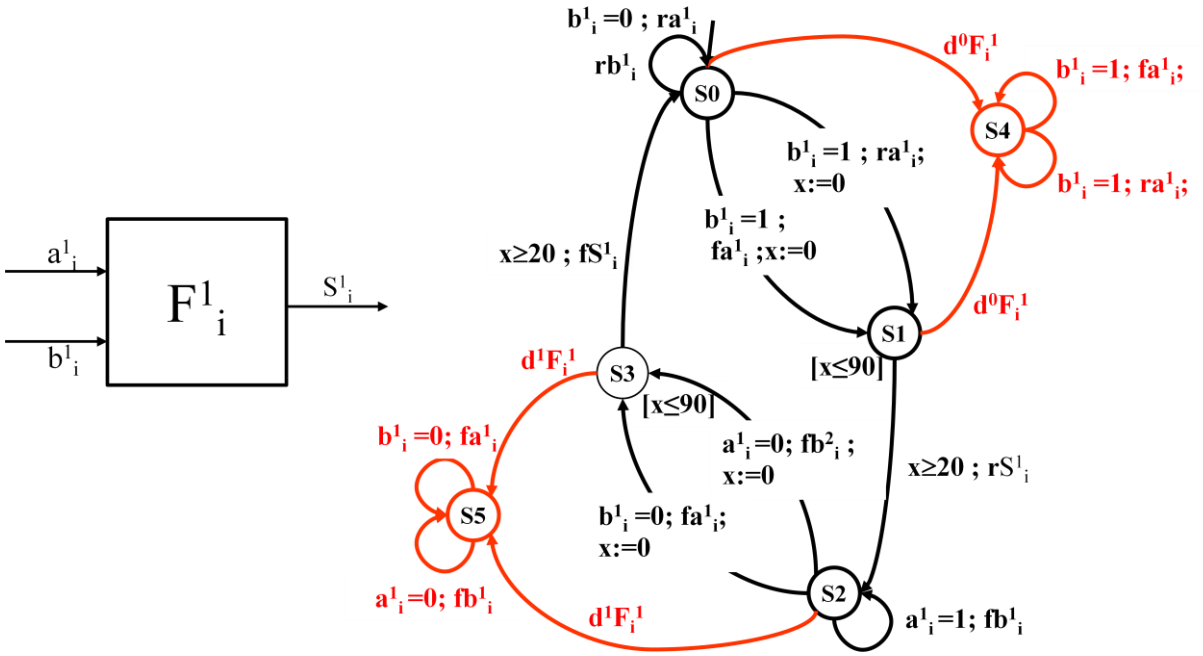


Figure C-5 : Modèle Comportemental de la fonction logique F_i^1

Diagnosticabilité modulaire appliquée au Diagnostic en ligne des Systèmes Embarqués Logiques

Résumé : Aujourd'hui, les systèmes embarqués sont de plus en plus utilisés pour contrôler les systèmes complexes. Dans ce travail de thèse, nous nous intéressons aux systèmes embarqués critiques utilisés pour la commande de systèmes de transport comme les systèmes ferroviaires. Le but de ce travail est de permettre la conception de systèmes tolérants aux fautes pour le contrôle-commande des systèmes de transport. Nous proposons une nouvelle approche de modélisation des systèmes embarqués temporisés pour le diagnostic de leurs fautes. Elle est basée sur une décomposition structurelle du système et sur une extension de la diagnosticabilité modulaire au contexte des systèmes temporisés. On distingue deux approches de base pour le diagnostic de fautes des SEL, une approche basée sur les diagnostiqueurs et une approche basée sur les signatures temporelles causales (STC). La principale limite de l'approche diagnostiqueur réside dans la gestion de l'explosion combinatoire. Dans ce travail, notre verrou principal est de combattre cette limite. Nous proposons une nouvelle méthode basée sur l'ingénierie par les modèles pour le diagnostic des systèmes embarqués critiques. D'autre part, la limite majeure de l'approche STC est la garantie de la cohérence d'une base de STC. Un deuxième niveau de difficulté réside dans l'interprétation des événements en entrée du système de diagnostic dans le cadre de l'hypothèse de défaillances multiples. Dans ce travail, nous proposons deux méthodes différentes pour la vérification de la cohérence d'une base de STC et nous proposons un algorithme d'interprétation basé sur le concept de monde qui garantit la correction du diagnostic.

Mots-clefs : Diagnostic des fautes, Diagnostiqueur, Diagnosticabilité, Systèmes embarqués, Model-checking, SEL, Observateur, STC

Modular diagnosability applied to on line Diagnosis of Digital Embedded System

Abstract : Today, embedded systems are increasingly used to control complex systems. In this thesis, we are interested in critical embedded systems used for the control of transport systems such as railway systems. The aim of this work is to enable the design of fault-tolerant systems for the control of transport systems. We propose a new timed embedded systems modeling approach to diagnose their faults. It is based on a structural decomposition of the system and extension of modular diagnosability to timed systems. In DES, there are two basic approaches for diagnosis: diagnoser based approach and (Causal Temporal Signature (CTS)) based approach. The major limitation of diagnoser approaches rely in the management of the combinatorial explosion related to the formalism of automata. In this work, our main lock is to combat this limit. We propose new engineering models based method for the diagnosis of critical embedded systems. On the other hand, the major limitation of chronicles approach is first to be able to guaranty the consistency of a database. A second level of difficulty is in interpreting some sequences of events at the input of the diagnostic system under the hypothesis of multiple failures. In this work, we propose two different methods to verify the consistency of a set of CTS and we propose an interpretation algorithm based on a world concept which guarantees the correctness of the diagnosis.

Keywords: Fault Diagnosis, Diagnoser, Diagnosability, Embedded systems, Model-checking, DES, Observer, CTS