Gestion des aléas dans un système multi-robots

Valentin Bouziat

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)*

Présentée et soutenue le *18/12/2020* **par :**

Valentin Bouziat

Gestion des aléas dans un système multi-robots

JURY

François Vernadat	Professeur d'Université	Président du Jury
Philippe Dague	Professeur d'Université	Membre du Jury
Gregor Goessler	Directeur de Recherche	Membre du Jury
Janan Zaytoon	Professeur d'Université	Membre du Jury
Alban Grastien	Chargé de Recherche	Membre du Jury
Louise Travé-Massuyès	Directeur de Recherche	Membre du Jury
Xavier Pucel	Ingénieur de Recherche	Membre du Jury
Stéphanie Roussel	Ingénieur de Recherche	Membre du Jury

École doctorale et spécialité :
EDSYS : Informatique 4200018

Unité de Recherche :
Office National d'Etudes et de Recherches Aérospatiales - DTIS

Directeur(s) de Thèse :
Louise Travé-Massuyès, *Xavier Pucel* et *Stéphanie Roussel*

Rapporteurs :
Philippe Dague et *Gregor Goessler*

Remerciements

Tout d'abord, je remercie Philippe Dague et Gregor Goessler de l'honneur qu'ils m'ont fait en acceptant d'être rapporteurs de cette thèse. Merci à eux pour l'assiduité de leur relecture et pour la pertinence de leur rapport qui ont été bénéfiques à la version finale de ce mémoire. J'exprime ma gratitude à Monsieur François Vernadat pour avoir accepté la présidence du jury. Je remercie tous les membres du jury notamment Alban Grastien et Janan Zaytoon qui ont accepté d'assister à la soutenance et dont les nombreuses questions lors de ma présentation ont montré un intérêt certain pour les travaux présentés dans ce document.

Je tiens à remercier Louise Travé-Massuyés, qui m'a encadré tout au long de cette thèse. Merci à elle pour le partage de son expérience et de ses précieux conseils lors de nos nombreuses réunions. J'adresse également mes remerciements à Xavier Pucel et Stéphanie Roussel, tous deux ingénieurs de recherche qui m'ont accueilli au sein du DTIS de l'ONERA et avec qui les discutions furent toujours enrichissantes. Merci à eux trois pour leur disponibilité, leur rigueur et pour leurs nombreuses relectures et corrections des différents travaux. J'ai énormément appris en travaillant à leurs côtés et le présent document est le fruit d'une collaboration de plus de trois ans avec eux.

Je tiens aussi à remercier Yannick Pencolé et Charles Lesire qui ont été membres de mon comité de suivi de thèse. Merci encore à eux de m'avoir conseillé les enseignements à l'INSA de Toulouse dans lesquels je continue de m'épanouir. Je remercie aussi les membres des différentes communautés de chercheurs que j'ai eu la chance de rencontrer lors de conférences, remerciements particuliers aux membres des communautés DX et PFIA.

Remerciements également à tous les personnels des différents établissements que j'ai fréquentés au cours de ces trois ans tels que l'ONERA, le Laas-CNRS, l'école doctorale EDSYS ainsi que l'ISAE-SUPAERO. Je tiens aussi à remercier tous les étudiants et post-doctorants de ces différents établissements pour les nombreux moments passés non loin de la machine à café, merci encore pour la bonne ambiance.

Un grand merci enfin à Sébastien, avec qui j'ai partagé le bureau et bien plus encore pendant ces trois ans et à qui je souhaite une pleine réussite pour mener à terme son doctorat.

Résumé

Les progrès de l'Intelligence Artificielle permettent aux systèmes de devenir plus autonomes. Il est primordial de comprendre leur comportement, notamment pour que ces systèmes soient acceptés dans l'environnement dans lequel ils évoluent. Dans cette thèse, nous nous intéressons aux systèmes modélisés sous la forme de systèmes à événements discrets partiellement observables et pour lesquels on cherche à fournir une estimation de l'état du système à partir du comportement observé de celui-ci.

Nous nous appuyons sur une approche d'estimation, basée sur un ensemble de préférences conditionnelles, qui consiste à ne garder qu'un seul diagnostic à chaque pas de temps. Cela permet notamment de satisfaire les contraintes opérationnelles liées aux applications robotiques (ressources de calcul ou de mémoire limitées) et faciliter la prise de décision. Cependant, cette approche peut mener à des scénarios d'impasse, c'est-à-dire des scénarios dans lesquels l'estimateur n'est plus en mesure d'expliquer l'observation du système en restant cohérent avec les estimations des états précédents.

L'objectif de la thèse est d'analyser, en mode hors-ligne, les stratégies d'estimation à état unique pour un système donné. La première contribution de la thèse consiste à détecter les scénarios d'impasse d'une taille fixe pour un système et une stratégie d'estimation données. Deuxièmement, pour un scénario d'impasse donné, nous calculons un ensemble minimal de préférences de la stratégie d'estimation à l'origine de l'impasse. Finalement, nous caractérisons la propriété d'estimabilité à état unique d'un système, qui consiste en l'existence d'une stratégie d'estimation pour celui-ci qui ne mène à aucun scénario d'impasse. L'ensemble des contributions de la thèse a donné lieu à des développements de prototypes basés sur des techniques Model-Checking et SAT et la réalisation d'expérimentations sur des jeux de données représentatifs du monde réel.

Abstract

Advances in Artificial Intelligence allow systems to become more autonomous. It is essential to understand their behavior, in particular so that these systems are accepted in the environment in which they evolve. In this thesis, we are interested in systems modeled as discrete event systems that are partially observable and for which we seek to provide an estimate of the state of the system from its observed behavior.

We rely on an estimation approach, based on a set of conditional preferences, which consists in keeping only one diagnosis at each time step. This allows in particular to satisfy the operational constraints related to robotic applications (limited computing or memory resources) and to facilitate decision making. However, this approach can lead to dead-end scenarios, i.e. scenarios in which the estimator is no longer able to explain the observation of the system while remaining consistent with the estimates of previous states.

The objective of the thesis is to analyze, in offline mode, single-state estimation strategies for a given system. The first contribution of the thesis consists in detecting dead-end scenarios of a given size for a system and an estimation strategy. Second, for a given dead-end scenario, we compute a minimum set of preferences of the estimation strategy at the origin of the dead-end. Finally, we characterize the single-state estimability property of a system, which consists in the existence of an estimation strategy for the system that does not lead to a dead-end scenario. All the contributions of the thesis have led to the development of prototypes based on Model-Checking and SAT techniques and to experiments on representative real-world data sets.

Table des matières

Chapitre 1

Introduction

Cette étude s'inscrit dans le domaine du diagnostic des systèmes modélisés sous forme de système à évènements discrets (noté SED). La problématique associée à ce domaine consiste à fournir une estimation de la présence d'aléas à partir du comportement observé d'un système. Plus généralement, on considère que le diagnostic des systèmes à évènements discrets consiste à estimer l'état d'un système à partir d'informations partielles que l'on possède sur celui-ci. On parle d'informations partielles car dans la plupart des cas, on ne peut directement observer l'état d'un système notamment lorsque celui-ci évolue de manière autonome. Si par exemple on s'intéresse à l'estimation de l'état d'un robot d'exploration, c'est-à-dire évaluer quels composants sont ou ne sont pas en mesure de fonctionner normalement, on ne peut se fier qu'aux informations perçues par un centre de contrôle ou un opérateur humain : généralement des valeurs issues des différents capteurs.

Pour mener à bien ce genre de missions, la modélisation du système a une place importante. On modélise de manière mathématique l'état des différents composants du système, la manière dont celui-ci se comporte au cours de la mission ainsi que les différents éléments qui peuvent être observés. Lorsqu'on cherche à effectuer le diagnostic d'un SED, il est nécessaire de prendre en compte la nature même d'un tel système. C'est pourquoi plusieurs études ont porté sur la définition de propriétés relatives à ce domaine afin de garantir un diagnostic cohérent avec certaines exigences. Par exemple, la propriété de diagnosticabilité d'un SED introduite par Sampath en 1994, indique qu'au cours de la phase de diagnostic, la présence ou l'absence d'une faute peut être déduite avec certitude après un certain nombre d'observations.

Cependant, les approches classiques de la littérature proposent des solutions au diagnostic de systèmes à évènements discrets difficilement applicables au monde réel car celles-ci rencontrent des problèmes de passage à l'échelle. En effet, dans ces approches, on considère à chaque étape discrète l'ensemble des états dans lequel le système est susceptible de se trouver. En plus de poser des problèmes de passage à l'échelle, cela a pour conséquence de fournir une estimation non déterministe sur l'état du système.

Prenons le cas de la robotique autonome. On considère par exemple un module de diagnostic qui doit fournir une information précise sur l'état du système à un opérateur humain ou à un planificateur afin que ce dernier puisse prendre une décision ou entreprendre une action. Cette éventuelle action est directement dépendante de l'état estimé par le module de diagnostic et il n'est pas souhaitable d'avoir un ensemble d'états possibles à considérer.

Aussi, la plupart des approches supposent que pour rester cohérent avec la dynamique du système, il est nécessaire de mémoriser l'ensemble des états antérieurs. Pour des architectures autonomes, cela n'est généralement pas souhaitable du fait de la taille limitée de la mémoire embarquée.

Afin de répondre à ces problématiques, les auteurs de [Pralet et al., 2016], ont proposé une méthode de diagnostic basée sur l'utilisation de préférences conditionnelles. Cela a permis non seulement d'être capable de dégager un seul état de croyance lors du diagnostic

mais aussi de pallier au problème de passage à l'échelle. De plus, le processus d'estimation proposé est incrémental, c'est à dire que l'estimation de l'état courant dépend uniquement de l'état précédemment estimé et d'une nouvelle observation reçue. Cela a donc permis de proposer un formalisme peu gourmand en mémoire puisqu'on ne mémorise pas tous les états antérieurs.

Toutefois, cette approche peut mener à des problèmes lors de sa mise en place. En effet, garder un seul état en mémoire lors du diagnostic peut parfois conduire à une situation dans laquelle l'observation produite par le système n'est plus cohérente avec l'état précédemment estimé, on parle d'une impasse.

C'est ce qui justifie l'objet de cette étude. Elle consiste à étudier la possibilité d'estimer l'état d'un SED en gardant un seul état de croyance à chaque étape discrète tout en conservant la cohérence avec la dynamique de ce système au cours de l'estimation.

La première partie de ce document dresse un état de l'art en présentant les notions utiles à la compréhension, les outils utilisés ainsi qu'un formalisme pour l'estimation à état unique. Le deuxième chapitre de ce document présente le diagnostic des systèmes, le formalisme des systèmes à évènements discrets avec les notations et propriétés associées. Différents outils mathématiques et informatiques permettant à la fois la définition et l'analyse de tels systèmes y sont également évoqués. Le troisième chapitre présente le formalisme d'estimation pour des systèmes autonomes utilisé tout au long de ce document. On y définit la notion d'état, d'observation ainsi que la manière dont on modélise le comportement du système. La notion d'estimateur à état unique y est définie pour effectuer la tâche du diagnostic. Ce formalisme repose sur l'utilisation de préférences conditionnelles et est inspiré de [Pralet et al., 2016]. Ce formalisme est cependant sujet à certaines limites, notamment les scénario d'impasse, dont l'étude et l'analyse font l'objet de la seconde partie.

La deuxième partie de ce document décrit les différentes contributions apportées. Dans le quatrième chapitre, on s'intéresse à l'étude de l'existence de scénarios d'impasse dans l'estimation à partir d'un SED et d'une théorie de préférences conditionnelles. L'objectif est d'être capable de détecter dès la phase de conception si le module de diagnostic est susceptible de rencontrer une impasse. Pour cela, une modélisation du problème inspirée de la méthode du Twin-Plant, permettant originalement de vérifier la diagnosticabilité, est proposée. Ensuite, ce modèle est encodé en langage *Electrum*, un outil de model-checking particulièrement adapté, d'une part car celui-ci permet le traitement de systèmes avec des dynamiques temporelles, d'autre part car son expressivité est suffisamment riche pour représenter les préférences conditionnelles. Ces travaux sont présentés dans [Bouziat et al., 2018] et fourni en annexe A.

Dans le chapitre 5, on s'intéresse à l'élimination d'un scénario d'impasse par modification de la stratégie d'estimation. L'objectif est d'être capable d'incriminer les préférences conditionnelles qui en sont responsables. On introduit alors la notion de préférence relaxée, afin de permettre plusieurs scénarios d'estimation possibles. Pour cela, le méta-diagnostic des préférences est mis en place afin de mettre en évidence les différents sous-ensembles de préférences, qui lorsqu'elles sont relaxées, élimine le scénario d'impasse. Ces travaux ont fait l'objet d'une publication [Bouziat et al., 2019a], qui est fournie en annexe B.

Le sixième chapitre présente l'étude de la possibilité de construire une fonction d'estimation sans impasse à partir des spécifications d'un système. En effet, dans certains cas, le méta-diagnostic des préférences ne suffit pas à éliminer tous les scénarios d'impasse suivant la nature du système. Pour caractériser des systèmes pouvant être estimés avec une théorie de préférences conditionnelles sans rencontrer d'impasse, une nouvelle propriété s'appliquant aux systèmes à évènements discrets est proposée. L'idée est de pouvoir statuer, dès la conception d'un système, si un système peut être estimé grâces à une théorie de préférences ou plus généralement en ne gardant qu'un unique état estimé au cours du temps. Deux méthodes algorithmiques permettant de vérifier la propriété d'*estimabilité à état unique* sont décrits et testés sur différents jeux de données. On s'intéressera aussi à

l'ajout de contraintes de correction dans l'estimateur. Ces éléments sont présentés dans [Bouziat et al., 2019b] et [Roussel et al., 2020], qui sont fournis en annexes C et D de ce document.

Contexte : estimation pour la robotique autonome

Les systèmes autonomes prennent une place de plus en plus importante dans nos environnements quotidiens et imposent des spécifications particulières. L'autonomie décisionnelle est un point essentiel dans la robotique autonome. De plus ces systèmes autonomes évoluent généralement dans des environnements incertains où la présence d'aléas doit être prise en compte au vue de la criticité des tâches à réaliser (par exemple pour des robots d'exploration sous-marine ou spatiale). Dans ce document on s'intéresse aux systèmes autonomes modélisés sous forme de systèmes à évènements discrets. Ce formalisme permet notamment de définir de manière logique et mathématique le comportement du système, les évènements pouvant survenir et parfois la nature de l'environnement.

Cette section motive l'estimation à état unique, pour les systèmes à évènements discrets. On étudiera dans un premier temps le diagnostic tel qu'il est décrit et utilisé dans les approches classiques, sans restriction du nombre d'états estimés. Puis, on s'intéressera au principe d'estimation à état unique pour de tels systèmes. La figure 1.1 illustre la boucle de décision d'une architecture autonome pour une mission mono ou multi-robots.

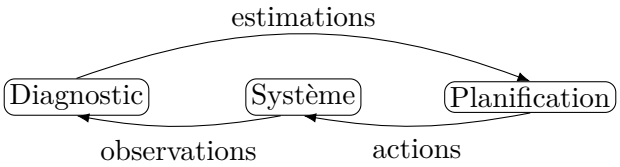


FIGURE 1.1 – Boucle de décision d'une architecture autonome

Afin de modéliser les différents acteurs impliqués dans la réalisation d'une mission autonome, on distingue trois modules : le système, le module de planification et le module de diagnostic. Le système est considéré comme une « boite noire » qui reçoit des actions à effectuer (des commandes), agit sur lui-même ou sur son environnement et produit des observations, généralement des informations issues de capteurs etc.

Le rôle du module de diagnostic est d'estimer, à partir des observations reçues du système, si des aléas sont survenus et de les identifier. À partir des observations fournies par le système, le module de diagnostic estime les états possibles du système. Cependant, pour une même observation, on ne peut souvent pas déterminer l'état du système sans ambiguïté. En effet le système n'est souvent observable que partiellement. Par exemple pour des robots d'exploration sous-marine ou spatiale, on ne peut se baser que sur des informations transmises par communication longue distance puisqu'on ne peut observer directement le système. Le module de diagnostic effectue donc une estimation de l'état du système à partir des informations partielles reçues.

Intervient ensuite le module de planification qui, à partir des informations transmises par le module de diagnostic, décide des actions à entreprendre. Ce module peut être complètement automatisé ou à l'inverse être tout simplement un opérateur humain. Si le module de diagnostic estime qu'un composant du système est défaillant, un propulseur par exemple, on peut envisager la mise en place d'un plan secondaire ; la mise en route d'un propulseur auxiliaire. Le diagnostic de l'aléa à donc un impact direct sur le déroulement de la mission et chaque aléa entraîne une action différente comme l'illustre l'exemple suivant :

Exemple 1 (Robot d'acquisition autonome). *On considère un robot autonome équipé d'un appareil photo ayant pour mission d'effectuer des clichés et d'explorer différents endroits. Il peut arriver plusieurs aléas lors de la mission que le module de diagnostic doit*

estimer. Par exemple, si le robot envoie une photo floue, cela peut signifier que :

- *l'appareil photo est en panne*
- *la communication est bruitée*
- *le robot est bloqué par un obstacle*

La panne de l'appareil photo peut par exemple mener le module de planification à continuer la mission d'exploration sans prendre de clichés, celui-ci peut décider d'attendre lorsque la communication est bruitée ou encore envisager un rapatriement lorsque le robot est bloqué.

Chaque aléa estimé par le module de diagnostic va donc entraîner une action différente entreprise par le module de planification.

Focalisons nous sur le module de diagnostic. Dans les approches classiques, on considère que le module de diagnostic peut calculer plusieurs états candidats au diagnostic c'est-à-dire tous les états dans lesquels le système peut se trouver. Cependant, calculer tous les diagnostics possibles présente plusieurs limites.

Premièrement, à chaque étape discrète, le nombre de diagnostics possibles peut être important, voire atteindre le nombre d'états du système. Cela peut poser problème lorsqu'on cherche à effectuer le diagnostic de systèmes complexes, le nombre de diagnostics possibles peut croître très rapidement au cours de l'exécution et donc nécessiter un espace mémoire conséquent. Lors des missions robotiques autonomes, on doit souvent se contenter d'un espace mémoire limité. Calculer un nombre de diagnostics restreint à chaque étape s'avère être une solution à ce problème de mémoire limitée.

Deuxièmement, si le module de diagnostic transmet plusieurs états dans lesquels le système peut se trouver, il devient difficile pour le module de planification de prendre une décision à partir de nombreux états possibles qui peuvent nécessiter des actions incompatibles. Le cadre de l'estimation à état unique permet de résoudre ce problème : au lieu de transmettre tous les candidats possibles au module de planification, on n'en transmet qu'un seul afin de simplifier le processus de planification.

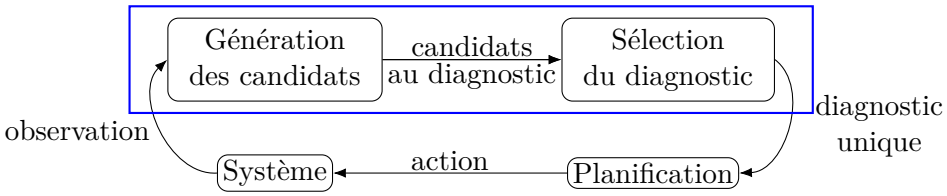


FIGURE 1.2 – Boucle de décision avec sélection d'un unique diagnostic

L'approche adoptée ici divise conceptuellement la phase de diagnostic en deux étapes (voir figure 1.2) : d'abord le calcul des candidats au diagnostic, puis la sélection d'un unique état parmi ces candidats. Certains travaux [Fabre and Jezequel, 2010] se basent sur les probabilités et sélectionnent l'état dans lequel le système a la probabilité la plus forte de se trouver. L'approche étudiée ici envisage le calcul d'un unique état estimé au sein du même module. Dans cette approche, le diagnostic unique est sélectionné à partir d'un modèle exprimant les préférences d'un expert. Cela permet de baser le processus de raisonnement non plus sur des probabilités, qui peuvent être difficiles à obtenir, mais sur des contraintes logiques précisément définies par un expert du système.

Première partie

État de l'art

Chapitre 2

Diagnostic des systèmes

Dès lors qu'on s'intéresse à des missions multi-robots, il est important d'avoir une représentation formelle du système conçu, c'est-à-dire un modèle. Il existe un domaine de la littérature appelé diagnostic à base de modèle qui regroupe deux communautés de scientifiques.

La première, FDI (Fault Detection and Identification) est issue de l'automatique. Ses fondements reposent sur la théorie de la commande, la théorie des systèmes et la décision statistique. La seconde, DX est issue de l'intelligence artificielle. Les concepts de base du diagnostic à base de modèle pour cette communauté se retrouvent notamment dans les travaux de [Reiter, 1987] et [deKleer and Williams, 1987]. Des théories générales sont proposées afin de déterminer dans un cadre logique les différences entre un comportement nominal du système et le comportement observé afin d'identifier les composants responsables. Des similarités existent entre ces deux approches et ont déjà fait l'objet d'études comparatives [Ceballos et al., 2005],[Cordier et al., 2014]. Notre travail étant ancré dans le domaine de l'intelligence artificielle, nous nous limitons aux approches de la communauté DX et porterons un accent particulier sur les « systèmes à évènements discrets ». Généralement, lorsqu'on souhaite superviser un système, c'est-à-dire surveiller et/ou agir sur son évolution, on ne peut observer directement l'état dans lequel se trouve ce système. On parle alors de système partiellement observable.

La première partie de ce chapitre rappelle la théorie générale du diagnostic pour des systèmes partiellement observables telle que décrite dans la littérature. On s'intéresse dans ces travaux plus particulièrement au diagnostic de systèmes à évènements discrets, les notions essentielles sur les SED sont donc rappelées en deuxième partie. La troisième partie présente les notions et propriétés associées au diagnostic des SED. Nous verrons que certaines de ces propriétés peuvent être vérifiées via model-checking, le sujet de la quatrième partie. Les logiques temporelles, permettant la définition de systèmes et de propriétés, ainsi que les solveurs SAT, permettant leur encodage, y seront abordés.

2.1 Le diagnostic à base de cohérence

Raymond Reiter [Reiter, 1987] propose une théorie générale du diagnostic à base de modèle connue sous le nom de diagnostic à base de cohérence, en généralisant les travaux de [deKleer and Williams, 1987] et [Genesereth, 1984]. On abstrait d'abord le comportement du système, ses différents composants et la manière dont celui-ci est observé. Ensuite, à partir du comportement observé (si celui-ci est différent du comportement attendu), on cherche à déterminer les composants du système qui en seraient responsables.

2.1.1 Élements de la théorie de Reiter

Reiter définit en premier lieu un système de la manière suivante :

Définition 1 (Système [Reiter, 1987]). *Un système est une paire* $(SD, COMPS)$ *dans laquelle :*

- *SD est un ensemble de règles de logiques du premier ordre qui donne la description du système,*
- *COMPS est un ensemble de constantes représentant les composants du système.*

Le diagnostic de système réel induit forcément une observation partielle du système. On ne peut équiper certains composants de capteurs qui émettent un signal de sortie. Afin de prendre en compte cet aspect, Reiter propose de définir un système et la manière dont on peut l'observer.

Définition 2 (Système partiellement observable [Reiter, 1987]). *Un système partiellement observable* $(SD, COMPS, OBS)$ *est un système* $(SD, COMPS)$ *dans lequel* OBS*, les observations du système, est un ensemble de phrases de logiques du premier ordre.*

Pour décrire un diagnostic, on introduit un prédicat $AB(c)$ où $c \in COMPS$, qui indique qu'un composant présente un comportement anormal. On peut donc supposer que lorsque le système suit un comportement nominal, tous ses composants ont un comportement normal et les observations ne rentrent pas en conflit avec ces deux hypothèses. Ainsi, la phrase logique suivante est cohérente :

$$SD \cup \{\neg AB(c) | c \in COMPS\} \cup OBS$$

Dès lors que le comportement observé rentre en conflit avec la description du système et l'hypothèse que tous les composants sont dans une configuration normale, on peut identifier quels composants sont anormaux : c'est le diagnostic. On définit alors un diagnostic comme un ensemble minimal de composants pour lesquels, lorsque le prédicat AB est vrai, on retrouve la cohérence entre la description du système et son comportement observé.

Définition 3 (Diagnostic [Reiter, 1987]). $\Delta \subseteq COMPS$ *est un diagnostic pour* $(SD, COMPS, OBS)$ *si et seulement si* Δ *est un ensemble minimal tel que :*

$$SD \cup OBS \cup \{\neg AB(c) | c \in COMPS - \Delta\} \text{ est cohérent}$$

Dans le but de calculer les diagnostics à partir du modèle d'un système et de ce qui est observé, Reiter définit les conflits.

Définition 4 (Conflits [Reiter, 1987]). *Un conflit pour* $(SD, COMPS, OBS)$ *est un ensemble de composants* $\{c_1, , .., c_k\} \subseteq COMPS$ *tel que :*

$$SD \cup OBS \cup \{\neg AB(c_1), ..., \neg AB(c_k)\} \text{ est incohérent}$$

On peut ainsi dire qu'un conflit pour $(SD, COMPS, OBS)$ est un conflit minimal si et seulement si il n'existe aucun sous-ensemble de celui-ci qui est aussi un conflit pour $(SD, COMPS, OBS)$. Il est donc possible de calculer les diagnostics à partir des propositions suivantes :

Proposition 1 ([Reiter, 1987]). $\Delta \subseteq COMPS$ *est un diagnostic pour* $(SD, COMPS, OBS)$ *si et seulement si* Δ *est un ensemble minimal tel que* $COMPS - \Delta$ *n'est pas un conflit pour* $(SD, COMPS, OBS)$.

Proposition 2 ([Reiter, 1987]). $\Delta \subseteq COMPS$ *est un diagnostic* $(SD, COMPS, OBS)$ *si et seulement si* Δ *est un ensemble couvrant minimal pour la collection de conflits pour* $(SD, COMPS, OBS)$.

En compilant tous les ensembles couvrants minimaux pour $(SD, COMPS, OBS)$ à partir de ce théorème on obtient tous les diagnostics possibles. Reiter a donc introduit les bases d'un raisonnement logique permettant de formaliser le diagnostic des systèmes. Cependant, le formalisme proposé ne tient pas compte des phénomènes temporels. En effet, le raisonnement est effectué avec un point de vue statique du système en considérant toutes les observations à un instant donné. On peut s'intéresser au diagnostic de systèmes dont le comportement varie au cours du temps. Pour cela, on peut aussi modéliser les systèmes sous forme de systèmes à évènement discrets, modèle mathématique permettant de décrire des systèmes dont l'état varie lorsque différents évènements surviennent.

2.1.2 Théorie du Méta-diagnostic

Les auteurs de [Belard et al., 2011] proposent une théorie du méta-diagnostic de système de diagnostic. L'idée est d'appliquer un algorithme de diagnostic sur un diagnostiqueur lorsque celui-ci est défaillant. Pour ce faire, ils définissent la notion de méta-système afin de représenter de manière théorique un outil de diagnostic. Un méta-système est composé de :

— $M\text{-}SD$: la description du méta-système, un ensemble de phrases de logique du premier ordre,

— $M\text{-}COMP$: les composants du méta-système, un ensemble de constantes booléennes,

— $M\text{-}OBS$: les méta-observations du système, un ensemble de phrases de logique du premier ordre.

Le problème du méta-diagnostic consiste à retrouver la cohérence, lorsque celle-ci est impossible avec tous les méta-composants dans un état normal, entre le comportement observé de l'outil de diagnostic $\mathtt{M-OBS}$ et sa description $\mathtt{M-SD}$. Pour le résoudre, on peut appliquer l'algorithme de diagnostic à base de cohérence de [Reiter, 1987]. On recherche alors un méta-diagnostic minimum, c'est-à-dire le plus petit ensemble de composants du méta-système qui lorsqu'ils sont dans un état anormal permettent de rétablir la cohérence entre la description du méta-système et les méta-observations. Nous utiliserons la théorie du méta-diagnostic dans le chapitre 5.

2.2 Les systèmes à évènements discrets

Les systèmes à évènements discrets, les formalismes et méthodes existantes pour les représenter et les analyser ont de nombreuses applications. On peut par exemple les utiliser dans la construction de programmes informatiques, de protocoles de communication ou encore en biologie. On considère qu'un SED admet un espace d'état fini, ne peut se trouver que dans un seul état à la fois et qu'un changement d'état est appelé une transition. Un système à évènements discrets satisfait deux propriétés :

— l'espace d'état est un ensemble discret
— la transition d'un état à un autre est déclenchée par un évènement

On peut reprendre la définition issue de [Cassandras and Lafortune, 2009], article qui présente les systèmes à évènements discrets et toutes leurs déclinaisons.

Définition 5 (Système à évènements discrets [Cassandras and Lafortune, 2009]). *Un SED est un système à espace discret dont l'évolution de l'état dépend entièrement de l'occurrence d'évènements asynchrones au cours du temps.*

Lorsque les systèmes sont abstraits sous forme de SED, le formalisme utilisé est généralement celui des machines à états finis.

Définition 6 (Machine à états finis). *Une machine à états finis est un tuple* $M = (S, \Sigma, \delta, s_0)$ *dans lequel*
— S *est un ensemble fini d'états,*
— Σ *est un ensemble fini d'évènements,*
— $\delta : S \times \Sigma \to 2^S$ *est une fonction de transition,*
— s_0 *est l'état initial.*

Il est aussi possible d'abstraire un SED sous forme de structure de Kripke, un modèle proche d'un automate fini.

Définition 7 (Structure de Kripke [Kripke, 1965]). *Une structure de Kripke est un tuple* $M = (S, I, R, L)$ *dans lequel*
— S *est un ensemble fini d'états,*
— $I \subseteq S$ *est un ensemble d'états initiaux,*
— $R \subseteq S \times S$ *est une relation de transition telle que pour tout* $s \in S$*, il existe* $s' \in S$ *tel que* $(s, s') \in R$*,*
— $L : S \to 2^{AP}$ *est une fonction d'étiquetage,* AP *étant un ensemble d'expressions booléennes portant sur des variables d'un ensemble* `S`*.*

Les structures de Kripke sont notamment utilisées dans le model-checking, technique informatique permettent de représenter le comportement d'un système et de vérifier certaines propriétés. Cette modélisation repose généralement sur l'utilisation de logiques temporelles, qui permettent à la fois de décrire le comportement du système dans le temps mais aussi de décrire l'existence de certains chemins. Les chemins sont des séquences d'états dans les machines à états, l'ensemble des chemins possibles forme le langage de la machine. On définit la notion de langage plus loin dans ce chapitre.

La principale différence entre les machines à états finis et les structures de Kripke réside dans la nature de la cadence des évènements. Pour le cas des machines à états finis, on considère que les évènements peuvent apparaître de manière asynchrone, on parle alors de cadence variable. A l'inverse, dans les structure de Kripke on peut considérer un évènement directement comme le changement d'état d'une variable booléene. Dans ce cas, la cadence des évènements est fixe (généralement correspondant à un tic d'horloge) et à partir des évaluations booléennes définissant un état, on peut déterminer si celui-ci a changé.

Il existe d'autres formalismes qui permettent de représenter un système à évènements discrets. On peut notamment citer les réseaux de Petri [Petri, 1966], et les machines de Mealy [Mealy, 1955].

2.2.1 SED partiellement observable

Afin de modéliser des systèmes dont seule une partie de la dynamique est observable, on peut utiliser les machines à états finis partiellement observables. On peut distinguer deux types d'observabilité partielle, la première portant sur les évènements, la seconde portant

sur les variables d'état, chacune s'apparentant respectivement au formalisme des machines à état finis et aux structures de Kripke.

Dans les approches classiques de la littérature utilisant le modèle des machines à états finis [Jéron et al., 2006, Sampath et al., 1995], il est considéré que certains évènements sont non-observables. L'ensemble des évènements Σ est alors partitionné en deux ensembles disjoints Σ_o et Σ_u représentant respectivement les ensembles d'évènements observables et non-observables.

L'observabilité partielle dans une structure de Kripke se retrouve au niveau des variables. Si on considère que l'ensemble d'état S est un ensemble d'affectation sur un ensemble de variables `S`, ce dernier est alors partitionné en deux ensembles disjoints `O` et `U` représentant respectivement les variables observables et celles qui ne le sont pas. L'observation n'est donc pas considérée comme un évènement, mais comme une affectation des variables de `O`. On note l'ensemble des observations O, c'est-à-dire l'ensemble d'affectation sur l'ensemble `O`. Si on souhaite représenter une structure de Kripke partiellement observable, on peut utiliser les automates de Moore [Moore, 1956] sans alphabet d'entrée, comme démontré dans [Schneider, 2003] et illustré dans la figure 2.1. Puisque l'observation est une affectation sur un sous-ensemble fixe de variables d'états, plutôt que de labéliser l'arc avec l'observation, on peut utiliser les observations dans l'alphabet de sortie.

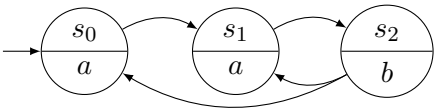


FIGURE 2.1 – Une structure de Kripke partiellement observable avec espace d'état $S = \{s_0, s_1, s_2\}$, et ensemble d'observations $O = \{a, b\}$ représentée comme un automate de Moore sans alphabet d'entrée.

Dans la figure 2.1, la partie haute d'un noeud représente un état complet du système et sa partie basse est la projection de cet état sur les observations. On considère que l'alphabet de sortie de la machine de Moore représente les observations pouvant être générées par le système. On aurait aussi pu représenter les observations avec l'alphabet d'entrée de la Machine de Moore (label des observations de l'état suivant sur l'arc). Cependant le choix d'utiliser l'alphabet de sortie permet de conserver l'aspect affectation sur variables observables de l'état.

2.2.2 Langages associés aux SED

La notion de langage intervient dans les SED afin de décrire l'ensemble des séquences d'états cohérentes dans un tel système et pouvoir raisonner sur les différents chemins d'états possibles.

Pour définir l'ensemble des états atteignables et des chemins dans un SED nous reprenons les concepts de la théorie des langages des automates. Une séquence d'états cohérente dans le système peut être considérée comme un mot accepté par l'automate du système. Pour désigner l'ensemble de ces mots, on parle du langage accepté de l'automate. Nous introduisons d'abord le langage d'exécution d'un système à évènements discrets comme l'ensemble des séquences d'états pouvant être produites par ce système.

Définition 8 (Langage d'exécution). *Pour un système à évènements discrets (S, Δ, s_0) où S est l'espace d'états, Δ est la relation de transition du système et s_0 est son état initial, le* langage d'exécution $\mathcal{L}(\Delta) \subseteq S^*$ *est l'ensemble des séquences commençant par s_0*

satisfaisant Δ *:*

$$\mathcal{L}(\Delta) = \{(s_0, s_1, \cdots, s_n)| \\ n \in \mathbb{N}^+, i \in [0, n-1], (s_i, s_{i+1}) \in \Delta\}$$

Notations : Soit $seq \in \mathcal{L}(\Delta)$ une séquence d'états, $seq[i]$ dénote le $i^{\text{ème}}$ état de la séquence commençant à l'état initial $seq[0]$, la taille de la séquence est notée $|seq|$. On note O l'ensemble des observations possibles et on note *obs* la fonction d'observation de S vers O qui projette un état sur son observation et nous l'étendons naturellement de S^* vers O^* comme suit : $|obs(seq)| = |seq|$ et $obs(seq)[i] = obs(seq[i])$ pour $i \in [0, |seq|[$.

Maintenant que l'ensemble des séquences d'états a été défini par le langage d'exécution du système, on s'intéresse à l'ensemble de toutes les séquences d'observations pouvant être produites par le système. On projette le langage d'exécution sur les observations afin d'obtenir le langage observable du système.

Définition 9 (Langage observable). *Le langage observable* $\mathcal{L}_{obs}(\Delta) \subseteq O^*$ *d'un SED* (S, Δ, s_0) *est l'ensemble des séquences d'observation cohérentes avec* Δ*.*

$$\mathcal{L}_{obs}(\Delta) = \{obs(seq)|seq \in \mathcal{L}(\Delta)\}$$

Les langages sont utiles lorsqu'on souhaite raisonner sur les chemins dans les machines à états. La notion de langage est utilisée dans le chapitre 6 afin de définir une condition nécessaire et suffisante pour la propriété d'*estimabilité à état unique*.

2.3 Diagnostic des systèmes à évènements discrets

Le problème du diagnostic des SED consiste à déduire, à partir des observations fournies par le système, si certains évènements sont ou non apparus au cours du temps. Généralement, certains de ces évènements ont un impact critique sur le fonctionnement du système : on parle alors de faute. Le problème consiste précisément à déduire l'occurrence ou non de fautes lors de l'exécution du système. Une faute peut être par exemple un court-circuit, la panne d'un composant du système ou encore une difficulté liée à l'environnement dans lequel évolue le système.

On distingue deux types de faute pouvant survenir dans un SED : les fautes permanentes et intermittentes. Le principal critère qui les différencie est la manière dont la faute persiste dans le système. Un faute permanente, dès lors qu'elle apparaît, persiste au cours du temps. Il peut s'agir par exemple de la perte d'un composant ou d'une panne irréparable. A l'inverse, une faute intermittente peut disparaître après un certain délai, par exemple un court-circuit ou encore un problème de communication du à une baisse de signal. De nombreuses approches ont été développées, on peut notamment citer l'algorithme de *Livingstone* [Williams and Nayak, 1996] qui est très en lien avec la théorie de [Reiter, 1987], où encore les algorithmes de [Forney, 1973] ainsi que l'algorithme *FastDiag* [Felfernig and Schubert, 2010]. Le chapitre [Grastien and Zanella, 2019] propose une étude de différentes approches pour le diagnostic des fautes dans les SED.

2.3.1 Diagnostic de fautes dans les SED

Dès lors qu'on étudie les SED partiellement observables, on peut s'intéresser au problème du diagnostic. Le diagnostic consiste à déterminer quelles fautes (des évènements non-observables) expliquent une séquence d'observations en se basant sur le modèle du système. Le diagnostic en ligne et l'analyse de diagnosticabilité pour les SED ont d'abord été introduits par [Sampath et al., 1995]. Le diagnostic de faute est lié au problème d'observabilité

d'un SED qui consiste à construire un automate déterministe dont les transitions sont régies uniquement par les évènements observables. On appelle cet automate un « observateur » [Zaytoon and Lafortune, 2013] et chaque état de cet automate correspond à une estimation de l'état réel du système observé. Dans l'approche de [Sampath et al., 1995], on suppose que le comportement du SED étudié est connu et que son modèle est décrit comme une machine à états G dans laquelle peuvent survenir différents évènements d'un ensemble Σ. Cet ensemble est subdivisé en deux ensembles disjoints Σ_o et Σ_u représentant respectivement les évènements observables et non-observables.

Définition 10 (Modèle d'un SED partiellement observable [Sampath et al., 1995]). *L'approche de [Sampath et al., 1995] consiste à représenter les systèmes à analyser sous forme d'automates finis où :*

1. *S est l'ensemble d'états finis incluant des états sains et d'autres où une faute est présente,*
2. *$\Sigma = \Sigma_o \cup \Sigma_u$ est l'ensemble des événements pouvant survenir.*
3. *δ est une relation de transition, $\delta \subseteq S \times \Sigma \times S$*
4. *s_o est l'état initial.*

On considère qu'au sein de l'ensemble Σ_u il existe un ensemble d'évènements de faute Σ_f tel que $\Sigma_f \subseteq \Sigma_u$. Le but du diagnostic des SED est donc de détecter les séquences fautives en se basant uniquement sur les évènements dans Σ_o. Une séquence fautive est une séquence d'évènements contenant au moins une fois l'occurrence d'un évènement de Σ_f. Le problème du diagnostic peut se traduire par la construction d'un diagnostiqueur qu'on peut représenter sous la forme d'une fonction $Diag : \mathcal{L}_{obs(G))} \rightarrow \{yes, no, ?\}$ qui indique si toutes les séquences d'évènements cohérentes avec une séquence d'observations en entrée contiennent au moins un évènement de Σ_f. Formellement, le problème du diagnostic est défini comme suit :

Définition 11 (Le problème du diagnostic ([Jéron et al., 2006])). *Considérons une machine à états G et Σ_f l'ensemble des évènements de fautes. Pour une séquence d'observations $s \in \mathcal{L}_{obs(G))}$, la fonction $Diag : \mathcal{L}_{obs(G))} \rightarrow \{yes, no, ?\}$ vérifie :*

$$Diag(s) = \left\{ \begin{array}{l} yes \text{ si } \forall t \in obs^{-1}(s) : \Sigma_f \in t \\ no \text{ si } \forall t \in obs^{-1}(s) : \Sigma_f \notin t \\ ? \text{ dans les autres cas.} \end{array} \right\}$$

Afin de répondre au problème du diagnostic, l'approche de [Sampath et al., 1995] consiste à construire un modèle intermédiaire appelé *diagnostiqueur*. Cette machine à états est le produit synchrone de la machine à états du système déterminisée sur les observations. Le diagnostiqueur est défini comme suit :

Définition 12 (Diagnostiqueur [Sampath et al., 1995]). *Le diagnostiqueur du modèle d'un système G est une machine à états déterministe $D_G = (\mathcal{Q}, \Sigma_d, \sigma_d, q_0)$ associée à une fonction de labélisation $S \times 2^L$ pour chaque état, avec $L = \{N, F\}$ (N pour Normal et F pour Fautif) dans laquelle :*

- *$\mathcal{Q} = 2^{S \times L}$ est l'ensemble des états,*
- *$\Sigma_d = \Sigma_o$ est l'ensemble des évènements,*
- *$\sigma_d : \mathcal{Q} \times \Sigma_d \rightarrow \mathcal{Q}$ est la relation de transition,*
- *$q_0 = (s_0, N)$ est l'état initial,*

Chaque état q du diagnostiqueur est de la forme $q = \{(s_1, l_1), .., (s_n, l_n)\}$ avec $s_i \in S$ et $l_i \in L$.

- Si $\forall i \in 1..n$ on a $l_i = N$, on dira que l'état du diagnostiqueur est *N-certain* puisqu'il ne contient que des états sans faute.

— Si $\forall i \in 1..n$ on a $l_i = F$, on dira que l'état du diagnostiqueur est *F-certain* puisqu'il ne contient que des états avec occurrence de faute.
— Dans les autres cas, on dira que q est *F-incertain* car il contient à la fois des états sans faute et des états fautifs.

Les auteurs émettent l'hypothèse que toutes les fautes sont de nature permanente. La propagation d'une faute suit donc un schéma précis dans le diagnostiqueur comme illustré dans la figure 2.2.

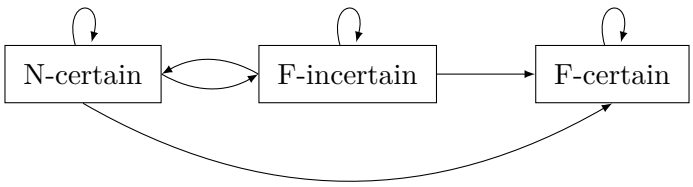


FIGURE 2.2 – Propagation des fautes dans le diagnostiqueur de [Sampath et al., 1995].

Une fois le diagnostiqueur construit, on peut répondre au problème du diagnostic dès lors que le diagnostiqueur se trouve dans un état *N-certain* ou *F-certain*. Cependant dans les cas où le diagnostiqueur est dans un état *F-incertain*, il est nécessaire d'attendre que celui-ci évolue dans un état *F-certain* ou *N-certain*. Il existe cependant des systèmes dont le diagnostiqueur peut parcourir un nombre illimité d'états *F-incertain*. Pour répondre à cette problématique les auteurs de [Sampath et al., 1995] proposent la définition d'une propriété portant sur les SED : la diagnosticabilité.

2.3.2 La diagnosticabilité

La propriété de diagnosticabilité a été introduite par [Sampath et al., 1995] sous l'hypothèse que toutes les fautes survenant lors de l'exécution du système sont permanentes. Cette propriété indique qu'à partir de l'occurrence d'une faute il existe une borne sur les observations accumulées permettent de déduire celle-ci avec certitude. Une faute est dite diagnosticable si celle-ci peut être détectée en un nombre fini d'observations après son occurrence. Un système est diagnosticable si toutes ses fautes sont diagnosticables.

Sur la base du diagnostiqueur étudié dans la section précédente, les auteurs ont développé une condition nécessaire et suffisante pour la diagnosticabilité. Celle-ci se base sur l'existence de cycle d'états labélisés par *F-incertain* dans le diagnostiqueur. Un cycle *F-incertain* dans le diagnostiqueur est un cycle composé uniquement d'états *F-incertain*. Un cycle *F-indéterminé* dans le diagnostiqueur est défini comme un cycle *F-incertain* pour lequel il existe deux cycles dans la machine à états du système G partageant les mêmes observations tel qu'un des deux cycles ne contient que des états fautifs et l'autre que des états nominaux. La notion de cycle *F-indéterminé* permet de définir une condition nécessaire et suffisante pour la diagnosticabilité dans les travaux de [Sampath et al., 1995].

Théorème 1 (Condition nécessaire et suffisante pour la diagnosticabilité). *Une machine à états finis G est diagnosticable si et seulement si il n'existe aucun cycle F-indéterminé dans son diagnostiqueur D_G pour chaque faute de Σ_f.*

Dans un article qui dresse un état de l'art des méthodes utilisés pour le diagnostic des systèmes à évènements discrets, [Zaytoon and Lafortune, 2013] proposent une définition plus générale de la diagnosticabilité.

Définition 13 (Système diagnosticable[Zaytoon and Lafortune, 2013]). *Une ma-*

chine à états finis G est diagnosticable s'il est possible de détecter dans un délai[1] *fini l'occurrence de fautes en utilisant uniquement l'historique des observations. La diagnosticabilité requiert que chaque occurrence de faute produise des observations suffisamment distinctes pour permettre son identification.*

2.3.3 Vérification de la diagnosticabilité

Les auteurs de [Cimatti et al., 2003] proposent une méthode de vérification formelle de la diagnosticabilié, appelée la méthode du *Twin-plant*. Cette méthode, basée sur le model-checking symbolique, permet de vérifier la propriété de diagnosticabilité en construisant le produit synchrone de la machine à états du système.

La méthode analyse la diagnosticabilité d'un système en vérifiant que celui-ci n'admet pas deux exécutions possibles pour une même séquence d'événements. Le système observé est représenté comme un système d'état-transition partiellement observable (plant) P = < X,U,Y, δ > où X,U,Y sont respectivement l'espace d'état, l'espace d'entrée et l'espace de sortie et $\delta \subseteq X \times U \times Y \times X$ est la relation de transition.

Étant données deux conditions $c1$ et $c2$ sur un état de P, on considère la condition de diagnostic $c1 \perp c2$ qui exprime le fait que le diagnostic peut décider entre $c1$ et $c2$. Ces conditions permettent généralement d'exprimer la présence ou non d'une faute : ($faute \perp \neg faute$). Une paire d'exécutions possibles pour la condition $c1 \perp c2$ est appelée paire-critique. Pour vérifier la condition $c1 \perp c2$, on construit le Twin Plant P^2 de P à partir de deux copies de P synchronisées sur les entrées et les sorties. Le twin plant de P = <X,U,Y,δ > est P^2 = <X^2,U,Y,δ' > où $X^2 = X \times X$ et $\delta' = \{((x_{01}, x_{02}),u,y,(x_1,x_2)) \mid ((x_{01},u,y,x_{02}),(x_1,u,y,x_2)) \in \delta\}$. Une exécution de P^2 correspond à une paire d'exécutions possibles de P. Trouver une paire critique devient alors un problème d'atteignabilité dans P^2, qui peut être résolu grâce au model-checking. Les travaux présentés dans le chapitre 4 s'inspirent de la technique du Twin-Plant.

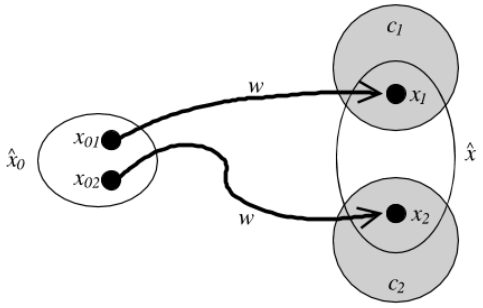


FIGURE 2.3 – Le principe du twin-plant, figure extraite de [Cimatti et al., 2003].

Une extension de cette propriété, la K-Diagnosticabilité [Zaytoon and Lafortune, 2013] impose le délai K entre l'occurrence d'une faute et sa détection. La diagnosticabilité a ensuite été déclinée pour d'autres types de systèmes, pour les SED stochastiques par exemple [Thorsley and Teneketzis, 2005], ou encore pour des modèles hybrides
[Bayoudh and Travé-Massuyès, 2014].

2.3.4 Autres propriétés

Il existe d'autres propriétés relatives au diagnostic des SED dont l'observabilité
[Ozveren and Willsky, 1990] ; en supposant que seule une partie des évènements peut être observée dans un SED, la propriété d'observabilité indique qu'a partir d'une séquence

1. Un délai n'est pas forcément un délai temporel mais exprime plutôt une borne sur le nombre d'observations.

d'évènements observables passée et connue il est possible de déterminer exactement l'état courant du système de manière intermittente mais dans un délai borné.

La détectabilité [Shu et al., 2007] est liée à l'observabilité, cette propriété indique qu'un système est « détectable » si il existe un observateur pouvant être utilisé comme diagnostiqueur.

La prédictabilité [Jéron et al., 2008] s'intéresse à la prédiction d'occurrences de motifs de pannes dans les SED. Un motif de panne est définie comme un ensemble de séquences et celui-ci est « prédictible » s'il est possible d'inférer l'occurrence de ce motif avant son exécution réel dans le système.

La manisfestabilité [Ye et al., 2016] propose une forme plus faible de diagnosticabilité, en s'assurant qu'un système admet au moins une trajectoire d'états futurs après l'occurrence d'une faute permettant de déduire avec certitude l'occurrence de celle-ci.

Pour un aperçu plus large de la littérature, on peut se référer à l'article de [Zaytoon and Lafortune, 2013] qui donne une vue d'ensemble des différentes méthodes et algorithmes pour le diagnostic développés au cours des dernières décennies. Tout comme la vérification de la diagnosticabilité, la vérification de ces propriétés peut être réalisée grâce au model-checking. C'est pourquoi nous lui consacrons la section suivante en commençant par une brève présentation des logiques temporelles sur lesquelles le model-checking repose.

2.4 Le model-checking

Le model-checking [Clarke et al., 1999] désigne un ensemble d'outils et de méthodes informatiques permettant de vérifier des propriétés sur des systèmes à évènements discrets. Cette technique permet notamment d'étudier l'existence de chemin dans des machines à états décrites sous forme de structure de Kripke. On peut grâce au model-checking vérifier si un système peut se retrouver dans certains états (analyse d'atteignabilité) mais on peut aussi vérifier d'autres propriétés plus complexes. Afin de définir des systèmes dynamiques mais aussi de décrire leurs propriétés, le model-checking repose sur l'utilisation de logiques temporelles.

2.4.1 Les logiques temporelles

Les logiques temporelles sont utilisées afin de décrire le fonctionnement dynamique des systèmes mais sont aussi utiles pour analyser des propriétés. On aborde ici différentes logiques temporelles utilisées en model-checking. Les différents opérateurs et leur sémantique sont décrits.

Définition 14 (LTL : Linear Temporal Logic [Pnueli, 1977]). LTL *est une logique permettant d'exprimer la notion de temps en raisonnant sur les instants suivants. Celle-ci permet d'écrire des formules sur l'avenir d'un chemin infini dans une machine à états finis mais aussi de décrire sa relation de transition. Pour des formules de logique propositionnelle* ϕ *et* ϕ_2*, on retrouve dans* LTL *les opérateurs suivants :*

- $\boldsymbol{X}(\phi)$*(*next*) est vraie si* ϕ *est vraie à l'instant suivant.*
- $\boldsymbol{F}(\phi)$*(*finally*) est vraie si* ϕ *est vraie dans au moins un des instants futurs.*
- $\boldsymbol{G}(\phi)$*(*globally*) est vraie si* ϕ *est vraie dans tous les instants futurs.*
- (ϕ) $\boldsymbol{U}$ (ϕ_2)*(*until*) est vraie si* ϕ *est vraie dans tous les instants jusqu'à ce que* ϕ_2 *soit vraie.*

Sémantique des opérateurs LTL :

Soit c un chemin infini dans une machine à états, c'est-à-dire une séquence d'états infinie $(s_0,s_1,s_2,..)$*.*

- $c \vDash \phi$ *si et seulement si* $s_o \vDash \phi$ *: les propriétés sont évaluées à l'état initial.*
- $c \vDash \boldsymbol{X}(\phi)$ *si et seulement si* $(s_1, s_2,..) \vDash \phi$*.*
- $c \vDash \boldsymbol{F}(\phi)$ *si et seulement si* $\exists\ i,\ s_i \vDash \phi$*.*

— $c \vDash \boldsymbol{G}(\phi)$ *si et seulement si* $\forall\ i,\ s_i \vDash \phi$.
— $c \vDash \phi_1 \boldsymbol{U} \phi_2$ *si et seulement si* $\exists\ i,\ s_i \vDash \phi_2 \wedge \forall\ j < i,\ c[j] \vDash \phi_1$.

On utilisera notamment l'opérateur **X** de la logique LTL dans certains encodages proposés dans les chapitres 4 et 5.

Définition 15 (CTL : Computational Tree Logic [Clarke and Emerson, 1982]). CTL *est une logique arborescente qui permet de raisonner sur l'ensemble des chemins d'un système à évènements discrets. Pour une formule* ϕ*, on retrouve les opérateurs* $\boldsymbol{A}$ *et* $\boldsymbol{E}$ *inspirés des opérateurs* $\forall$ *et* $\exists$ *de la logique du premier ordre.*
— $\boldsymbol{A}(\phi)$ *tous les chemins satisfont* ϕ
— $\boldsymbol{E}(\phi)$ *il existe au moins un chemin satisfaisant* ϕ

Sémantique des opérateurs CTL :
Soit $\square$ *qui représente* $\boldsymbol{F}(\phi)$, $\boldsymbol{G}(\phi)$, $\boldsymbol{X}(\phi)$ *ou* $\phi_1 \boldsymbol{U} \phi_2$ *et* C *un ensemble fini de chemins infinis* $\{c_1,c_2,..,c_n\}$
— $s \vDash \boldsymbol{A}\square$ *si et seulement si* $\forall\ c \in C,\ c \vDash \square$.
— $s \vDash \boldsymbol{E}\square$ *si et seulement si* $\exists\ c \in C,\ c \vDash \square$.

Les logiques *LTL* et *CTL* permettent de raisonner sur des chemins dans des machines à états. Le model-checking vérifie la satisfaction d'une propriété, pouvant s'écrire avec ces logiques, par un modèle. Si par exemple on souhaite vérifier sur un système qu'il existe un chemin qui satisfait à tout instant une formule ϕ , on cherchera à satisfaire la formule $\mathbf{EG}(\phi)$.

Définition 16 (PTLTL : Past Time Linear Temporal Logic [Emerson, 1990]). *Pour des formules* ϕ *et* ϕ_2*, on retrouve en PTLTL les opérateurs Y, O, S, Z :*
— $\boldsymbol{Y}(\phi)$ *(*yesterday*) est vraie si* ϕ *est vrai à l'instant précédent.* $\boldsymbol{Y}(\phi)$ *est toujours vrai à l'instant* 0.
— $\boldsymbol{O}(\phi)$ *(*once*) est vraie si* ϕ *est vrai dans au moins un des instants passés.*
— $\boldsymbol{H}(\phi)$ *(*historically*) est vraie si* ϕ *est vrai dans tous les instants passés.*
— $(\phi)\boldsymbol{S}(\phi_2)$ *(*since*) est vraie si* ϕ *est vraie depuis l'instant ou* ϕ_2 *est vraie.*
— $\boldsymbol{Z}(\phi)$ *fonctionne de la même manière que* $\boldsymbol{Y}$*, à la différence que* $\boldsymbol{Z}(\phi)$ *est faux à l'instant* 0.

Sémantique des opérateurs PTLTL :
Soit c un chemin fini dans une machine à états, c'est-à-dire une séquence d'états finie $(s_0,s_1,...,s_k)$.
— $c \vDash \phi$ *si et seulement si* $s_k \vDash \phi$ *: les propriétés sont évaluées à partir de l'état courant.*
— $c \vDash \boldsymbol{Y}(\phi)$ *si et seulement si* $k = 0$ *ou* $s_{k-1} \vDash \phi$
— $c \vDash \boldsymbol{O}(\phi)$ *si et seulement si* $\exists\ i \in [0,k-1],\ s_i \vDash \phi$
— $c \vDash \boldsymbol{H}(\phi)$ *si et seulement si* $\forall\ i \in [0,k-1],\ s_i \vDash \phi$
— $c \vDash (\phi_1)\boldsymbol{S}(\phi_2)$ *si et seulement si* $k = 0$ *alors* $s_0 \vDash \phi_2$*, sinon* $\exists\ i \in [0,k],\ s_i \vDash \phi_2$ *et* $\forall\ j \in [i+1,k],\ s_j\ \phi_1$.
— $c \vDash \boldsymbol{Z}(\phi)$ *si et seulement si* $k \neq 0$ *et* $s_{k-1} \vDash \phi$

On remarque une certaine symétrie entre les opérateurs de *LTL* et *PTLTL*, les premiers permettant de raisonner sur des instants futurs, les seconds sur des instants passés. L'utilisation de *PTLTL* par rapport à *LTL* dans la définition de relations de transition est motivée par le raisonnement sur un horizon borné (des séquences d'états finis). Les opérateurs de *PTLTL* seront utilisés dans les chapitres 3, 4 et 5 afin de décrire des modèles de diagnostic.

Définition 17 (TLA+ : Temporal Logic of Actions [Merz, 2003]). *On retrouve en TLA+ plusieurs opérateurs pour établir des spécifications ou définir des propriétés à vérifier :*

— ***always*** *: équivalent à* $\boldsymbol{AG}$ *en CTL*

— ***until*** *: équivalent à* ***U*** *en LTL*
— ***after*** *: équivalent à* ***X*** *en LTL*
— ***eventually*** *: équivalent à* ***EF*** *en CTL*

TLA+ n'est pas une logique mais un langage de spécification haut niveau ciblant des systèmes réactifs, distribués et en particulier des systèmes asynchrones. TLA+ combine les aspects de logique temporelle déjà présents dans TLA et la théorie des ensembles mathématiques. Ces opérateurs permettent notamment de quantifier sur les chemins. On retrouvera les opérateurs de la logiques *TLA+* dans certains encodages de problème proposés dans les chapitres 4 et 5.

2.4.2 Les classes de complexité

En informatique théorique, les classes de complexité permettent de classifier les problèmes en fonction de leur difficulté calculatoire. Les deux classes de complexité les plus connues sont P et NP :

— P regroupe tous les problème pour lesquels la résolution peut être effectuée en temps polynomial par rapport à la taille de l'entrée,
— NP englobe la classe P ($P \subseteq NP$) et regroupe tous les problèmes dont la vérification d'une solution candidate peut-être effectué en temps polynomial.

Le problème P=NP constitue une conjecture mathématique, c'est-à-dire qu'il n'existe à ce jour aucune preuve montrant que P=NP ou P≠NP. Pour un problème de NP, on peut aussi parler de sa NP-complétude. Les problèmes dits NP-Complet peuvent être vus comme les problèmes les plus difficiles de NP. De plus, on peut ramener tout problème de NP à un problème NP-Complet par l'intermédiaire de réductions polynomiales. Il existe d'autre classes de complexité, celle juste au dessus NP se nomme P-SPACE ($NP \subseteq$ P-SPACE). Cela signifie qu'une mémoire de taille polynomiale en la taille de l'entrée est nécessaire pour la vérification d'un problème de cette classe. Par exemple les problèmes de vérification d'une propriété écrite en *LTL*,*CTL* ou *PTLTL* appartiennent à la classe de complexité P-SPACE.

2.4.3 Les solveurs SAT

Les solvers SAT sont des outils permettant de répondre au problème de la satisfiabilité d'une formule aussi appelé problème **SAT** introduit par [Cook, 1971]. A partir d'un ensemble de contraintes en logique propositionnelle et d'un ensemble de variables, le problème SAT consiste à trouver, si elle existe, une affectation booléenne sur toutes les variables cohérente avec l'ensemble de contraintes. Un solveur SAT peut renvoyer deux types de réponse en fonction du problème en entrée : UNSAT si la formule n'est pas satisfiable ou SAT si la formule est satisfiable. Dans le deuxième cas, il est possible de retrouver le modèle du problème, c'est-à-dire une affectation possible pour chacune des variables.

Les solveurs SAT sont notamment utilisés en diagnostic à base de cohérence et dans les outils de model-checking qui transforment les structures de Kripkes encodées dans leur syntaxe en des problèmes SAT. Il existe plusieurs extensions aux solveurs SAT, notamment les solveurs SMT (Sat Modulo Therory) qui permettent de combiner un solveur SAT avec d'autres théories telles que la théorie des réels ou celle des graphes par exemple.

Voici une liste non-exhaustive de quelques outils et solveurs existants dont certains ont été utilisés pour mener les travaux de cette thèse.

Outils de model-checking

— *NuSMv* [Cimatti et al., 2000] : un model-checker, développé à l'institut FBK en Italie permettant de vérifier des propriétés écrites en *LTL*.
— *Alloy* [Jackson, 2006] : un outil d'analyse de logiciel, développé au MIT, très efficace pour la vérification de propriétés statiques.

— *Electrum* [Macedo et al., 2016] : une surcouche d'Alloy intégrant des opérateurs de logique temporelle de TLA+, développé à l'ONERA en France et à l'université de Minho au Portugal. On l'utilise dans les chapitres 4 et 5 de ce document.

Solveurs

— *zChaff* [Moskewicz et al., 2001] : un solver SAT développé à l'université de *Princeton* (New Jersey), utilisé par NuXMV et pouvant être utilisé avec Electrum.
— *Sat4j* [Le Berre and Parrain, 2010] : un solveur SAT pour Java, utilisé dans les chapitres 4 et 5.
— *Monosat* [Bayless et al., 2015] : un solveur SMT qui utilise la théorie des graphes ; on l'utilise dans le chapitre 6.

Les différents outils mathématiques et informatiques utilisés tout au long de ce document ont été abordés dans ce chapitre. Certaines définitions et concepts sont utiles à la compréhension des chapitres suivants. Le chapitre suivant décrit le formalisme de représentation des systèmes à évènements discrets utilisé dans ces travaux ainsi que la manière d'effectuer l'estimation d'un état unique à chaque étape discrète.

Chapitre 3

Formalisme pour l'estimation à état unique

Ce chapitre présente le formalisme utilisé pour concevoir des modèles de diagnostic pour des systèmes à évènements discrets. Celui-ci est inspiré des travaux de [Pralet et al., 2016]. On décrit dans ce chapitre la construction de ce qu'on appellera un estimateur, qui produit un diagnostic non seulement incrémental mais aussi unique. Le diagnostic émis est incrémental, car l'estimation de l'état courant repose sur l'estimation de l'état précédent. On parle de diagnostic unique puisqu'à chaque étape discrète un seul état est estimé à partir de l'observation du système et de l'état précédemment estimé. Contrairement aux approches classiques de la littérature, cela permet de pallier le problème de passage à l'échelle puisque tous les états possibles (les candidats) ne sont pas gardés en mémoire. De plus, cela facilite la prise de décision, qu'elle soit humaine ou automatique, quant à la suite de la mission autonome.

La dynamique du système surveillé est modélisée par des contraintes en logique propositionnelle. La méthode permettant le choix d'un diagnostic unique est formalisée à l'aide d'une théorie de préférences conditionnelles. Le problème de l'estimation est ensuite résolu grâce à un solveur Max-SAT. Nous décrivons le modèle de diagnostic permettant de définir des estimateurs incrémentaux à état unique, puis nous étudions les limites d'un tel formalisme.

3.1 Cadre formel

On considère que le système évolue dans le temps de manière discrète et que chaque étape discrète est de même durée. Le modèle de diagnostic étudié se présente sous la forme d'un tuple (s_0, Δ, Γ) où s_0 est l'état initial du système, Δ est le modèle comportemental du système (un ensemble de règles qui régissent les transitions du système) et Γ est le modèle de préférences conditionnelles permettant de sélectionner un état unique parmi les candidats au diagnostic. Le couple (s_0,Δ) décrit donc un système à évènements discrets sous forme de structure de Kripke (voir chapitre 2). Cette section présente en détail le formalisme et les notations adoptées afin de représenter des modèles d'estimation de la forme (s_0, Δ, Γ).

Notations

- Pour un ensemble de variables booléennes $\mathtt{X}$, on définit une *affectation* comme une fonction de $\mathtt{X}$ vers $\{\top, \bot\}$ qui associe à chaque variable $\mathtt{x}$ de $\mathtt{X}$ une valeur booléenne *vrai* ($\top$) ou *faux* ($\bot$).
- On note x et $\overline{x}$ l'affectation de $\{\mathtt{x}\}$ telle que $x(\mathtt{x}) = \top$ et $\overline{x}(\mathtt{x}) = \bot$.
- Pour un ensemble de variables $\mathtt{X} = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$, si pour tout $i \in [1, n]$, f_i est une affectation $\{\mathtt{x_i}\} \to \{\top, \bot\}$, alors $f_1\ f_2 \ldots f_n$ désignent l'affectation f sur $\mathtt{X}$ telle que

$\forall \mathtt{x_i} \in \mathtt{X}, f(\mathtt{x_i}) = f_i(\mathtt{x_i})$. Par exemple, $a\ \overline{b}\ \overline{c}$ est l'affectation de $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ qui affecte vrai à a et faux à b et c.

3.1.1 Variables d'état

On utilise un ensemble de variables P pour décrire l'état du système à chaque instant. Cet ensemble est lui-même partitionné en deux ensembles disjoints O and E qui représentent respectivement les éléments que l'on observe et ceux que l'on doit estimer dans le système. On considère que ces ensembles sont invariants lors de l'exécution. À chaque étape discrète, étant donnée une affectation sur les variables de O et un état précédent estimé, le problème de l'estimation consiste à estimer la valeur des variables de E. On introduit les états et les observations comme des affectations respectivement sur les variables d'état et les variables observables du système et les notations associées.

Définition 18 (État et observation). *Un état, noté s, est une affectation sur les variables de* P*. On note $\mathcal{S}$ l'ensemble des états. Une observation, notée o, est une affectation sur les variables de* O*. On note $\mathcal{O}$ l'ensemble des observations.*

3.1.2 Le modèle comportemental du système

Le modèle comportemental $\Delta \subseteq \mathcal{S}^2$ décrit la relation de transition du système. On ne souhaite pas décrire de manière explicite cette relation de transition pour l'instant. On utilise des contraintes en logique propositionnelle qui représentent de manière compacte les transitions entre états précédents et états suivants possibles dans le système. On introduit également une fonction bijective *pre* sur l'ensemble P. Pour une variable p de P, $pre(\mathtt{p})$ représente la valeur de la variable p à l'instant précédent. On définit l'ensemble de variables $\mathtt{P_{pre}} = \{\mathtt{pre\_p} \mid \mathtt{p} \in \mathtt{P}\}$ tel que $\forall \mathtt{p} \in \mathtt{P}, pre(\mathtt{p}) = \mathtt{pre\_p}$. Δ est représenté sous la forme d'un ensemble de règles de logique propositionnelle Δ_p portant sur les variables de P et de $\mathtt{P_{pre}}$.

Exemple 2 (Modèle comportemental d'un robot autonome). *On considère un robot autonome décrit par un modèle comportemental Δ et dont l'état initial est s_0.*

$$\Delta = \left\{ \begin{array}{c} \mathtt{move} \leftrightarrow \mathtt{engine} \wedge \neg \mathtt{f_{engine}} \wedge \neg \mathtt{f_{move}} \\ \mathtt{pre\_f_{engine}} \rightarrow \mathtt{f_{engine}} \end{array} \right\}$$

$\mathtt{s_0}$ = *move engine* $\overline{f_{engine}}$ $\overline{f_{move}}$

Δ décrit le comportement suivant : le robot est en mouvement (move) si et seulement si le moteur est sous tension (engine), qu'il n'y a pas de faute au niveau du moteur ($\overline{f_{move}}$) ni au niveau des roues (un glissement : $\overline{f_{engine}}$). La faute $\mathtt{f_{engine}}$ est une faute permanente puisque si celle-ci était présente à l'état précédent, alors elle l'est aussi à l'état courant. On ne peut observer que le mouvement du robot (move) et si le moteur est sous tension (engine).

Définition 19 (Affectation entre deux états successeurs). *Pour toute paire d'états $(s_{pre}, s_{now}) \in \mathcal{S}^2$, on définit l'affectation $\sigma_{s_{pre},s_{now}}$ sur les variables de $\mathtt{P} \cup \mathtt{P_{pre}}$ telle que*

— $\forall \mathtt{p} \in \mathtt{P}, \sigma_{s_{pre},s_{now}}(\mathtt{p}) = s_{now}(\mathtt{p})$ *et* $\sigma_{s_{pre},s_{now}}(pre(\mathtt{p})) = s_{pre}(\mathtt{p})$.

On considère qu'une paire d'états appartient à la relation de transition Δ si et seulement si l'affectation correspondante satisfait les formules de Δ_p.

Définition 20 (Transition). *Une paire d'états $(s_{pre}, s_{now}) \in \mathcal{S}^2$ est une transition, notée $(s_{pre}, s_{now}) \in \Delta$, si et seulement si $\sigma_{s_{pre},s_{now}} \models \Delta_p$.*

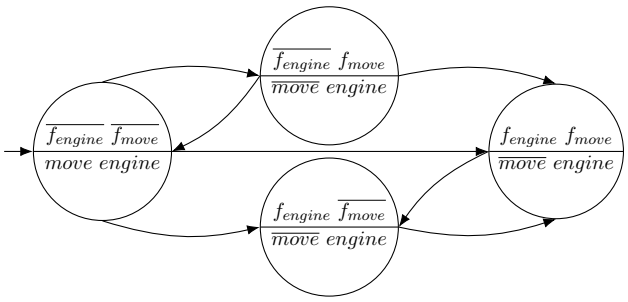


FIGURE 3.1 – Représentation en machine de Moore du système décrit par le modèle comportemental de l'exemple 2, certaines transitions sont volontairement omises par soucis de clarté. La partie haute de l'état représente les variables estimées, la partie basse les variables observées.

Dans toute la suite de ce document, on ne distinguera pas Δ et Δ_p. Pour décrire les différentes évolutions possibles des états du système et les observations associées, on définit les séquences d'états et les séquences d'observations cohérentes.

Définition 21 (Séquence d'états cohérente). *Une séquence d'états* $(s_0, s_1, \ldots, s_n) \in \mathcal{S}^n$ *est cohérente si et seulement si* $\forall i \in [1, n]$, $(s_{i-1}, s_i) \in \Delta$.

Définition 22 (Séquence d'observations cohérente). *Une séquence d'observations* $(o_0, o_1, \ldots, o_n) \in \mathcal{O}^n$ *est cohérente si et seulement s'il existe une séquence d'états cohérente* $(s_0, s_1, \ldots, s_n)$ *telle que* $\forall i \in [0, n], \forall \mathtt{o} \in \mathtt{O}, s_i(\mathtt{o}) = o_i(\mathtt{o})$.

On parlera généralement de séquence d'états et de séquence d'observations pour désigner respectivement les séquences d'états cohérentes et les séquences d'observations cohérentes lorsqu'il n'y a pas d'ambiguïté. Étant donnés un état précédent et une observation du système, on définit l'ensemble des candidats au diagnostic comme l'ensemble des états compatibles avec ceux-ci et le modèle comportemental Δ.

Définition 23 (Candidats au diagnostic). *Soient un état précédent* s_{pre} *et une observation* o. *On considère une fonction cands* : $\mathcal{S} \times \mathcal{O} \rightarrow \mathcal{S}$ *permettant d'extraire l'ensemble des candidats au diagnostic* $cands(s_{pre}, o)$ *pour le couple* (s_{pre}, o) *telle que :*

$$cands(s_{pre}, o) = \left\{ s_{now} \in \mathcal{S} \,|(s_{pre}, s_{now}) \in \Delta \text{ et } \forall \mathtt{o} \in \mathtt{O}, s_{now}(\mathtt{o}) = o(\mathtt{o}) \right\}$$

Exemple 3 (Candidats pour l'exemple 2). *A partir de l'état initial* s_0 *et en recevant l'observation* $\mathtt{o}_1 = \overline{move}$ *engine, on peut extraire trois candidats cohérents avec le modèle comportemental* Δ *:*

- $\mathtt{s}_1 = \overline{move}$ *engine* $\overline{f_{engine}}$ f_{move}
- $\mathtt{s}'_1 = \overline{move}$ *engine* f_{engine} $\overline{f_{move}}$
- $\mathtt{s}''_1 = \overline{move}$ *engine* f_{engine} f_{move}

3.1.3 Le modèle de préférences conditionnelles

Pour un état précédent et une observation, dans le but de produire un diagnostic unique, il est nécessaire de choisir un unique état parmi les candidats au diagnostic. Ce choix est réalisé grâce à un modèle de préférences conditionnelles, noté Γ, qui se présente sous la forme d'un ensemble ordonné de préférences conditionnelles inspiré du formalisme de [Boutilier et al., 2004].

Une préférence conditionnelle porte sur une variable du système que l'on estime et indique sous quelle condition on *préfère* estimer celle-ci à vrai plutôt qu'à faux. Cette variable est appelée « cible de la préférence ». Une préférence peut être vue comme une contrainte souple puisque celle-ci n'est appliquée uniquement lorsque chacune des valeurs booléennes

est possible pour la variable cible par rapport à l'état précédent, l'observation et le modèle comportemental.

Définition 24 (Préférence conditionnelle). *Une préférence conditionnelle γ porte sur une variable* `e` *de* `E` *et est écrite* `cond` $: e \prec \overline{e}$. *La condition de la préférence* `cond` *est une formule propositionnelle sur* $\mathtt{P} \cup \mathtt{P}_{\mathtt{pre}}$. *La variable* `e` *est la cible de la préférence.*

Une préférence est utilisée afin de sélectionner l'estimation préférée parmi les candidats au diagnostic. Ainsi, les états comparés sont toujours des états qui proviennent du même état précédent estimé et qui ont la même observation.

On peut noter que `cond` $: \overline{e} \prec e$ est équivalent à $\neg$`cond` $: e \prec \overline{e}$. Pour une variable `e` de `E`, la préférence conditionnelle `cond` $: e \prec \overline{e}$ exprime une préférence pour les états dans lesquels `e` est vraie par rapport à ceux dans lesquels to `e` est fausse si et seulement si `cond` est satisfaite. Formellement, une préférence $\gamma =$ `cond` $: e \prec \overline{e}$ définit un ordre partiel $\prec_\gamma$ et une relation d'équivalence $\approx_\gamma$ entre les paires de transition comme suit.

- Pour tout triplet $(s_{pre}, s, s') \in \mathcal{S}^3$, γ préfère strictement la transition (s_{pre}, s) à la transition (s_{pre}, s') (noté $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$) si et seulement si $\sigma_{s_{pre},s} \models$ `cond` $\leftrightarrow$ `e` et $\sigma_{s_{pre},s'} \not\models$ `cond` $\leftrightarrow$ `e`. Informellement, si $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$ alors cela signifie que dans la première transition il y a équivalence entre la satisfaction de la condition *cond* et la cible e, mais pas dans la deuxième transition.
- Les transitions (s_{pre}, s) et (s_{pre}, s') sont équivalentes vis-à-vis de γ (noté $(s_{pre}, s) \approx_\gamma (s_{pre}, s')$) si et seulement si $(\sigma_{s_{pre},s} \models$ `cond` $\leftrightarrow$ `e`$) \Leftrightarrow (\sigma_{s_{pre},s'} \models$ `cond` $\leftrightarrow$ `e`$)$.

On suppose que dans Γ, chaque variable estimée de `E` est la cible d'une seule et unique préférence conditionnelle. Bien que cela ne soit pas toujours nécessaire suivant la nature de (s_0,Δ), cela garantit un choix déterministe de l'état estimé. D'après [Boutilier et al., 2004], l'état estimé peut différer suivant l'ordre dans lequel on applique les préférences. Afin de déterminiser le diagnostic, on impose que Γ soit un ensemble totalement ordonné de préférences.

Définition 25 (Modèle de préférences conditionnelles). *Un modèle de préférences conditionnelles Γ est une séquence de préférences conditionnelles $(\gamma_1, \gamma_2, .., \gamma_n)$ avec $\gamma_i =$* $\mathtt{cond_i} : e_i \prec \overline{e_i}$ *pour tout $i \in [1, n]$ telle que chaque variable* `e` *de* `E` *est la cible d'exactement une préférence.*

Exemple 4 (Modèle de préférences pour un robot autonome).

$$\Gamma = \begin{pmatrix} \mathtt{pre\_f_{move}} : f_{move} \prec \overline{f_{move}} & (\gamma_1) \\ \neg\mathtt{engine} : f_{engine} \prec \overline{f_{engine}} & (\gamma_2) \end{pmatrix}$$

Le modèle de préférence implémente les hypothèses suivantes pour le choix de l'état estimé : si il y avait un glissement estimé à l'état précédent, alors on préfère les états dans lesquels le glissement est toujours présent (γ_1). Si le moteur n'est pas sous tension, alors on préfère les états dans lesquels une faute est présente dans le moteur (γ_2). Nous ajoutons que Γ doit être un modèle de préférences acyclique afin de garantir sa cohérence (voir [Boutilier et al., 2004]).

Exemple 5 (Préférences cycliques). *Soient* `a` *et* `b` *deux variables booléennes. On considère deux préférences :*

- $(\gamma_1) =$ `a` $: b \prec \overline{b}$
- $(\gamma_2) =$ `b` $: a \prec \overline{a}$

(γ_1) et (γ_2) forment un cycle indéfini dans notre interprétation. Cela donne lieu à des sorties incomparables allant à l'encontre de l'exigence d'un diagnostic déterministe.

On impose donc que la condition d'une préférence γ_i ne puisse contenir d'autres variables qui sont déjà la cible d'une préférence de plus haut rang dans la séquence. Formellement, $\forall i \in [1, n]$, la condition $\mathtt{cond_i}$ ne peut porter que sur un sous-ensemble de $\mathtt{P_{pre}} \cup \mathtt{O} \cup \{\mathtt{e_j} | 1 \leq j < i\}$.

Étant donné un modèle de préférences Γ, il est possible de définir un ordre partiel $\prec_\Gamma$ entre les paires d'états de la manière suivante. On considère les préférences γ_i dans l'ordre de leur index dans la séquence Γ et on applique chaque préférence suivant cet index dans la relation d'ordre $\prec_{\gamma_i}$ définie plus tôt. Cet ordre est partiel car il ne compare que des paires de transition (s_{pre}, s_{now}) et $(s_{pre}{}', s_{now}{}')$ ayant le même état précédent, c'est-à-dire $s_{pre} = s_{pre}{}'$ et dont les états successeurs produisent la même observation, c'est-à-dire $\forall \mathtt{o} \in \mathtt{O}, s_{now}(\mathtt{o}) = s_{now}{}'(\mathtt{o})$).

Définition 26 (Ordre partiel induit par Γ). *$\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) est strictement préféré à (s_{pre}, s') par Γ (noté $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$) si et seulement s'il existe $i \in [1, n]$ tel que pour tout $j < i$, $(s_{pre}, s) \approx_{\gamma_j} (s_{pre}, s')$ et $(s_{pre}, s) \prec_{\gamma_i} (s_{pre}, s')$.*

Bien que l'ordre $\prec_\Gamma$ soit partiel sur $\mathcal{S}^2$, celui-ci est complet sur chaque ensemble de candidats au diagnostic comme montre la proposition suivante.

Proposition 3. *Soient s_{pre} un état, o une observation, Δ un modèle comportemental et Γ un modèle de préférences. Si s et s' sont deux candidats au diagnostic pour s_{pre} et o $(s, s') \in cands(s_{pre}, o)^2$ tels que $s \neq s'$ alors soit $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$, soit $(s_{pre}, s') \prec_\Gamma (s_{pre}, s)$.*

Démonstration. Par la définition 23, s et s' appartiennent à $cands(s_{pre} o)$, cela implique que $\forall \mathtt{o} \in \mathtt{O}, s(\mathtt{o}) = s'(\mathtt{o})$. Comme $s \neq s'$, cela signifie qu'il existe une variable $\mathtt{e} \in \mathtt{E}$ telle que $s(\mathtt{e}) \neq s'(\mathtt{e})$. Donc, il existe une préférence $\gamma \in \Gamma$ telle que $s \approx_\gamma s'$ est faux. Soit i l'index de la première préférence de la séquence dans ce cas. Formellement, γ_i est telle que $\forall j < i$, $(s_{pre}, s) \approx_{\gamma_j} (s_{pre}, s')$, $(s_{pre}, s) \approx_{\gamma_i} (s_{pre}, s')$ est faux. Cela signifie que $(s_{pre}, s) \prec_{\gamma_i} (s_{pre}, s')$ ou $(s_{pre}, s') \prec_{\gamma_i} (s_{pre}, s)$. D'après la définition de l'ordre $\prec_\Gamma$, on a donc $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$ ou $(s_{pre}, s') \prec_\Gamma (s_{pre}, s)$. ■

Cette relation d'ordre complète est utilisée pour sélectionner un unique état préféré parmi les candidats au diagnostic à chaque étape discrète lors du processus d'estimation. Comme dit précédemment, une préférence peut être vue comme une contrainte souple puisque celle-ci n'est appliquée uniquement lorsque chacune des valeurs booléennes est possible pour la variable cible par rapport à l'état précédent, l'observation et le modèle comportemental du système.

3.2 Calcul des estimations

3.2.1 Processus d'estimation

Puisque l'estimation se base sur l'état précédemment estimé et sur l'observation reçue, on parle d'estimation incrémentale. À chaque étape discrète, le diagnostic émis respecte la dynamique du système par rapport à l'état estimé à l'étape précédente. L'estimation se base donc sur :

- la dynamique du système (Δ)
- l'observation reçue
- l'état précédemment estimé (s_0 à l'étape 1)

À chaque étape discrète, étant donnés les candidats au diagnostic $cands(s_{pre}, o)$ pour un état précédent s_{pre} et une observation o, on sélectionne parmi ceux-ci l'état préféré par le modèle de préférences conditionnelles Γ. On peut considérer Γ comme une fonction partielle $estim : S \times O \to S$ puisque celle-ci produit un état estimé pour un état précédent et une observation. La figure 3.2 décrit le processus d'estimation incrémental.

Définition 27 (État estimé). *L'état estimé $\hat{s} = estim(s_{pre}, o)$ pour un état précédent s_{pre} et une observation o est l'élément de $cands(s_{pre}, o)$ préféré par $\prec_\Gamma$. Formellement, $\hat{s} = estim(s_{pre}, o)$ si :*

1. *$\hat{s} \in cands(s_{pre}, o)$, et*
2. *$\forall s_{now} \in cands(s_{pre}, o)$ tel que $s_{now} \neq \hat{s}$, on a $(s_{pre}, \hat{s}) \prec_\Gamma (s_{pre}, s_{now})$.*

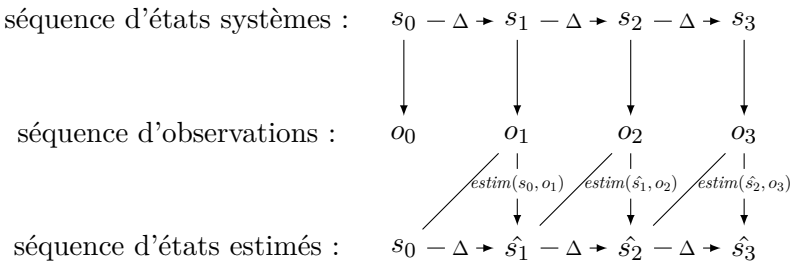


FIGURE 3.2 – Le processus d'estimation incrémental

On définit maintenant les séquences d'états estimés qui sont produites par Γ à partir des séquences d'observations produites par (s_0, Δ).

Définition 28 (Séquence d'états estimés). *Une séquence d'états $(s_0, \hat{s}_1, \hat{s}_2, ..., \hat{s}_k)$ est la séquence d'états estimés pour une séquence d'observations $(o_0, o_1, o_2, ..., o_k)$ si et seulement si $\forall i \in [1, k], \hat{s}_i = estim(\hat{s}_{i-1}, o_i)$.*

Exemple 6 (Estimation pour un robot autonome). *On considère le modèle comportemental Δ et l'état initial s_0 de l'exemple 2 et on y associe le modèle de préférences conditionnelles Γ de l'exemple 4 afin de produire des estimations. A partir de l'état initial s_0 et en recevant l'observation $\mathtt{o_1} = \overline{move}engine$ on retrouve les candidats au diagnostic de l'exemple 3 :*

- *s_1 = $\overline{move}$ engine $\overline{f_{engine}}$ f_{move}*
- *s'_1 = $\overline{move}$ engine f_{engine} $\overline{f_{move}}$*
- *s''_1 = $\overline{move}$ engine f_{engine} f_{move}*

La première préférence, notée γ_1, indique qu'on préfère estimer $\mathtt{f_{move}}$ à sa valeur estimée à l'étape précédente. Un tel mécanisme permet d'apporter une certaine stabilité au diagnostic puisque $\mathtt{f_{move}}$ est une faute intermittente : si Δ permet les deux valeurs booléennes pour $\mathtt{f_{move}}$, on préfère maintenir le diagnostic choisi à l'étape précédente.

En appliquant en premier la préférence (γ_1), comme on a $\overline{pre\_f_{move}}$ dans les trois états, on préfère les états dans lesquels on a $\overline{f_{move}}$. Dans ce cas précis, seul s'_1 satisfait ce critère. On retrouve donc $(s_0, s'_1) \prec_\Gamma (s_0, s_1)$ et $(s_0, s'_1) \prec_\Gamma (s_0, s''_1)$ dans la relation d'ordre induite par Γ. Puisqu'on a déjà éliminé tous les candidats sauf un, la préférence (γ_2) n'est pas appliquée. L'unique état préféré est alors $\hat{s}_1 = estim(s_0, o_1) = s'_1 = \overline{move}$ engine f_{engine} $\overline{f_{move}}$.

3.2.2 Description de l'estimateur en logique temporelle

Il est possible de décrire les contraintes de Δ et de Γ en utilisant la logique *PTLTL* présentée dans le chapitre précédent (définition 16 page 17). L'utilisation la plus simple est l'utilisation de l'opérateur `Y` (Yesterday) faisant référence à la valeur de la variable cible à l'état précédent.

- La deuxième règle du modèle comportemental Δ de l'exemple 2 $\mathtt{pre\_f_{engine}} \rightarrow \mathtt{f_{engine}}$ devient $\mathtt{Y(f_{engine})} \rightarrow \mathtt{f_{engine}}$
- La première préférence du modèle Γ de l'exemple 4 $\mathtt{pre\_f_{move}} : f_{move} \prec \overline{f_{move}}$ devient $\mathtt{Y(f_{move})} : f_{move} \prec \overline{f_{move}}$

Cela permet notamment de n'écrire que des contraintes portant sur l'ensemble P, on abstrait alors l'ensemble P_{pre}. Utiliser *PTLTL* permet d'écrire des contraintes plus compréhensibles pour le concepteur, c'est d'ailleurs la logique utilisée dans l'article [Pralet et al., 2016] afin de décrire les modèles de diagnostic.

Les estimateurs de la forme (s_0,Δ,Γ) qui reposent sur un formalisme à base de contraintes peuvent être encodés pour des solvers de type SAT (voir chapitre 2). Cet encodage a pour but à partir d'un état précédent s_{pre} et d'une observation o de déterminer l'état estimé.

On décrit ici deux manières d'encoder le processus d'estimation :

1. avec un solveur SAT classique
2. avec un solveur Max-SAT

Pour chaque encodage on transforme les contraintes de Δ en forme normale conjonctive. Les valeurs des variables constituant l'état précédent et l'observation reçue sont imposées. L'application des préférences est cependant différente suivant l'approche adoptée.

Pour un solver SAT, on considère que le solver admet une fonction isSAT() qui renvoie vrai (et un modèle) si la formule en entrée est satisfiable, faux sinon. On teste dans un premier temps isSAT($\Delta \wedge pre(s_{pre}) \wedge o$). Si le solver renvoie faux alors c'est qu'il n'y a pas d'état successeur possible. Sinon c'est qu'il a un ou plusieurs états successeurs possibles et on doit choisir parmi ceux-ci quel est l'état estimé grâce aux préférences. On rappelle ensuite le solver de manière itérative avec la condition des préférences et la valeur de leur cible possiblement fixée par une préférence de plus haut rang. Pour la première préférence de la forme $c_1 : e_1 \prec \overline{e_1}$ on teste donc isSAT($\Delta \wedge pre(s_{pre}) \wedge o \wedge c_1$). Si dans le modèle trouvé $\mathtt{e_1}$ est vrai on testera alors ensuite pour la deuxième préférence isSAT($\Delta \wedge pre(s_{pre}) \wedge o \wedge e_1 \wedge c_2$). On continue ainsi jusqu'à traiter toutes les préférences et le modèle renvoyé par le solver est donc l'état estimé.

Pour un solver Max-SAT, on considère que la contrainte ($\Delta \wedge pre(s_{pre}) \wedge o$) est une contrainte dure. On ajoute ensuite une contrainte souple pour chaque préférence de la forme $cond \leftrightarrow e$ avec un poids permettant de produire un ordre lexicographique sur l'application des préférences suivant leur rang dans la séquence grâce aux puissance de 2. Pour la première préférence on fixe un poids de 2^n avec n le nombre de préférences de Γ, pour la deuxième 2^{n-1}, etc , jusqu'à la dernière préférence qui aura un poids de 2^1.

3.3 Problématique

La sélection d'un unique état estimé parmi les candidats à chaque étape discrète peut parfois mener à des incohérences entre la trajectoire d'états estimés et la séquence d'observations reçue, on parle alors de scénario d'impasse. Le phénomène de l'impasse est illustré ici et l'impasse sera définie dans le chapitre suivant.

Exemple 7 (Scénario d'impasse dans l'estimation). *On reprend s_0, Δ et Γ des exemples 2 et 4. On considère que l'observation $\mathtt{o_1} = \overline{move}engine$ est reçue. Conformément à l'exemple précédent, en appliquant Γ, on sélectionne donc l'état estimé $\hat{s}_1 = {s'}_1 = \overline{move}engine$. On estime donc qu'une faute affecte le moteur.*

Cependant, il est possible que pour la séquence d'observations (o_0, o_1), le système ai évolué dans l'état $\mathtt{s_1} = \overline{move}$ engine $\overline{f_{engine}}$ f_{move} puis revienne dans l'état initial à l'étape suivante en produisant l'observation $o_2 = o_0 = move\ engine$. Lorsque l'estimateur essaye de générer les candidats pour o_2 et son état précédent ${s'}_1$, il est impossible de trouver une affectation de f_{engine} et f_{move} satisfaisant Δ.

Il existe une séquence d'états estimés pour la séquence d'observations (o_0, o_1, o_2) : c'est la séquence $(s_0, \hat{s}_1)$. Cependant, il n'y a pas de séquence d'états estimés pour la séquence d'observations (o_0, o_1, o_2), l'estimateur rencontre une impasse dans l'estimation et perd la cohérence avec le modèle comportemental du système. On appelle un tel phénomène un scénario d'impasse.

Étape i	o_i	s_i	$\hat{s}_i$
0	*move engine*	$\overline{f_{engine}}\ \overline{f_{move}}$	$\overline{f_{engine}}\ \overline{f_{move}}$
1	$\overline{move}$ *engine*	$\overline{f_{engine}}\ f_{move}$	$f_{engine}\ \overline{f_{move}}$
2	*move engine*	$\overline{f_{engine}}\ \overline{f_{move}}$	/

FIGURE 3.3 – Scénario d'impasse illustré dans l'exemple 7. Les colonnes s_i et $\hat{s}_i$ représentent respectivement les états du système et les états estimés pour l'étape i. On omet les variables de `O` dans les deux dernières colonnes car celles-ci sont identiques à celles de la colonne o_i, et on ne représente que les variables estimés de `E`.

Le phénomène de l'impasse ne touche pas forcément tous les modèles définis avec le formalisme (s_0,Δ, Γ). Dans cet exemple, la situation d'impasse pourrait être éliminée en changeant l'ordre ou les conditions des préférences.

Exemple 8 (Modèle d'estimation sans impasse). *Pour le modèle (s_0,Δ) de l'exemple 2, il est possible de construire un modèle de préférences qui ne rencontre pas d'impasse. Par exemple en inversant l'ordre des préférences de l'exemple 4, on obtient alors le modèle* Γ′ *suivant :*

$$\Gamma' = \left(\begin{array}{cc} \neg\texttt{engine} : f_{engine} \prec \overline{f_{engine}} & (\gamma_1) \\ \texttt{pre\_f}_{\texttt{move}} : f_{move} \prec \overline{f_{move}} & (\gamma_2) \end{array} \right)$$

En partant de l'état initial s_0 et en recevant l'observation $o_1 = \overline{move}engine$, on obtient toujours les même candidats de l'exemple 3. Cependant, l'état estimé sera différent puisque les préférences ne sont pas évaluées dans le même ordre. Puisque `engine` *est vraie, la première préférence (γ_1) élimine les candidats dans lesquels la faute* $\texttt{f}_{\texttt{engine}}$ *est présente. Il ne reste alors que l'état $s_1 = \overline{move}engine\overline{f_{engine}}f_{move}$ qui sera donc l'état unique estimé. Or si le système reçoit ensuite l'observation $o_2 = o_0 = moveengine$ comme dans l'exemple 7, il existe des candidats pour l'état précédent s_1 et l'observation o_2. L'état unique estimé sera alors s_0 (le seul candidat dans ce cas précis). Il existe donc une séquence d'états estimés $(\hat{s_0}, \hat{s_1}, \hat{s_2})$ en utilisant* Γ′ *qui permet une estimation de la séquence d'observations (o_0, o_1, o_2) avec :*

- $\hat{s_0} = s_0$
- $\hat{s_1} = s_1$
- $\hat{s_2} = s_0$

Cependant, dès lors que l'on s'intéresse à des systèmes plus complexes, il devient difficile de détecter ces scénarios d'impasse et encore plus de les résoudre. Il est donc souhaitable d'être capable, à partir de la description d'un système et de sa stratégie d'estimation, de détecter la possibilité de rencontrer un scénario d'impasse dans l'estimation.

Définition 29 (Fonction estimateur). *Pour un estimateur, on définit une fonction partielle estimSeq qui prend en entrée une séquence d'observations et qui retourne la séquence d'estimation lorsque celle-ci existe. Formellement pour une séquence d'observations $seqObs \in \mathcal{L}_{obs}(\Delta)$, $estimSeq : O^* \to S^*$ est définie telle que :*

$$estimSeq = \left\{ \begin{array}{l} (\hat{s}_0, \hat{s}_1, \hat{s}_2, .., \hat{s}_n) \text{ tel que } \forall i \in 1..n,\ \hat{s}_i = estim(\hat{s}_{i-1}, obs(seqObs(i))) \\ \text{indéfinie sinon (impasse)} \end{array} \right\}$$

Les problématiques de ce document s'articulent autour du problème de l'impasse. Dans le chapitre 4, la possibilité de **détecter** un scénario d'impasse dans l'estimation est étudiée. À partir d'un scénario d'impasse, on cherche à **blâmer** certaines des préférences dans le chapitre 5. Enfin, le chapitre 6 explore la possibilité de **construire** des estimateurs sans scénario d'impasse à partir des spécifications d'un système.

Deuxième partie

Contributions

Chapitre 4

Détection d'un scénario d'impasse

Dans le chapitre précédent, un formalisme permettant de réaliser l'estimation à état unique des systèmes à évènements discrets a été présenté. Cependant, l'estimation à état unique est sujette à certaines limites, notamment les impasses dans l'estimation, comme illustré précédemment.

Certaines approches de la littérature proposent des processus d'estimation incrémentale et des mécanismes permettant la gestion des impasses. [Grastien et al., 2005] propose de remettre à zéro le processus d'estimation lorsque pendant l'exécution la trajectoire d'états estimés n'est plus cohérente avec la dynamique du système. Ce mécanisme a cependant pour inconvénient de perdre momentanément la cohérence avec la dynamique du système. Les auteurs de [Kurien and Nayak, 2000] proposent un retour en arrière dans la séquence et reviennent sur un choix de transition pour retrouver la cohérence. Ces mécanismes ne sont pas envisageables dans notre approche car coûteux en terme de temps de calcul et ce n'est pas souhaitable lorsqu'on traite d'architectures autonomes souvent limitées en ressources. Ce phénomène peut toutefois être atténué, notamment en certifiant une fenêtre (un nombre d'états) sur laquelle on peut revenir en arrière [Su et al., 2014]. Cependant, le module de planification aura certainement envisagé des actions, basées sur les précédentes estimations, sur lesquelles un retour en arrière est souvent impossible. De même, autoriser l'estimateur à franchir des transitions qui ne font pas partie du modèle peut poser des problèmes par rapport au module de planification. C'est pourquoi dans notre approche on interdit le processus de retour en arrière sur les états estimés, afin de garantir une estimation toujours cohérente au fil du temps et peu coûteuse en terme de calcul, ce qui est impossible lorsque l'estimateur risque de rencontrer des impasses.

Le chapitre précédent présentait le processus d'estimation en ligne utilisé dans ce document. Dans ce chapitre, on s'intéresse à la vérification hors-ligne de propriétés pour un système et son estimateur. On étudie ici à la possibilité de détecter les scénarios d'impasse lors de la phase de conception de l'estimateur.

Pour cela on formalise dans un premier temps le phénomène de l'impasse pour un système et son estimateur. Ensuite, on s'intéresse à la vérification de la présence de scénario d'impasse à partir des spécifications d'un système et de son estimateur. La méthode utilisée s'inspire des approches de type Twin-Plant [Jiang et al., 2001], [Cimatti et al., 2003], [Pencole, 2005] une méthode permettant à l'origine de vérifier la diagnosticabilité d'un système. On adapte cette méthode afin de pouvoir simuler l'exécution d'un système et de son estimateur et détecter un ou plusieurs scénarios d'impasse s'ils existent. Cette méthode formelle est, lors d'une première approche, encodée en *Electrum*, un outil de model-checking adapté pour des systèmes à dynamique à la fois complexe et temporelle. La deuxième approche consiste à simuler l'exécution du système par l'intermédiaire de requêtes itératives en SAT puis de tenter de générer des séquences d'estimations avec un solver MAX-SAT. Chaque approche est accompagnée d'expérimentations.

4.1 Formalisation de l'impasse

On s'intéresse dans un premier temps à la formalisation du phénomène de l'impasse entre un système et son estimateur. On propose une définition pour les scénarios d'impasse. Un estimateur est en situation d'impasse lorsque celui-ci ne peut plus fournir une estimation cohérente par rapport au modèle comportemental du système et la séquence d'observations reçue. Plus généralement, on considère un scénario d'impasse lorsqu'il existe une séquence d'états estimés pour laquelle il n'existe aucune continuation possible par rapport à la séquence d'observations reçue.

Définition 30 (Impasse). *Un modèle d'estimation* (s_0, Δ, Γ) *rencontre une impasse pour une séquence d'observations* $(o_0, o_1, o_2, \ldots, o_k)$ *avec* $k > 1$ *si et seulement si :*

1. *il existe une séquence d'états estimés pour* $(o_0, \ldots, o_{k-1})$*, et*
2. *il n'existe aucune séquence d'états estimés pour* $(o_0, \ldots, o_k)$

Remarque. *Il ne peut y avoir d'impasse à* $\hat{s}_1$ *car on suppose que l'estimateur est correctement initialisé à* s_0*.*

Dans l'exemple 7 vu en fin de chapitre précédent, le scénario d'impasse peut être éliminé simplement en inversant l'ordre des préférences. Il existe d'autres manières de modifier Γ afin d'éviter les impasses, par exemple en modifiant les conditions de certaines préférences. Cette problématique sera traitée dans le chapitre suivant.

On s'intéresse dans ce chapitre à la recherche de scénarios d'impasse dans les modèles d'estimation en amont de leur déploiement afin d'anticiper ces situations avant qu'elles ne surviennent en mission. On peut ainsi valider le bon fonctionnement d'un estimateur si celui-ci n'est pas sujet aux scénarios d'impasse ou dans le cas contraire modifier les contraintes du modèle ou les préférences de l'estimateur. Les deux prochaines sections présentent deux approches qui permettent la détection hors ligne d'un scénario d'impasse dans l'estimation.

4.2 Approche model-checking

Cette section présente la première approche mise en oeuvre afin de vérifier la présence de scénarios d'impasse dans les modèles d'estimation par l'intermédiaire du model-checker *Electrum*. Dans cette section, nous décrivons comment adapter la technique du Twin-plant pour la recherche de scénario d'impasse. Puis, nous expliquerons comment implémenter une machine témoin en *Electrum* afin de détecter les scénarios d'impasse à partir des spécifications d'un système et de son modèle d'estimation.

4.2.1 Encodage d'une machine témoin

Afin de détecter la présence de scénarios d'impasse, on propose une méthode de vérification inspirée de la technique du Twin-plant [Cimatti et al., 2003] qui s'utilise à l'origine pour vérifier la diagnosticabilité d'un système en construisant le produit synchrone de la machine à états du système.

La méthode utilisée ici est similaire puisqu'on construit une machine à états appelée « machine témoin » en synchronisant deux machines à états. La première machine à états représente le système initialement dans l'état s_0 et dont la dynamique est formalisée par la relation de transition Δ. Cette première machine est utilisée afin de générer toutes les exécutions possibles du système et donc d'obtenir toutes les séquences d'observations possibles. La deuxième machine à états représente l'estimateur. Celle-ci est aussi contrainte par s_0 et Δ, mais choisit de manière déterministe l'état suivant en appliquant les préférences de Γ.

La machine témoin est le produit de ces deux machines à états synchronisées sur les variables observables à chaque étape discrète. La vérification de scénario d'impasse est

ensuite réalisée en vérifiant s'il existe une séquence d'observations qui mènerait la machine témoin dans un état où le système aurait au moins un successeur mais l'estimateur n'en aurait aucun.

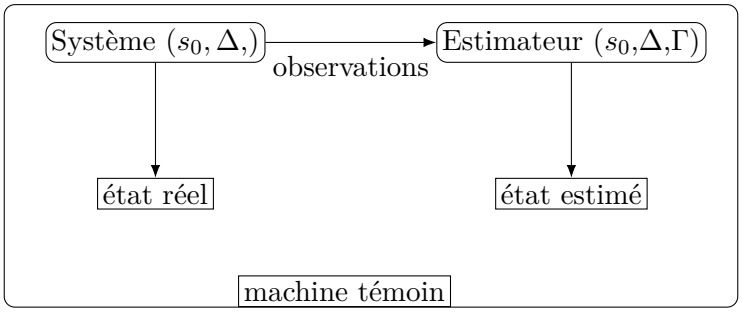


FIGURE 4.1 – Schéma de la machine témoin

La figure 4.1 schématise le fonctionnement de la machine témoin que l'on cherche à construire. On simule d'un côté le fonctionnement du système contraint par les règles de Δ produisant un état réel. Puis on simule l'estimateur à état unique (s_0,Δ,Γ) synchronisé sur les observations du système et produisant un état estimé. Afin de vérifier l'absence de scénario d'impasse, on cherche à savoir s'il existe un état estimé pour chaque état réel du système.

Le concept de préférence est partie intégrante de la définition de la machine témoin. Afin de les traduire dans le langage *Electrum* que nous utiliserons, nous proposons la définition d'une préférence sous sa forme quantifiée. On utilise les opérateurs $\forall$ et $\exists$ de l'algèbre relationnelle. Le quantificateur $\forall$ est utilisé pour exprimer les deux évaluations booléennes d'une variables. $\forall x\ \phi$ est équivalent à $\phi(x = \top) \wedge \phi(x = \bot)$ où $\phi(x = \top)$ (respectivement $\phi(x = \bot)$) représente la formule ϕ dans laquelle toutes les instances de x sont remplacées par $\top$ (respectivement $\bot$). Cela permet d'exprimer qu'il existe plusieurs états candidats avec différentes valeurs booléennes pour la variable cible de la préférence, c'est-à-dire qu'un choix existe pour la variable cible. Pour $\exists$ on a $\exists x\ \phi$ équivalent à $\phi(x = \top) \vee \phi(x = \bot)$. Ce quantificateur est utilisé pour exprimer le fait qu'il existe une évaluation booléenne pour les variables cibles d'une préférence de plus faible rang, c'est à dire que le choix effectué reste compatible avec le modèle comportemental. La combinaison de ces deux quantificateurs permet de définir la possibilité d'appliquer une préférence.

Définition 31 (Forme quantifiée d'une préférence). *Soit $\Gamma = (\gamma_1, \ldots, \gamma_n)$ un modèle de préférences dans lequel chaque préférence est de la forme $\gamma_i = \mathtt{cond_i} : e_i \prec \overline{e_i}$. La forme quantifiée d'une préférence γ_i est la formule ψ_i définie par :*

$$\psi_i = (\forall \mathtt{e_i}, \exists \mathtt{e_{i+1}}, \ldots, \mathtt{e_n},\ \Delta) \rightarrow (\mathtt{e_i} \leftrightarrow \mathtt{cond_i})$$

La forme quantifiée ψ_i se compose de deux parties. La partie gauche est uniquement satisfaite lorsque chaque valeur booléenne est possible pour la variable (cible de la préférence) $\mathtt{e_i}$, tout en restant cohérent avec Δ. Cela représente les cas où la préférence est appliquée, c'est-à-dire si Δ et l'application des préférences de plus haut rang permettent un choix sur la variable e_i. La partie droite exprime les effets de la préférence, c'est-à-dire que $\mathtt{e_i}$ est affectée à vrai si et seulement si la condition $\mathtt{cond_i}$ est satisfaite.

Exemple 9 (Forme quantifiée des préférence pour l'exemple 4). *On reprend le modèle Γ étudié dans les chapitres précédents :*

$$\Gamma = \begin{pmatrix} \mathtt{pre\_f_{move}} : f_{move} \prec \overline{f_{move}} & (\gamma_1) \\ \neg \mathtt{engine} : f_{engine} \prec \overline{f_{engine}} & (\gamma_2) \end{pmatrix}$$

Les deux préférences sous leurs forme quantifiée deviennent :

— $\psi_1 = (\forall \mathtt{f_{move}}, \exists \mathtt{f_{engine}}, \Delta) \rightarrow (\mathtt{f_{move}} \leftrightarrow \mathtt{pre\_f_{move}})$
— $\psi_2 = (\exists \mathtt{f_{engine}}, \Delta) \rightarrow (\mathtt{f_{engine}} \leftrightarrow \neg \mathtt{engine})$

Les préférences sous leurs formes quantifiées peuvent s'interpréter de la manière suivante : lorsque quelle que soit la valeur de $\mathtt{f_{move}}$*, il existe une valeur de* $\mathtt{f_{engine}}$ *qui satisfait* Δ*, alors on préfère* $\mathtt{f_{move}}$ *à sa valeur dans l'état précédent (*ψ_1*). Lorsque quelle que soit la valeur de* $\mathtt{f_{engine}}$ *on peut satisfaire* Δ*, alors on préfère la valeur opposée à celle de* engine *(*ψ_2*)*

À travers la proposition suivante, on montre que les préférences de Γ et leurs formes quantifiées sont équivalentes du point de vue de l'état estimé pour un état précédent et une observation.

Proposition 4. *Pour un état précédemment estimé* $\hat{s}_{pre}$*, une observation* o*, un candidat* $\hat{s} \in cands(\hat{s}_{pre}, o)$*,* $\hat{s} = estim(\hat{s}_{pre}, o)$ *si et seulement si l'affectation* $\sigma_{\hat{s}_{pre},\hat{s}}$ *satisfait la conjonction* $\bigwedge_{i\in[1,n]} \psi_i$*.*

Démonstration. Soit $\hat{s}_{pre}$ un état précédemment estimé, o une observation et $\hat{s}$ un état de $cands(\hat{s}_{pre}, o)$. On montre par induction que $\hat{s}$ est préféré pour la séquence de préférences $(\gamma_1, \ldots, \gamma_k)$ si et seulement si l'affectation $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfait $\bigwedge_{i\in[1,k]} \psi_i$.

— Pour $k = 1$, l'ensemble des variables libres de la formule $pref_1 = \forall \mathtt{e_1}, \exists \mathtt{e_2}, \ldots \mathtt{e_n}, \Delta$ est l'ensemble $\mathtt{O}$, puisque toutes les variables de $\mathtt{E}$ sont quantifiées. Si la formule $pref_1$ est fausse, cela signifie que pour tout s et s' de $cands(\hat{s}_{pre}, o)$, $s(\mathtt{e_k}) = s'(\mathtt{e_k})$. Alors, la préférence γ_k n'est pas appliquée.
Dans l'autre cas, la formule $pref_1$ est satisfaite par une observation o si et seulement si oe_1 et $o\overline{e_1}$ ont une extension dans $\mathtt{P} \cup \mathtt{P_{pre}}$ qui satisfait Δ. Cela signifie que $cands(\hat{s}_{pre}, o)$ contient au moins deux états s et s' avec $s(\mathtt{e_1}) \neq s'(\mathtt{e_1})$. Parmi ces deux états, celui qui satisfait $\mathtt{cond_1} \leftrightarrow \mathtt{e_1}$ est l'état préféré par la préférence γ_1, et satisfait aussi ψ_1. Comme les états préférés par γ_1 sont exactement ceux dans lesquels l'affection $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfait ψ_1, alors la proposition tient pour $k = 1$.

— Pour $k > 1$, on suppose que l'induction précédente est vraie pour $k - 1$, c'est-à-dire que $\hat{s}$ est préféré par la séquence de préférences $(\gamma_1, \ldots, \gamma_{k-1})$ si et seulement si l'affectation $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfait $\bigwedge_{i\in[1,k-1]} \psi_i$. Soient s et s' deux états de $cands(\hat{s}_{pre}, o)$ préférés par $(\gamma_1, \ldots, \gamma_{k-1})$, c'est-à-dire que $\sigma_{\hat{s}_{pre},s'}$ satisfont $\bigwedge_{i\in[1,k-1]} \psi_i$. Nous avons alors pour tout $i < k$, $s(\mathtt{e_i}) = s'(\mathtt{e_i})$.
L'ensemble des variables libres dans la formule $pref_k = \forall \mathtt{e_k}, \exists \mathtt{e_{k+1}}, \ldots \mathtt{e_n}, \Delta$ est l'ensemble $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$. Une affectation op sur ces variables satisfait $pref_k$ si et seulement si $op\ e_k$ et $op\ \overline{e_k}$ chacune ont une extension vers $\mathtt{P} \cup \mathtt{P_{pre}}$ qui satisfait Δ.
  — Si pour une affectation donnée op sur $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$ la formule $pref_k$ est fausse, cela signifie que pour tout s et s' de $cands(\hat{s}_{pre}, o)$, $s(\mathtt{e_k}) = s'(\mathtt{e_k})$. Alors, la préférence γ_k n'est pas appliquée et les états préférés par $(\gamma_1, \ldots, \gamma_{k-1})$ sont aussi tous préférés par $(\gamma_1, \ldots, \gamma_k)$. Dans ce cas, ψ_k est toujours satisfaite et les états s' tels que $\sigma_{\hat{s}_{pre},s'}$ satisfait $\bigwedge_{i\in[1,k-1]} \psi_i$ sont les mêmes pour lesquels $\sigma_{\hat{s}_{pre},s'}$ satisfait $\bigwedge_{i\in[1,k]} \psi_i$ et l'induction est vérifiée.
  — Si pour une affectation donnée op sur $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$ la formule $pref_k$ est vraie, alors $cands(\hat{s}_{pre}, o)$ contient au moins deux états s et s' tels que $s(\mathtt{e_i}) = s'(\mathtt{e_i})$ pour tout $i < k$ et $s(\mathtt{e_k}) \neq s'(\mathtt{e_k})$. Pour ces deux états , celui dans lequel $\mathtt{cond_k} \leftrightarrow \mathtt{e_k}$ est vraie est l'état préféré par la préférence γ_k. Comme les états préférés par γ_k sont exactement ceux pour lesquels l'affectation $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfait ψ_k, alors l'induction est valide pour k.

■

Nous nous intéressons à présent à l'encodage de la machine témoin dans le langage *Electrum*, qui s'appuie sur les formes quantifiées des préférences. Chaque état de cette machine contient un état du système et un état de l'estimateur, ainsi qu'une variable indiquant si

une impasse à eu lieu. Afin de distinguer les états des deux machines, c'est-à-dire les états du système et ceux de l'estimateur, on introduit deux ensembles de variables $\mathtt{P}^{\mathtt{sys}}$ and $\mathtt{P}^{\mathtt{est}}$ comme des copies de $\mathtt{P}$. On utilise aussi une variable `no_impasse` qui indique si l'estimateur admet un état suivant.

Définition 32 (États de la machine témoin). *L'ensemble des variables de la machine témoin est défini par* $\mathtt{P}^{\mathtt{temoin}} = \mathtt{P}^{\mathtt{sys}} \cup \mathtt{P}^{\mathtt{est}} \cup \{\mathtt{no\_impasse}\}$*, dans lequel :*

- $\mathtt{P}^{\mathtt{sys}} = \{\mathtt{p\_sys} \mid \mathtt{p} \in \mathtt{P}\}$ *est l'ensemble des variables décrivant l'état du système,*
- $\mathtt{P}^{\mathtt{est}} = \{\mathtt{p\_est} \mid \mathtt{p} \in \mathtt{P}\}$ *est l'ensemble des variables décrivant l'état de l'estimateur,*
- `no_impasse` *indique s'il existe un état estimé dans l'estimateur et donc que celui-ci n'est pas en situation d'impasse.*

Un état de la machine témoin est une affectation sur les variables de $\mathtt{P}^{\mathtt{temoin}}$*.*

Afin de définir la machine à états résultant du produit synchrone, on définit successivement l'état initial de la machine témoin (définition 33) et sa relation de transition (définition 34). On suppose que lors de l'exécution, le système et son estimateur partagent le même état initial.

Définition 33 (État initial de la machine témoin). *L'état initial* s_0^{temoin} *de la machine témoin est défini tel que :*

- $\forall \mathtt{p} \in \mathtt{P}$, $s_0^{temoin}(\mathtt{p\_sys}) = s_0(\mathtt{p})$,
- $\forall \mathtt{p} \in \mathtt{P}$, $s_0^{temoin}(\mathtt{p\_est}) = s_0(\mathtt{p})$,
- $s_0^{temoin}(\mathtt{no\_impasse}) = \top$.

Définition 34 (Relation de transition de la machine témoin). *Soient* s_{pre}^{temoin} *et* s_{now}^{temoin} *deux états de la machine témoin. La paire* $(s_{pre}^{temoin}, s_{now}^{temoin})$ *est une transition de la machine témoin si et seulement si :*

(4.1) *les variables associées aux observations ont la même valeur dans le système et dans l'estimateur,*

(4.2) *la transition du système décrite par les variables de* $\mathtt{P}^{\mathtt{sys}}$ *satisfait toujours* Δ*,*

(4.3) *la variable* `no_impasse` *indique s'il existe un éventuel état estimé, c'est-à-dire si l'ensemble des candidats est non-vide,*

(4.4) *si* `no_impasse` *est vrai, alors la transition de l'estimateur décrite par les variables de* $\mathtt{P}^{\mathtt{est}}$ *satisfait* Δ *et* Γ*.*

Formellement :

$$\begin{aligned}\forall \mathtt{o} \in \mathtt{O}, s_{pre}^{temoin}(\mathtt{o\_sys}) = s_{pre}^{temoin}(\mathtt{o\_est})\ et\\ \forall \mathtt{o} \in \mathtt{O}, s_{now}^{temoin}(\mathtt{o\_sys}) = s_{now}^{temoin}(\mathtt{o\_est})\end{aligned} \tag{4.1}$$

$$\begin{aligned}\exists (s_{pre}, s_{now}) \in \Delta, \forall \mathtt{p} \in \mathtt{P},\\ s_{pre}(\mathtt{p}) = s_{pre}^{temoin}(\mathtt{p\_sys})\ et\\ s_{now}(\mathtt{p}) = s_{now}^{temoin}(\mathtt{p\_sys})\end{aligned} \tag{4.2}$$

$$\mathtt{no\_impasse} \leftrightarrow \begin{pmatrix}\exists (s_{pre}, s_{now}) \in \Delta,\\ \forall \mathtt{p} \in \mathtt{P}, s_{pre}(\mathtt{p}) = s_{pre}^{temoin}(\mathtt{p\_est})\ et\\ \forall \mathtt{p} \in \mathtt{O}, s_{now}(\mathtt{p}) = s_{now}^{temoin}(\mathtt{p\_est})\end{pmatrix} \tag{4.3}$$

$$\mathtt{no\_impasse} \rightarrow \begin{pmatrix}\exists (s_{pre}, s_{now}) \in \Delta, \forall \mathtt{p} \in \mathtt{P},\\ s_{pre}(\mathtt{p}) = s_{pre}^{temoin}(\mathtt{p\_est})\ et\\ s_{now}(\mathtt{p}) = s_{now}^{temoin}(\mathtt{p\_est})\ et\\ \begin{pmatrix}\nexists s_{best} \in \mathcal{S},\\ \forall \mathtt{o} \in \mathtt{O}, s_{best}(\mathtt{o}) = s_{now}(\mathtt{o}),\ et\\ (s_{pre}, s_{best}) \in \Delta,\ and\\ (s_{pre}, s_{best}) \prec_{\Gamma} (s_{pre}, s_{now})\end{pmatrix}\end{pmatrix} \tag{4.4}$$

En pratique, et d'après la proposition 4, les conditions (4.3) et (4.4) sont satisfaites seulement si les variables de $\mathtt{P^{est}}$ satisfont les préférences quantifiées.

Proposition 5. *Un modèle d'estimation (s_0, Δ, Γ) est sujet à un scénario d'impasse si et seulement si la machine témoin associée admet un chemin dans lequel* `no_impasse` *est faux. Il y a donc correspondance entre le scénario d'impasse et la séquence d'observations associée à la séquence d'états de la machine témoin.*

Démonstration. ($\Rightarrow$) On suppose que le modèle d'estimation est sujet à un scénario d'impasse. D'après la définition 30, il existe une séquence d'observations $(o_0, o_1, \ldots, o_n)$ générée par une séquence d'états $(s_0, s_1, \ldots, s_n)$, telle qu'il existe une séquence d'états estimés $(\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{n-1})$ pour la séquence $(o_0, o_1, \ldots, o_{n-1})$ et il n'existe aucune séquence d'états estimés $(\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_n)$ pour la séquence d'observations complète. On construit alors la séquence d'états de la machine témoin $(t_0, t_1, \ldots, t_n)$ comme suit :

— $\forall i \in [0, n], \forall p \in \mathtt{P}, s_i^{temoin}(\mathtt{p\_sys}) = s_i(\mathtt{p})$,
— $\forall i \in [0, n-1], \forall p \in \mathtt{P}, s_i^{temoin}(\mathtt{p\_est}) = \hat{s}_i(\mathtt{p})$,
— $\forall i \in [0, n-1], s_i^{temoin}(\mathtt{no\_impasse}) = \top$,
— $\forall \mathtt{p} \in \mathtt{O}, s_n^{temoin}(\mathtt{p\_est}) = o_n(\mathtt{p})$,
— $\forall \mathtt{p} \in \mathtt{E}, s_n^{temoin}(\mathtt{p\_est}) = \top$,
— $s_n^{temoin}(\mathtt{no\_impasse}) = \bot$.

On montre que pour tout $i \in [0, n-1]$, (t_i, t_{i+1}) est une transition de la machine témoin : soit i un entier de $[0, n-1]$, on montre que les quatre contraintes de la définition 34 sont satisfaites :

— $\forall \mathtt{o} \in \mathtt{O}$, $\forall p \in \mathtt{P}, s_i(\mathtt{p}) = \hat{s}_i(\mathtt{p})$ and $s_{i+1}(\mathtt{p}) = \hat{s}_{i+1}(\mathtt{p})$: l'équation (4.1) est satisfaite ;
— l'équation (4.2) tient par construction de la machine témoin ;
— pour $i < n-1$, $s_i^{temoin}(\mathtt{no\_impasse}) = \top$ and $(\hat{s}_i, \hat{s}_{i+1})$ est la paire d'états dans Δ satisfaisant la première partie de l'équation (4.3) ;
— pour $i = n-1$, la définition d'un scénario d'impasse implique qu'il n'existe aucune paire d'états satisfaisant Δ : la deuxième partie de l'équation (4.3) est satisfaite ;

— pour $i < n-1$, $s_i^{temoin}(\texttt{no\_impasse}) = \top$ et $(\hat{s}_i, \hat{s}_{i+1})$ est la paire d'état dans Δ satisfaisant l'équation (4.4) ;

($\Leftarrow$) On suppose que la machine témoin admet un chemin dans lequel `no_impasse` est faux. On considère un tel chemin $(s_0^{temoin}, s_1^{temoin}, \ldots, s_n^{temoin})$ dans lequel `no_impasse` est affectée à vrai dans s_i^{temoin} pour $i \in [1, n]$ et faux dans s_n^{temoin}. On construit alors les deux séquences d'états $(s_0, s_1, \ldots, s_n)$ et $(s_0, \hat{s}_1, \ldots, \hat{s}_{n-1})$ comme suit :
— $\forall i \in [0, n], \forall \texttt{p} \in \texttt{P}, s_i(\texttt{p}) = s_i^{temoin}(\texttt{p\_sys})$,
— $\forall i \in [0, n-1], \forall \texttt{p} \in \texttt{P}, \hat{s}_i(\texttt{p}) = s_i^{temoin}(\texttt{p\_est})$
D'après la définition 30, on prouve que ces deux séquences d'états sont cohérentes avec Δ. D'après la définition 34, `no_impasse` est fausse (à l'étape précédente) donc aucun état s_{now} ne permet de satisfaire la partie droite de la condition de l'équation (4.3) pour $s_{pre} = \hat{s}_{n-1}$. Cela signifie qu'il n'existe aucun état candidat (définition 23 page 23). Donc il existe un scénario d'impasse : la séquence d'observation générée par $(s_0, s_1, \ldots, s_n)$.

■

4.2.2 Implémention en Electrum

Dans cette section, on illustre comment la machine témoin des définitions 32, 33 et 34 est encodée en *Electrum*. En *Electrum*, les variables ont des domaines quelconques, et les opérateurs booléens ne sont applicables qu'aux prédicats. Nous utilisons la librairie `bool` qui introduit le domaine `Bool` booléen ainsi que les prédicats `isTrue` et `isFalse` qui testent la valeur d'une variable.

On présente ici l'encodage du système simple décrit dans les exemples 2 et 4 du chapitre précédent. Afin de modéliser les états de la machine du système, on déclare un prédicat correspondant exactement au modèle Δ. Ensuite, on introduit une contrainte (un `fact` en *Electrum*) spécifiant que le système respecte ce prédicat à chaque étape discrète (`always` en *Electrum*), comme illustré dans la figure 4.2. Une contrainte similaire est déclarée pour la machine à états de l'estimateur, mais sa satisfaction n'est pas imposée pour justement pouvoir identifier les scénarios d'impasse. Enfin, on synchronise les variables de `O` de la machine à état du système observé avec celles de la machine à états de l'estimateur, comme décrit dans la condition (4.1) de la définition 34.

Afin de représenter l'aspect temporel du formalisme d'estimation, on utilise l'opérateur « ' » d'*Electrum* qui décrit une variable à l'état suivant. Ce mécanisme illustré dans la figure 4.3 reproduit le fonctionnement de la fonction bijective *pre* du modèle d'estimation à état unique (la valeur de la variable `pre_p` à l'étape suivante est égale à celle de `p` dans l'état courant).

Electrum supporte l'utilisation des opérateurs de l'algèbre relationnelle, on utilise donc ceux-ci pour décrire les préférences sous leur forme quantifiée. Il est possible pour chaque préférence d'écrire un prédicat qui indique si une préférence γ_i est applicable : si les deux valeurs booléennes sont possibles pour $\texttt{e}_\texttt{i}$ (opérateur `all`), et si au moins une valeur booléenne est possible pour les autres variables $\texttt{e}_\texttt{j}, i < j \leq n$ (opérateur `some`) en restant cohérent avec Δ et l'observation. Cela correspond à la partie gauche de la forme quantifiée d'une préférence (voir définition 31 page 33). La figure 4.4 illustre la définition des prédicats qui encodent ces conditions avec les opérateurs d'*Electrum*. L'ordre dans lequel les préférences sont appliquées dans Γ, est complètement induit par la quantification dans les formes quantifiées des préférences.

4.2.3 Expérimentations pour l'approche Model-checking

Les expérimentations proposées implémentent des versions étoffées du modèle d'estimation présentés dans les exemples 2 et 4 du chapitre précédent, dans lequel un ou plusieurs robots autonomes évoluent sur une grille de taille paramétrable. Les robots peuvent être

```
pred delta[move,engine,fmove,fengine,pre_fengine : Bool] {
  (isTrue[move] <=> ( isTrue[engine] and
                      !isTrue[fengine] and
                      !isTrue[fmove]))
    and
  (isTrue[pre_fengine] => isTrue[fengine])
}
var one sig RealSystem {
  var move : Bool,
  var engine : Bool,
  var fmove : Bool,
  var fengine : Bool,
  var pre_fmove: Bool,
  var pre_fengine : Bool,
}
fact always_delta_RealSystem {
  always {
    delta[RealSystem.move,
          RealSystem.engine,
          RealSystem.fmove,
          RealSystem.fengine,
          RealSystem.pre_fengine]}
}
fact synch_obs {
  always {
    Estimator.move = RealSystem.move
    Estimator.engine = RealSystem.engine}
}
```

FIGURE 4.2 – Le système observé et la synchronisation des observations en *Electrum*.

```
fact next_Estimator {
  always {
    Estimator.pre_fmove' = Estimator.fmove
    Estimator.pre_fengine' = Estimator.fengine }
}
```

FIGURE 4.3 – Lien entre les variable de P et $\mathtt{P}_{\mathtt{pre}}$ en *Electrum*, l'opérateur « ' » fait référence à la variable ciblée dans l'état suivant, on retrouve le même prédicat pour le système.

touchés par une faute intermittente ou par une faute permanente. On introduit des variables représentant les états des cellules de la grille (normal, dangereuse ou difficile). Les règles de Δ représentent à la fois l'environnement (la grille et le fonctionnement des différentes cellules) et le comportement des robots chacun sujet à une faute permanente et une faute intermittente (similaire à l'exemple 4). On suppose que l'estimateur ne connaît pas à l'avance l'état des cellules de la grilles. On retrouve donc dans Γ une préférence pour chaque case de la grille, ainsi que deux préférences par robot (une pour chaque type de faute).

Exemple 10 (Modélisation d'une mission autonome). *Les expérimentations ont été effectuées sur une version étoffée des modèles présentés dans les exemples 2 et 4. On considère un ou plusieurs robots autonomes se déplaçant sur une grille en deux dimensions de taille variable. Chaque robot a pour objectif d'atteindre une case cible de la grille préalablement définie. La taille de la grille et le nombre de robots sont paramétrables afin de produire des expérimentations sur des modèles de différentes tailles. Pour chaque case de la grille de coordonnées (i,j) et un robot numéroté n on retrouve les variables suivantes :*

- `bot_n_permfail` *: vraie si le robot n est affecté par une faute permanente*
- `bot_n_tempfail` *: vraie si le robot n est affecté par une faute intermittente.*
- `bot_n_canmove` *: vraie si le robot est capable de changer de case.*
- `bot_n_moving` *: vraie si le robot change de case à l'étape courante*
- `bot_n_at_ij` *: vraie si le robot se trouve en (i,j) sur la grille*
- `bot_n_targetij` *: vraie si la case cible du robot est (i,j)*
- `bot_n_attarget` *: vraie si le robot à atteint sa cible.*

```
pred preference_fmove_possible{
  all fmove : Bool |some fengine : Bool |
    delta[Estimator.move,
          Estimator.engine,
          fmove,
          fengine,
          Estimator.pre_fengine]
}

pred preference_fengine_possible{
  all fengine : Bool |
    delta[Estimator.move,
          Estimator.engine,
          Estimator.fmove,
          fengine,
          Estimator.switch,
          Estimator.pre_fengine]
}

pred preference_fmove_applied{
  isTrue[Estimator.fmove] <=> isTrue[Estimator.preF_fmove]
}
pred preference_fengine_applied{
  isTrue[Estimator.fengine] <=> isFalse[engine]
}

fact preferences_applied {
  always {
    preference_fmove_possible implies preference_fmove_applied
    preference_fengine_possible implies preference_fengine_applied }
}
```

FIGURE 4.4 – Implémentation des préférences sous forme quantifiée en *Electrum*

```
assert deadlock_free_Estimator{
always {
    delta[Estimator.move,
          Estimator.engine,
          Estimator.fmove,
          Estimator.fengine,
          Estimator.pre_fengine]}
}
check deadlock_free_Estimator
```

FIGURE 4.5 – Vérification de la propriété en *Electrum*, cette assertion correspond à la valeur de `no_impasse` qui est ici abstraite.

- `bot_n_atdangerous` *: vraie si le robot n se trouve sur une case dangereuse.*
- `bot_n_atdifficult` *: vraie si le robot n se trouve sur une case difficile*
- `c_ij_dangerous` *: vraie si la case (i, j) est dangereuse ; un robot ne peut pas sortir d'une case dangereuse, il reste bloqué.*
- `c_ij_difficult` *: vraie si la case est difficile à traverser, un robot restera bloqué pendant un certain temps mais pourra en sortir.*

Les variables observées sont les variables de type `bot_n_at_ij` `bot_n_moving`, `bot_n_targetij`. *On peut donc observer à tout instant la position du robot sur la grille, si celui-ci change de case et la case cible que celui-ci cherche à atteindre. Toutes les autres variables sont estimées et sont donc chacune la cible d'une préférence.*

On construit l'estimateur (s_0, Δ, Γ) avec les contraintes suivantes, en utilisant la logique PTLTL *(définition 16 page 17) :*

Pour chaque case de coordonnées (i, j) de la grille et chaque robot ayant pour numéro n, on retrouve dans le modèle comportementale les règles suivantes :

$$\Delta = \left\{ \begin{array}{cl} \boldsymbol{Y}(\texttt{bot\_n\_permfail}) \rightarrow \texttt{bot\_n\_permfail} & (\delta_1) \\ \boldsymbol{Y}(\texttt{bot\_n\_at\_ij}) \wedge \texttt{bot\_n\_moving} \rightarrow \neg(\texttt{bot\_n\_at\_ij}) & (\delta_2) \\ \neg\texttt{bot\_n\_canmove} \leftrightarrow \texttt{bot\_n\_tempfail} \vee \texttt{bot\_n\_permfail} & \\ \vee\texttt{bot\_n\_atdangerous} \vee \texttt{bot\_n\_atdifficult} & (\delta_3) \\ \texttt{bot\_n\_moving} \rightarrow \texttt{bot\_n\_canmove} \wedge \neg\texttt{bot\_n\_attarget} & (\delta_4) \\ \bigwedge_{i,j}(\texttt{c\_ij\_dangerous} \leftrightarrow \neg\texttt{c\_ij\_difficult}) & (\delta_5) \\ \texttt{bot\_n\_atdangerous} \leftrightarrow \bigvee_{i,j}(\texttt{c\_ij\_dangerous}) \wedge (\texttt{bot\_n\_at\_ij}) & (\delta_6) \\ \texttt{bot\_n\_atdifficult} \leftrightarrow \bigvee_{i,j}(\texttt{c\_ij\_difficult}) \wedge (\texttt{bot\_n\_at\_ij}) & (\delta_7) \end{array} \right\}$$

Les règles de Δ *décrivent le comportement du système illustré en figure* 4.6 *comme suit.* δ_1 *indique que si la faute permanente affectait le robot à l'étape précédente, alors celle-ci l'affecte à l'état courant (comme dans l'exemple 2). Le déplacement d'une case à l'autre est assuré par la contrainte* δ_2 *; si un robot se déplace alors il change de case sur la grille.* δ_3 *indique qu'un robot peut se déplacer si et seulement si aucune faute ne l'affecte et que la case sur laquelle il se trouve n'est ni dangereuse, ni difficile.* δ_4 *indique que si le robot se déplace alors c'est qu'il peut se déplacer et qu'il n'a pas atteint sa cible. Enfin, une case de la grille ne peut pas être à la fois dangereuse et difficile (*δ_5*). (*δ_6*) et (*δ_7*) indique si un robot se trouve sur une case dangereuse ou difficile. Afin de limiter la taille du modèle, on rajoute aussi des contraintes interdisant à un robot de se trouver sur deux cases à la fois ou de cibler plusieurs cases. Toutes ces contraintes ne sont pas décrites explicitement dans le modèle comportemental ci-dessus.*

On construit Γ*, pour chaque robot numéroté* n *et chaque case* (i, j) *de la manière suivante :*

$$\Gamma = \left(\begin{array}{ll} \bot : c\_ij\_dangerous \prec \overline{c\_ij\_dangerous} & (\gamma_1) \\ \top : c\_ij\_difficult \prec \overline{c\_ij\_difficult} & (\gamma_2) \\ \top : bot\_n\_permfail \prec \overline{bot\_n\_permfail} & (\gamma_3) \\ \bot : bot\_n\_tempfail \prec \overline{bot\_n\_tempfail} & (\gamma_4) \\ \top : bot\_n\_canmove \prec \overline{bot\_n\_canmove} & (\gamma_5) \end{array} \right)$$

Γ *décrit la stratégie d'estimation suivante : on préfère estimer les états dans lesquels la case n'est pas dangereuse (*γ_1*). A l'inverse on préfère les états où la case de la grille est difficile à traverser (*γ_2*). On préfère estimer qu'une faute permanente (*γ_3*) affecte le ou les robots, alors qu'on préfère estimer l'absence des fautes intermittentes (*γ_4*). Enfin, on préfère estimer les états dans lesquels les robots ont la capacité de se déplacer (*γ_5*).*

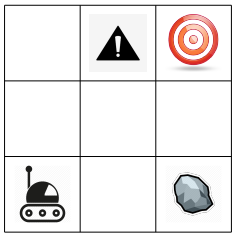


FIGURE 4.6 – Modèle d'une mission autonome utilisé pour les expérimentations par l'approche *Electrum*. Un où plusieurs robots cherchent à atteindre une case cible de la grille. Les cases de la grilles peuvent être normal, dangereuse ou difficile

Afin de créer des exemples avec et sans impasses, on modifie la préférence (γ_4). Lorsque la condition de γ_4 est vraie (jamais), on préfère les états où la faute permanente est présente, cette stratégie d'estimation produit des impasses. Inversement, si on fixe la condition de γ_4

à faux, préférant ainsi les états où la faute permanente est absente, on obtient des modèles dans lesquels aucune impasse n'est trouvée par *Electrum*. La figure 4.7 présente les résultats expérimentaux. Chaque ligne correspond à deux versions d'un modèle de taille donnée avec et sans impasse où la (ou les) condition(s) de la (des) préférence(s) de type γ_4 est (sont) modifiée(s).

ID	robots	‖grille‖	‖P‖	‖Δ‖	‖Γ‖	CNF vars	CNF clauses	avec	sans
1	1	1 x 2	13	17	7	1083	1586	0,1s	0,8s
2	1	2 x 2	21	27	11	3400	3925	0,5s	4,8s
3	1	2 x 3	29	37	15	13757	10593	7,3s	52,9s
4	2	2 x 2	34	50	14	12631	13561	6,7s	60,5s
5	2	2 x 3	46	68	18	65 030	47 669	144,6s	1001,4s

FIGURE 4.7 – Vérification des modèles en *Electrum*. Les colonnes ‖P‖, ‖Δ‖ et ‖Γ‖ indiquent respectivement le nombre de variables de P, le nombre de règles de Δ et le nombre de préférences dans Γ. Les colonnes 7 et 8 indiquent respectivement le nombre de variables et de clauses envoyées au solver Sat4j par *Electrum*. Les deux dernières colonnes indiquent le temps de vérification pour une version du modèle sujette à un scénario d'impasse et une autre version pour laquelle un tel scénario n'existe pas.

Les modèles sont écrits en Scala et automatiquement traduits vers la syntaxe *Electrum*. *Electrum* est ensuite paramétré afin que celui-ci utilise le solver Sat4j. Les tests ont été réalisés sur un processeur intel core i5-7600 cadencé à 3.5 Ghz. Electrum est paramétré afin d'explorer des séquences de 15 états maximum.

4.3 Approche SAT itérative

Electrum est très expressif et permet donc de représenter Γ dans un langage de model-checking. Cependant les expérimentations (figure 4.7) montrent non seulement des temps de calcul assez long pour des systèmes complexes, mais aussi une transformation vers une CNF peu efficace. En effet, on ne contrôle pas la manière dont *Electrum* encode notre problème lorsqu'il le traduit pour un solver SAT et de nombreuses variables et clauses sont générées de manière probablement sous-optimale. On propose donc ici une deuxième approche utilisant directement les solvers SAT. Pour cela, on définit dans cette section la notion de séquence d'états dépliée. La recherche d'impasse est effectuée par l'intermédiaire d'un solver SAT de manière itérative et les préférences sont implémentées grâce à un solver MAX-SAT.

4.3.1 Encodage du problème en SAT

On développe ici une approche en deux étapes basée sur les solvers SAT. Comme avec l'approche par model-checking, cette approche est bornée dans la taille des scénarios d'impasse, c'est-à-dire qu'on pourra prouver la présence ou l'absence de scénario d'impasse d'une taille donnée.

Soit k la taille maximale des scénario d'impasse que l'on cherche à détecter. On déplie le modèle comportemental sur cet horizon en créant de nouvelles variables représentant les variables de Δ pour chaque pas de temps de 0 à k. Plus précisément pour chaque variable $\mathtt{p}$ de P et pour chaque étape t dans $[0, k]$, on crée une variable $\mathtt{p}^t$ qui représente la variable $\mathtt{p}$ à l'étape t. Pour une formule f portant sur les variables de P et $\mathtt{P}_{\mathtt{pre}}$ et une étape t comprise dans $[1, k]$, on définit l'instanciation de la formule f à t, noté f^t, en remplaçant syntaxiquement dans f toutes les variables $\mathtt{p}$ de P par $\mathtt{p}^t$ et chaque variable $pre(\mathtt{p})$ de $\mathtt{P}_{\mathtt{pre}}$ par $\mathtt{p}^{t-1}$. A l'étape 0, chaque variable $\mathtt{p}^0$ de P est remplacée par sa valeur dans s_0. On instancie

la formule Δ pour chaque étape $t > 0$ dans la formule Δ^t. Dans cette configuration, on duplique les variables du système ce qui permet de définir les séquences d'états dépliées.

Proposition 6 (Séquence d'états dépliée). *Soit u une affectation sur toutes les variables dupliquées $\{\mathtt{p}^t | t \in [0,k]\}$. u représente la séquence d'états $(s_0, s_1, \ldots, s_k)$ tel que $u(\mathtt{p}^t) = s_t(\mathtt{p})$ pour $\mathtt{p} \in \mathtt{P}$ et $t \in [0,k]$.*

u satisfait $\Delta^1 \wedge \ldots \wedge \Delta^k$ si et seulement si u représente une séquence d'état cohérente (définition 21 page 23).

Soit v une affectation portant sur toutes les variables observables dupliquées $\{\mathtt{o}^t | t \in [0,k], \mathtt{o} \in \mathtt{O}\}$. v représente la séquence d'observation $(o_0, \ldots, o_k)$ tel que $v(\mathtt{o}^t) = o_t(\mathtt{o})$ pour $\mathtt{o} \in \mathtt{O}$ et $t \in [0,k]$.

v est cohérente avec $\Delta^1 \wedge \ldots \wedge \Delta^k$ si et seulement si v représente une séquence d'observation cohérente (définition 22 page 23).

Démonstration. Pour chaque $t \in [1,k]$, par construction Δ^t contient uniquement des variables de $\mathtt{p}^{t-1}$ et $\mathtt{p}^t$. L'affectation u restreinte à ces variables est l'affectation σ_{s_{t-1},s_t}. u satisfait Δ^t si et seulement σ_{s_{t-1},s_t} satisfait Δ puisque ces deux formules réfèrent aux mêmes variables. ■

On déplie aussi les préférences de Γ sur l'horizon temporel k : pour chaque préférence $\gamma_i : \mathtt{cond_i} : e_i \prec \overline{e_i}$ et chaque étape $t > 0$, on définit la préférence $\gamma_i^t : \mathtt{cond_i}^t : e_i^t \prec \overline{e_i^t}$ où $\mathtt{cond_i}^t$ est la formule $\mathtt{cond_i}$ instanciée à l'étape t. e_i^t et $\overline{e_i^t}$ sont les affectations dans lesquelles les variables $\mathtt{e_i}^t$ sont respectivement affectées à vrai et à faux. Ces séquences et préférences dépliées sont utilisées dans les algorithmes présentés ensuite.

L'algorithme repose sur l'utilisation de la fonction *estimSeq* définie dans le chapitre 3 (définition 29 page 28). On l'intègre ici dans une fonction qui à partir d'une séquence d'observations retourne la séquence d'états estimés ou une erreur en cas d'impasse. On intègre cette implémentation dans une fonction *Impasse* qui appelle la fonction *estimSeq* et indique si l'estimateur peut ou non produire une séquence d'états estimés pour un modèle d'estimation (s_0, Δ, Γ) et une séquence d'observations *seqObs*. Nous avons vu dans le chapitre 3 comment cette fonction peut être implémentée grâce à un solveur MaxSAT.

L'algorithme cherche un scénario d'impasse de taille maximale k. Son implémentation repose sur l'énumération de modèle en SAT [Morgado and Marques-Silva, 2005] afin de produire toutes les séquences d'observations cohérentes avec Δ et valide ou non chaque séquence d'observations avec la fonction *Impasse*. Pour chaque étape t, on définit la formule *toCheck* comme la conjonction de toutes les formules Δ^i de $i \in [1,t]$, c'est-à-dire le modèle comportemental dans sa version dépliée sur l'horizon du t courant.

On énumère tous les modèles de la formule *toCheck* projetés sur les variables observables, c'est-à-dire les modèles dans lesquels seules les variables observables sont considérées. De cette manière on génère des séquences d'observations cohérentes. La variable *obsSeq* est donc peuplée itérativement avec des séquences d'observations de taille t.

Pour chaque séquence d'observations, si celle-ci est une impasse , alors on retourne celle-ci. Sinon, la recherche continue jusqu'à ce qu'aucun scénario d'impasse de taille inférieure ou égale à k ne soit trouvé : cela signifie qu'il existe une séquence d'estimations pour chaque séquence d'observations.

Cet algorithme peut être étendu afin d'énumérer tous les scénarios d'impasse de taille inférieur à k présents dans le système (algorithme 1). Pour cela, on introduit un ensemble *impasses*, initialement vide, qui stocke toutes les séquences d'observations se révélant être des impasses (ligne 7). Dès lors qu'une impasse est trouvée, celle-ci est ajouté à *impasses* (ligne 13) et sa négation est ajoutée à la formule *toCheck* (ligne 9). L'ajout de la négation de toutes les impasses permet non seulement de ne pas détecter une impasse déjà trouvée mais aussi de ne pas explorer sa continuation. L'algorithme retourne ensuite tous les scénarios d'impasses trouvés (l'ensemble *impasses*) (ligne 16). Si l'ensemble retourné est vide, on peut certifier qu'il n'existe pas de scénario d'impasse pour l'horizon borné k.

Algorithm 1 *TrouverImpasses*(s_0, Δ, Γ, k) : retourne toutes les impasses de taille inférieure ou égale à k

Entrees
  (s_0, Δ, Γ) : un modèle d'estimation
  k : horizon de vérification
Sorties
  impasses : l'ensemble des séquences d'observations étant des impasses
function TROUVERIMPASSES(s_0, Δ, Γ, k)
  $impasses \leftarrow \{\}$
  for $t \leftarrow 1 : k$ **do**
    $toCheck \leftarrow \bigwedge_{i=1}^{t} \Delta^i \wedge \bigwedge_{imp \in impasses} \neg imp$
    $obsSeqs \leftarrow subModels(toCheck, \{\mathtt{o}^i | i \in [1,t], \mathtt{o} \in \mathtt{O}\})$
    for $seqObs \leftarrow obsSeqs$ **do**
      if $Impasse(s_0, \Delta, \Gamma, seqObs)$ **then** $imp \leftarrow seqObs$
        $impasses \leftarrow impasses \cup \{imp\}$
    end for
  end for
  return *impasses*

4.3.2 Expérimentations pour l'approche SAT

On considère dans un premier temps une architecture simple illustrée sur la figure 4.8 dotée de trois fonctions : une fonction de déplacement, une fonction de communication et une fonction d'alimentation Les alarmes et les perturbations environnementales peuvent provoquer des fautes dans le système sur les fonctions de mouvement et/ou de communication respectivement représentées par les variables $\mathtt{fenv}_{\mathtt{move}}$ et $\mathtt{fenv}_{\mathtt{com}}$. Un défaut de l'alimentation est représentée par la faute $\mathtt{low}_{\mathtt{pow}}$. Le but est d'estimer si chacune des fonctions est disponible pour mener à bien la mission autonome. Plutôt que d'estimer l'état physique de chaque fonction, on estime des niveaux de confiance. Les variables $\mathtt{t}_{\mathtt{move}}$, $\mathtt{t}_{\mathtt{com}}$, et $\mathtt{t}_{\mathtt{pow}}$ représentent une totale confiance pour chacune de leur fonction respective. On suppose que la valeur de ces variables est ensuite transmise à un opérateur humain ou à un module de planification. On écrit les contraintes de Δ et Γ en utilisant *PTLTL*. On introduit ici une version de l'opérateur **H**(Historically) sur un horizon borné. Pour un nombre k d'étapes discrètes, l'opérateur $\mathbf{H}_k$ est défini comme une k conjonction d'opérateur **Y**(Yesterday). Par exemple, pour une variable $\mathtt{v}$ et avec $k = 3$ on a :

$$\mathbf{H}_k \leftrightarrow v \wedge \mathbf{Y}(v) \wedge \mathbf{Y}(\mathbf{Y}(v)) \wedge \mathbf{Y}(\mathbf{Y}(\mathbf{Y}(v)))$$

Les règles de Δ illustrées en figure 4.8 représentent le comportement suivant. On ne peut avoir confiance en les fonctions de mouvement et de communication que si on a confiance en la fonction d'alimentation (δ_1). Une perturbation de la fonction de mouvement ou une baisse de tension provoquent une alarme liée au mouvement (δ_2), de même pour la fonction de communication (δ_3). Une confiance perdue dans la fonction d'alimentation ne peut être retrouvée (δ_4). L'alarme portant sur la communication influe directement la confiance en celle-ci (δ_5). Les règles (δ_1) et (δ_4) ne représentent pas nécessairement le comportement du système, elle jouent plutôt le rôle de spécification propre à la tâche du diagnostic ; on ne calculera jamais de séquences d'états estimés allant à l'encontre de ces règles.

Afin de proposer des expérimentations de taille plus importante, on implémente aussi une architecture complète illustrée en figure 4.9. On considère deux fonctions additionnelles : la localisation et la navigation. La confiance de chacune des fonctions est maintenant évaluée sur des ensembles de variables $\{\mathtt{ok}, \mathtt{deg}, \mathtt{ko}\}$, représentant respectivement une confiance totale, dégradée ou nulle pour chaque fonction de l'architecture. Les causes des fautes sont toujours modélisées telles qu'une baisse de tension, une perturbation environnementale ou

$$\Delta = \left\{ \begin{array}{cc} (\mathtt{t}_{\mathtt{move}} \vee \mathtt{t}_{\mathtt{com}}) \rightarrow \mathtt{t}_{\mathtt{pow}} & (\delta_1) \\ (\mathtt{fenv}_{\mathtt{move}} \vee \mathtt{low}_{\mathtt{pow}}) \rightarrow \mathtt{al}_{\mathtt{move}} & (\delta_2) \\ (\mathtt{fenv}_{\mathtt{com}} \vee \mathtt{low}_{\mathtt{pow}}) \rightarrow \mathtt{al}_{\mathtt{com}} & (\delta_3) \\ \neg \mathbf{Y}(\mathtt{t}_{\mathtt{pow}}) \rightarrow \neg \mathtt{t}_{\mathtt{pow}} & (\delta_4) \\ \mathtt{t}_{\mathtt{com}} \leftrightarrow \neg \mathtt{al}_{\mathtt{com}} & (\delta_5) \end{array} \right\}$$

$$\Gamma = \left(\begin{array}{lcrl} \neg \mathbf{Y}(\mathtt{al}_{\mathtt{move}}) \wedge \mathtt{al}_{\mathtt{move}} \wedge \neg \mathbf{Y}(\mathtt{al}_{\mathtt{com}}) \wedge \mathtt{al}_{\mathtt{com}} & : & low_{pow} \prec \overline{low_{pow}} & (\gamma_1) \\ \mathbf{O}(\mathbf{H}_{\kappa}(\mathtt{low}_{\mathtt{pow}})) & : & \overline{t_{pow}} \prec t_{pow} & (\gamma_2) \\ \mathbf{H_2}(\mathtt{al}_{\mathtt{move}}) \wedge \neg \mathtt{low}_{\mathtt{pow}} & : & fenv_{move} \prec \overline{fenv_{move}} & (\gamma_3) \\ \mathbf{H_4}(\mathtt{al}_{\mathtt{move}}) & : & t_{move} \prec \overline{t_{move}} & (\gamma_4) \\ \mathtt{al}_{\mathtt{com}} \wedge \neg \mathtt{low}_{\mathtt{pow}} & : & fenv_{com} \prec \overline{fenv_{com}} & (\gamma_5) \\ \top & : & t_{com} \prec \overline{t_{com}} & (\gamma_5) \end{array} \right)$$

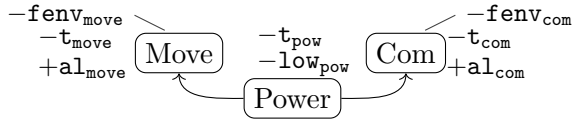


FIGURE 4.8 – Δ, Γ et le schéma de l'architecture simple. Les variables labélisées avec + sont observables, celle avec − sont estimés. Les arcs représentent les dépendances fonctionnelles.

encore la surchauffe d'un moteur ($\mathtt{ovh}_{\mathtt{move}}$). Les déviations de trajectoire sont détectées dans le composant de navigation ($\mathtt{dev}_{\mathtt{nav}}$). La communication peut être rapide, lente, ou encore perdue ($\{\mathtt{fast}, \mathtt{slow}, \mathtt{none}\}$). Enfin la fonction de localisation prend compte des perturbations environnementales ($\mathtt{fenv}_{\mathtt{loc}}$), et peut déclencher une alarme ($\mathtt{al}_{\mathtt{loc}}$) en cas de faible signal GPS ou une alarme ($\mathtt{deadgps}_{\mathtt{loc}}$) si le GPS tombe en panne. Le modèle de l'architecture complète est composé de 27 variables dont 20 sont estimées. On retrouve 48 formules dans Δ et 20 préférences dans Γ.

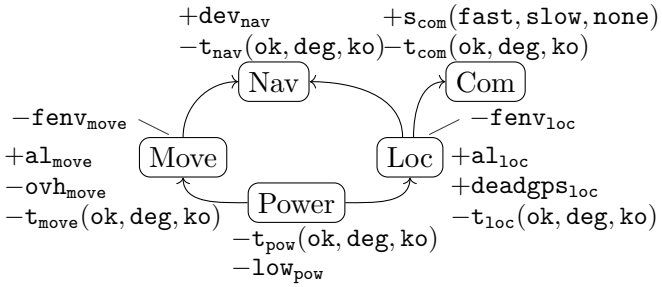


FIGURE 4.9 – Illustration de l'architecture fonctionnelle complète.

Tout comme l'approche par model-checking, les modèles sont écrits en Scala. Il est possible d'intégrer l'utilisation de Sat4j dans l'architecture Scala. Les tests sont là encore réalisés sur un processeur intel core i5-7600 cadencé à 3.5 Ghz.

4.4 Résultats

En ce qui concerne l'approche par model-checking, la première conclusion que l'on peut tirer est que la vérification est plus rapide dans les cas où il y a présence de scénarios d'impasse dans le modèle. Cette différence notable en terme de temps de calcul est directement due au fait que la recherche prend fin dès lors qu'un scénario d'impasse est trouvé et un contre exemple est renvoyé (la séquence d'observations provoquant une impasse). A l'inverse, lorsque le modèle n'est pas sujet à un scénario d'impasse sur l'horizon étudié, toutes les exécutions possibles du système, c'est-à-dire toutes les séquences d'états cohérentes avec

Instance	Time (s)	nPaths
small_3	0	16
small_4	0	58
small_5	0	222
small_6	0	870
small_7	4	3446
small_8	20	13718
small_9	171	54743
small_10	1694	218718
large_3	1	68
large_4	2	1649
large_5	108	39488
large_6	+3600	–

TABLE 4.1 – Résultats des expérimentations pour la recherche d'un scénario d'impasse. Les colonnes sont définies comme suit : nom de l'instance (*Instance*) ; temps de calcul en seconde (*Time*) ; nombre de séquences d'observations testées (*nPaths*).

le modèle comportemental, doivent être explorées.

On constate dans les deux approches que les temps de calculs pour des modèles avec et sans impasses sont courts pour des modèles de petites tailles. Cependant, dès lors qu'on enrichit le nombre de variables, et donc qu'on implémente d'avantage de préférences, on constate que les temps de calculs croissent de manière exponentielle. Il y a même certains modèles testés pour lesquelles le solver Sat4j ne dispose pas d'assez de mémoire pour effectuer la vérification. Si on s'intéresse à la comparaison des deux approches, bien que celles-ci aient été réalisées sur des modèles différents mais de taille similaire, on constate que l'approche par solver SAT s'avère être plus efficace. Cela s'explique par le fait qu' *Electrum* crée un nombre important de variables intermédiaires lorsque celui-ci traduit le problème vers un problème SAT. On le constate en effet dans la septième colonne de la figure 4.7 : un modèle d'une cinquantaine de variables propositionnelles peut générer jusqu'à 65 000 variables pour le solver.

De plus, l'approche par *Electrum* permet de trouver un unique scénario d'impasse tandis que l'approche par solver permet d'obtenir tous les scénarios d'impasse pour un modèle donné. Il est cependant possible d'effectuer plusieurs vérifications de manière itérative avec *Electrum* afin de générer tous les scénarios d'impasses mais cela reste fastidieux. Il est important de noter que même si les temps de calculs sont assez longs pour des modèles de grande taille, cela n'est pas rédhibitoire. En effet, le but de ces travaux est de proposer des méthodes de vérification hors-ligne pour les modèles d'estimation avant leur déploiement, on s'abstrait donc des contraintes temps réel qui restent primordiale dans le domaine.

Avec les deux approches, il est donc possible de détecter (sur une fenêtre bornée) si un modèle d'estimation préalablement définie est sujet à des scénarios d'impasse. Cependant il reste difficile à ce stade d'identifier les composants du modèle responsable d'un scénario d'impasse. Ce sujet est traité dans le chapitre suivant sous l'hypothèse que la stratégie d'estimation (les préférences) provoque le scénario d'impasse par ses choix d'estimation.

Instance	k	nDLs	Time (s)	nPaths
small_3	4	8	0	77
	6	84	2	1005
	8	984	19	13613
	10	12884	1905	187949
small_4	4	2	0	85
	6	28	1	1301
	8	356	31	19981
	10	5122	4473	308661
small_5	6	8	1	1357
	8	100	33	21477
small_6	6	2	1	1365
	8	28	37	21781
small_7	8	8	38	21837
small_8	8	2	38	21845
small_noDL	4	0	0	85
	6	0	1	1365
	8	0	31	21845
large_3	4	672	30	16397
large_4	4	128	30	17293
large_noDL	4	0	43	17293

TABLE 4.2 – Résultats des expérimentations pour la recherche de tous les scénarios d'impasse de taille inférieure ou égale à k. Les colonnes sont définies comme suit : nom de l'instance (*Instance*) ; taille maximale de l'impasse (k), nombre d'impasse détectées (*nDLs*), temps de calcul en seconde (*Time*).

Chapitre 5

Méta-Diagnostic des préférences

Dans le chapitre précédent, on a vu qu'il était possible de vérifier si un estimateur risque de rencontrer une ou plusieurs impasses. On souhaite désormais fournir à l'utilisateur plus d'informations lorsqu'il y a présence d'un tel scénario et on cherche notamment à cibler les préférences de l'estimateur à état unique qui seraient responsables de l'impasse. Ce chapitre présente un formalisme de méta-diagnostic d'estimateurs à état unique, inspiré de la théorie générale du méta-diagnostic [Belard et al., 2011] qui reprend les concept de base du diagnostic à base de modèle [Reiter, 1987] et les applique aux modèles de diagnostic eux-mêmes.

On suppose lors de ces travaux que la dynamique du système étudié est conforme à ses spécifications, on ne remet donc pas en cause la relation de transition du système surveillé, ni son état initial. On s'intéresse plutôt aux choix liés à l'estimation, issus des préférences conditionnelles, qui seraient responsables de l'apparition du scénario d'impasse.

La méthode du méta-diagnostic nous permet d'identifier les différents sous-ensembles de préférences de la stratégie d'estimation à état unique afin que le concepteur sache quelles préférences modifier afin d'éviter les scénarios d'impasse. Pour cela, une formalisation du méta-diagnostic des préférences est décrite dans ce chapitre et deux implémentations sont proposées : la première repose sur le model-checker *Electrum*, la seconde sur un solver SAT. Des expérimentations sont ensuite menées sur les jeux de données du chapitre précédent dont on récupère un scénario d'impasse.

5.1 Modélisation du méta-diagnostic des préférences

Dans cette approche de méta-diagnostic, on considère un modèle de préférence inapproprié menant à un scénario d'impasse dans l'estimation. On s'intéresse à l'élimination du scénario d'impasse en désactivant certaines préférences de Γ ou plus précisément en les relaxant. On définit le problème de méta-diagnostic comme la recherche d'un sous-ensemble de préférences, qui lorsque celle-ci sont relaxées, permettent de rétablir la cohérence dans l'estimation. On applique alors une approche de diagnostic à base de cohérence sur le modèle de préférences. Plus précisément, étant donné un scénario d'impasse, on cherche à savoir si l'estimateur accepterait la séquence d'observation menant à une impasse si les préférences en cause sont relaxées. On peut alors indiquer à l'utilisateur quelles préférences modifier afin d'éliminer le scénario d'impasse et de fournir une estimation. On s'intéresse ici à l'identification des préférences responsables d'un scénario d'impasse, la manière dont il faudrait les modifier n'est pas étudiée mais reste une perspective future. On décrit d'abord la sémantique associée à la relaxation de préférences issues de Γ, ensuite on fournit une définition pour le diagnostic à base de cohérence pour un ensemble de préférences.

5.1.1 Modèle de préférences relaxées

Pour une variable e de E, une *préférence relaxée* $\gamma^{\nsim}$ indique qu'aucune valuation e ne peut être préférée quelque soit le contexte.

Définition 35 (Préférence relaxée). *Une préférence relaxée pour une variable* e *de* E *(la cible de la préférence relaxée), notée* $\gamma^{\nsim}$*, est de la forme* $\langle e \nsim \overline{e} \rangle$*.*

On définit le concept de préférence générale comme étant soit une préférence conditionnelle, soit une préférence relaxée :

Définition 36 (Préférence). *Une préférence de* $\mathtt{e} \in \mathtt{E}$*, notée* φ*, est soit une préférence conditionnelle de la forme* $\langle \mathtt{cond} : e \prec \overline{e} \rangle$ *soit une préférence relaxée de la forme* $\langle e \nsim \overline{e} \rangle$*.*

Les préférences généralisent le concept de préférence conditionnelle et induisent un ordre partiel sur les transitions de Δ différent de l'ordre partiel défini en définition 26 page 25. Pour chaque préférence φ ciblant une variable e de E, l'ordre partiel associé est défini comme suit : $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) est strictement préféré à (s_{pre}, s') d'après φ (noté $(s_{pre}, s) \prec_{\varphi} (s_{pre}, s')$) si et seulement si φ est une préférence conditionnelle γ et $(s_{pre}, s) \prec_{\gamma} (s_{pre}, s')$.

La relation d'équivalence associée est elle aussi différente. Soit φ une préférence ciblant une variable e de E, $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) et (s_{pre}, s') sont équivalentes d'après φ (noté $(s_{pre}, s) \approx_{\varphi} (s_{pre}, s')$) si et seulement si φ est une préférence conditionnelle γ et $(s_{pre}, s) \approx_{\gamma} (s_{pre}, s')$, ou si φ est une préférence relaxée et $s(\mathtt{e}) = s'(\mathtt{e})$.

Intuitivement, remplacer une préférence conditionnelle par une préférence relaxée retire des paires de transitions de la relation d'ordre partiel et de la relation d'équivalence induite par le modèle relaxé. Cela créé des paires de transitions incomparables. On relaxe ainsi le concept d'estimation unique à des fins d'analyse en permettant possiblement plusieurs états estimés possibles. On définit grâce aux préférences relaxées les modèles de préférences relaxées.

Définition 37 (Modèle de préférences relaxées). *Étant donné un modèle de préférences conditionnelles* $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ *tel que chaque* $\mathtt{e} \in \mathtt{E}$ *(*$n = card(\mathtt{E})$*) est la cible d'une unique préférence et un sous-ensemble* $\Omega \subseteq \Gamma$*, un* modèle de préférences relaxées Γ_{Ω} *est une séquence de préférences* $(\varphi_1, \varphi_2, \ldots, \varphi_n)$ *telles que* $\varphi_i = \gamma_i$ *si* $\gamma_i \notin \Omega$*, et* $\varphi_i = \langle e_i \nsim \overline{e_i} \rangle$ *autrement. Notons que* $\Gamma_{\emptyset} = \Gamma$ *est un modèle de préférences conditionnelles.*

À partir d'un modèle de préférences relaxées Γ_{Ω}, on définit l'ordre partiel $\prec_{\Gamma_{\Omega}}$ entre les paires d'états de la même manière qu'un modèle de préférences conditionnelles (définition 25 page 24) : $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) est strictement préféré à (s_{pre}, s') d'après Γ_{Ω} (noté $(s_{pre}, s) \prec_{\Gamma_{\Omega}} (s_{pre}, s')$) si et seulement si il existe $i \in [1, n]$ tel que pour tout $j < i$, $(s_{pre}, s) \approx_{\varphi_j} (s_{pre}, s')$ et $(s_{pre}, s) \prec_{\varphi_i} (s_{pre}, s')$.

On généralise maintenant les notions associées aux estimateurs à état unique (voir chapitre 3) aux modèles de préférences relaxées. Au cours de ce procédé, on perd la propriété de la proposition 3 : dans un modèle relaxé, étant donné un état précédent et une observation, la transition préférée n'est pas toujours unique. On parle alors de transition non-dominée s'il n'existe pas de transition strictement préférée, c'est-à-dire si elle est maximale dans $\prec_{\Gamma_{\Omega}}$. En fait, en relaxant des préférences, on crée des transitions non-dominées, comme décrit dans la proposition suivante.

Proposition 7 (Relaxation de préférences). *Soit* (s_0, Δ, Γ) *un modèle de diagnostic, et soit* $\Gamma_{\Omega 1}$ *et* $\Gamma_{\Omega 2}$ *deux relaxations de* Γ *telles que* $\Omega 1 \subseteq \Omega 2$*. Si une transition n'est pas dominée dans* $\Gamma_{\Omega 1}$*, alors elle ne l'est pas non plus dans* $\Gamma_{\Omega 2}$ *:*

$$\begin{aligned} \forall (s_{pre}, s) \in \mathcal{S}^2, \forall o \in \mathcal{O}, \\ & (\nexists s_{best} \in cands(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)) \\ & \Rightarrow (\nexists s_{best} \in cands(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)) \end{aligned}$$

Démonstration. On prouve que si il y avait un état préféré s_{best} tel que $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)$, alors on aurait $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)$. Soit e la première variable (d'après l'ordre des préférences) telle que $s_{best}(\mathsf{e}) \neq s(\mathsf{e})$ et soit $\gamma \in \Gamma$ la préférence conditionnelle associée. Alors, pour chaque préférence γ' antérieure à γ dans la séquence Γ, (s_{pre}, s_{best}) et (s_{pre}, s) sont équivalentes. Toutes les préférences antérieures à γ dans Γ dépendent de variables qui ont la même valeur dans s_{best} et s. Alors pour ces préférences (relaxées ou non), les transitions (s_{pre}, s_{best}) et (s_{pre}, s) sont équivalentes. Si φ_1 et φ_2 sont toutes deux conditionnelles ou toutes deux relaxées, alors $\prec_{\varphi_2}=\prec_{\varphi_1}$ et $\approx_{\varphi_2}=\approx_{\varphi_1}$. En conséquence, $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)$ est équivalent $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)$. Soient φ_1 une préférence conditionnelle et φ_2 une préférence relaxée (cela arrive lorsque $\gamma \in \Omega 2 - \Omega 1$). Alors, $(s_{pre}, s_{best}) \prec_{\varphi_2} (s_{pre}, s)$ est faux puisque φ_2 est relaxée. Alors, $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)$ est faux et cela implique que φ_2 est une préférence conditionnelle. Cela est faux car $(s_{pre}, s_{best}) \prec_{\varphi_2} (s_{pre}, s)$ est impossible puisque φ_2 est relaxée, et $(s_{pre}, s_{best}) \approx_{\varphi_2} (s_{pre}, s)$ est incompatible avec $s_{best}(\mathsf{e}) \neq s(\mathsf{e})$. Alors l'implication est vraie, de même que la proposition 7. ■

Par conséquent, dans un modèle de préférences relaxées, l'étape d'estimation ne retourne plus un état unique mais un ensemble d'états correspondant à l'ensemble des états successeurs des transitions non-dominées.

Définition 38 (Estimation avec Γ_Ω). *On considère une relation de transition Δ, un modèle de préférences relaxées Γ_Ω, un état précédent estimé $\hat{s}_{pre} \in \mathcal{S}$ et une observation o. Une* estimation *consiste à trouver l'ensemble $\mathcal{S}_\Delta^{\Gamma_\Omega}(s_{pre}, o) \subseteq cands(s_{pre}, o)$ contenant les états issus des transitions non-dominées dans $\prec_{\Gamma_\Omega}$. Formellement :*

$$\mathcal{S}_\Delta^{\Gamma_\Omega}(s_{pre}, o) = \{\hat{s} \in cands(s_{pre}, o) | \nexists s_{best} \in cands(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_\Omega} (s_{pre}, \hat{s})\}$$

Puisqu'à chaque étape discrète, l'estimation avec Γ_Ω n'est pas unique, alors pour une séquence d'observations donnée, la séquence d'états estimés par Γ_Ω ne l'est pas non plus.

Définition 39 (Séquence d'états estimés avec Γ_Ω). *Étant donnés un état initial $s_0 \in \mathcal{S}$, une relation de transition Δ, un modèle de préférences relaxées Γ_Ω, on définit les séquences d'états estimés comme suit :*

- *(s_0) est l'unique séquence d'état estimé pour la séquence d'observations vide $()$*
- *$(s_0, \hat{s}_1, \ldots, \hat{s}_k)$ est une séquence d'états estimés pour la séquence d'observations $(o_1, \ldots, o_k)$ si et seulement si $(s_0, \hat{s}_1, \ldots, \hat{s}_{k-1})$ est une séquence d'états estimés pour $(o_1, \ldots, o_{k-1})$ et $\hat{s}_k \in \mathcal{S}_\Delta^{\Gamma_\Omega}(\hat{s}_{k-1}, o_k)$*

Corollaire 1. *D'après la proposition 7, pour une séquence d'observations donnée et pour deux ensembles de préférences $\Omega 1 \subseteq \Omega 2 \subseteq \Gamma$, $\Gamma_{\Omega 2}$ accepte un sur-ensemble de séquences d'états estimés acceptées elles aussi par $\Gamma_{\Omega 1}$. C'est pourquoi on parle de relaxation de préférences. En conséquence, si on considère une séquence d'observations qui est une impasse pour $\Gamma_{\Omega 1}$, il est possible qu'il existe une (ou plusieurs) séquence(s) d'états estimés avec $\Gamma_{\Omega 2}$ pour cette séquence d'observations.*

Afin de vérifier si un modèle de préférences relaxées admet une séquence d'états estimés pour une séquence d'observations donnée, on peut réaliser plusieurs tests de cohérence décrits dans cette section. On cherche le(s) plus petit(s) ensemble de préférences qui, lorsqu'elles sont relaxées permettent d'éliminer le scénario d'impasse. Pour cela, on peut utiliser un algorithme de diagnostic à base de cohérence similaire à celui de [Reiter, 1987]. On adapte donc la définition de diagnostic et de diagnostic minimal pour nos modèles d'estimation.

5.1.2 Formalisation du méta-diagnostic des préférences

Afin de transposer le formalisme du méta-diagnostic (voir chapitre 2) à notre problème, on considère que les préférences sont les méta-composants puisque c'est celles-ci qu'on

cherche à blâmer. Pour définir le problème de méta-diagnostic, nous adoptons la forme dépliée du modèle d'estimation présenté en section 4.3.1. Dans cette forme, les variables du système, les contraintes de Δ et les préférences de Γ sont dupliquées chaque pas de temps sur un intervalle de temps borné. Ici, cet intervalle est celui qui permet de représenter *seqObs*.

Proposition 8. *Transposition au formalisme de méta-diagnostic de [Belard et al., 2011] Le modèle d'estimation* (s_0, Δ, Γ) *et l'impasse seqObs sont associés au problème de méta-diagnostic suivant :*

- $M\text{-}COMPS = \Gamma = \{\ \gamma_1, \gamma_2, .., \gamma_n\ \}$,
- $M\text{-}SD =$
  $\{\Delta^t \mid t \in [1,n]\} \cup$
  $\{\neg AB(\gamma_i) \rightarrow ((\forall e_i^t, \exists e_{i+1}^t, \ldots, \exists e_n^t, \Delta^t) \rightarrow (e_i^t \leftrightarrow cond_i^t)) \mid \gamma_i \in \Gamma, t \in [1,n]\} \cup$
  $\{s_0^0\}$, *l'état initial sous forme dépliée,*
- $M\text{-}OBS = seqObs^t$, *l'impasse sous forme dépliée.*

Dans cette configuration et en appliquant l'approche de diagnostic à base de cohérence [Reiter, 1987], un sous ensemble de préférences Ω est un diagnostic si et seulement si :

$$(M\text{-}SD \wedge M\text{-}OBS \wedge \bigwedge_{\gamma \in \Gamma - \Omega} \neg AB(\gamma) \nvdash \bot)$$

On essaye dans ce chapitre de construire un système de vérification d'impasse pour un modèle de préférence relaxées. On considère une fonction qui prend en entrée un modèle d'estimation (s_0, Δ, Γ), l'impasse *seqObs* et un sous ensemble $\Omega \subseteq \Gamma$ et qui vérifie si il existe une séquence d'états estimés avec Γ_Ω conformément à la définition 39.

$$\neg Impasse(s_0, \Delta, \Omega, seqObs) \text{ si et seulement si } (M\text{-}SD \wedge M\text{-}OBS \wedge \bigwedge_{\gamma \in \Gamma - \Omega} \neg AB(\gamma) \nvdash \bot)$$

On cherche alors un méta-diagnostic minimal Ω ; un sous-ensemble minimal de préférences à modifier pour éliminer le scénario d'impasse.

Définition 40 (Méta-Diagnostic des préférences). *Soient* (s_0, Δ, Γ) *un modèle d'estimation, et* $(o_1, \ldots, o_k)$ *un scénario d'impasse pour ce modèle. Un ensemble de préférences* $\Omega \subseteq \Gamma$ *est un méta-diagnostic si et seulement si il existe une séquence d'états estimés pour le modèle de préférences relaxées* $(s_0, \Delta, \Gamma_\Omega)$*. Un méta-diagnostic* Ω *est un méta-diagnostic minimal si et seulement si il n'existe aucun méta-diagnostic* $\Omega' \subseteq \Gamma$ *tel que* $\Omega' \subset \Omega$*.*

Un méta-diagnostic Ω est interprété de la manière suivante : il est possible de modifier les préférences de Ω afin que l'estimateur ne rencontre plus de scénario d'impasse pour la séquence d'observations associée. D'après la proposition 7, si Ω est un méta-diagnostic, alors tous les sur-ensembles de Ω sont aussi des méta-diagnostics.

Exemple 11 (Méta-diagnostic des préférences). *On considère le modèle d'estimation présenté dans les exemples 2 et 4 du chapitre 3 avec* s_0 *=move engine* $\overline{f_{engine}}\ \overline{f_{slip}}$

$$\Delta = \left\{ \begin{array}{c} \texttt{move} \leftrightarrow \texttt{engine} \wedge \neg \texttt{f}_{\texttt{move}} \wedge \neg \texttt{f}_{\texttt{engine}} \\ \texttt{pre\_f}_{\texttt{engine}} \rightarrow \texttt{f}_{\texttt{engine}} \end{array} \right\}$$

$$\Gamma = \left(\begin{array}{ll} \texttt{pre\_f}_{\texttt{move}} : f_{move} \prec \overline{f_{move}} & (\gamma_1) \\ \neg \texttt{engine} : f_{engine} \prec \overline{f_{engine}} & (\gamma_2) \end{array} \right)$$

En considérant trois observations $o_1 = \overline{move}engine$ et $o_2 = o_0 = moveengine$, on a vu dans le chapitre précédent que la séquence d'observations (o_0, o_1, o_2) était un scénario d'impasse. On s'intéresse au méta-diagnostic des préférences et on considère donc deux sous ensembles de préférences $\Omega_1 = \{\gamma_1\}$ et $\Omega_2 = \{\gamma_2\}$.

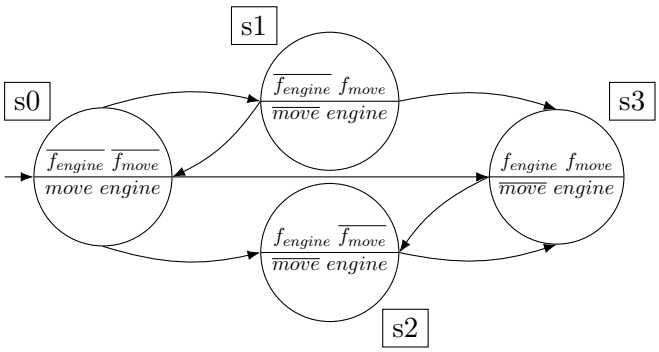


FIGURE 5.1 – Représentation en machine de Moore du système de l'exemple 11, on étiquette chaque état afin d'améliorer la clarté des explications.

— Avec $\Gamma_{\Omega 2} = \Omega_1$, le modèle où γ_2 est relaxé, en partant de l'état initial s_0 et en recevant l'observation o_1, en appliquant seulement la préférence γ_1 on estime donc les états dans lesquels $\texttt{f}_{\texttt{move}}$ est faux. On estime donc l'état s_2. En recevant à l'étape suivante l'observation o_2, il n'existe aucun candidat pour l'état précédent s_2 et l'observation o_2, le scénario d'impasse subsiste.
— Avec $\Gamma_{\Omega 1} = \Omega_2$, le modèle où γ_1 est relaxé, en partant de l'état initial s_0 et en recevant l'observation o_1, on ne fait pas de choix sur la variable $\texttt{f}_{\texttt{move}}$ et on conserve donc les trois candidats s_1,s_2 et s_3. On applique ensuite la préférence γ_2, puisqu'on a *engine* dans s_0, on estime les états dans lesquels $\texttt{f}_{\texttt{engine}}$ est faux. On estime donc l'état s_1. En recevant à l'étape suivante l'observation $o2$, il existe bien un successeur à s_2 : s_0 car on a $(s_2,s_0) \in \Delta$. Le scénario d'impasse est éliminé.

On constate qu'en appliquant seulement la préférence γ_1 le scénario d'impasse subsiste tandis qu'en appliquant seulement la préférence γ_2 celui-ci est éliminé. On en déduit donc que Ω_1 est un méta-diagnostic (et c'est aussi le méta-diagnostic minimal).

5.2 Encodage du problème

Cette section présente deux encodages pour le problème du méta-diagnostic des préférences. La première approche consiste à utiliser le model-checker *Electrum* à partir d'un scénario d'impasse préalablement retourné par celui-ci. La deuxième approche consiste à utiliser directement un solver SAT.

5.2.1 Encodage en Electrum

L'approche *Electrum* consiste, à partir d'un scénario d'impasse, à créer de nouveaux fichiers de vérifications utilisant le principe du Twin-plant comme décrit dans le chapitre précédent. On réalise le même encodage du Twin-plant à la différence qu'on ajoute et modifie plusieurs prédicats. L'approche pour vérifier si un sous-ensemble de préférences Ω est un méta-diagnostic consiste à :

1. Forcer la séquence d'observations correspondant au scénario d'impasse.
2. Supprimer les prédicats d'application des préférences de Ω.
3. Vérifier s'il existe un estimateur sans impasse dans cette configuration.

On automatise la génération de nouveaux fichiers *Electrum* à partir d'un scénario d'impasse afin d'obtenir tous les fichiers de vérification pour chaque sous-ensemble possible de Γ. Les trois figures ci-dessous illustrent un encodage dans lequel on désactive la première préférence.

```
fact frozen_obs_t1 {
    Estimator.move' = False
    Estimator.engine' = True
}
fact frozen_obs_t2 {
    Estimator.move'' = True
    Estimator.engine'' = True
}
```

FIGURE 5.2 – On force la séquence d'observations en utilisant l'opérateur ′ d'*Electrum* faisant référence à l'état suivant. La concaténation de n opérateurs fait référence à l'étape $t + n$ et permet donc d'imposer des valeurs booléennes aux observations sur un horizon borné.

```
pred preference_fmove_applied{}

pred preference_fengine_applied{
  isTrue[Estimator.fengine] <=>
    isFalse[engine]
}
```

FIGURE 5.3 – Représentation *Electrum* d'une préférence relaxée et une préférence active.

```
assert deadlock_free_Estimator{
    no e :Estimator  | always {
        delta[
        e.move,
        e.engine,
        e.fmove,
        e.fengine,
        e.pre_fengine]
    }
}
check deadlock_free_path_Estimator
```

FIGURE 5.4 – La propriété vérifiée en *Electrum* pour le méta-diagnostic des préférences. On cherche l'existence d'un estimateur qui est toujours capable de satisfaire Δ, donc de produire une séquence d'estimation cohérente

L'encodage *Electrum* permet de réaliser le méta-diagnostic des préférences pour un scénario d'impasse donné. Cependant cette approche demande la création d'un nouveau fichier pour chaque méta-diagnostic possible, donc pour chaque sous-ensembles de Γ. Même si ce processus est automatisé, cela reste fastidieux et peut s'avérer coûteux en temps et en mémoire pour des modèles de préférences de grande taille. Tout comme dans la deuxième partie du chapitre précédent, on s'intéresse dans la section suivante à un encodage utilisant directement un solveur SAT sans passer par un model-checker.

5.2.2 Encodage en SAT itératif

Afin de vérifier si un sous-ensemble de préférences Ω est un méta-diagnostic, on réalise plusieurs séries de requêtes SAT qui recréent les candidats non-dominés à chaque étape (voir définition 28 page 54). On reprend l'approche des séquences d'états dépliées sur un horizon borné comme décrit dans la proposition 6 et on déplie ainsi le modèle sur un nombre k d'étapes discrètes correspondant à la taille du scénario d'impasse étudié. On utilise une structure de données que l'on appelle *scenario* afin de représenter des séquences d'estimations partiellement construites. Un objet scénario est constitué de 3 attributs :

— *decisions* est une formule représentant les choix d'estimation effectués jusqu'alors ;

— *step* $\in [1, k]$ est une étape discrète ;
— *pref* est la prochaine préférence qu'on tente d'appliquer.

On couple ensuite deux algorithmes afin de générer et traiter des *scenarios*. L'algorithme 3 génère des objets de type *scenario*. La fonction *incr* affecte l'attribut *pref* à la prochaine préférence à appliquer si celle-ci existe, sinon elle augmente l'attribut *step* et remet l'attribut *pref* à la première préférence. L'objet *decisions* contient tous les successeurs possibles pour la préférence courante *pref*. Lorsqu'une préférence *pref* ciblant une variable `e` est relaxée (ligne 13), les successeurs *curSc* et *curScBis*, respectivement avec `e` à vrai (ligne 15) et à faux (ligne 16), sont tous les deux possibles. Lorsque *pref* est conditionnelle (ligne 22), alors il n'y a qu'une seule décision possible. La cohérence avec Δ est testée avec la fonction *isSAT* faisant directement appel à un solveur SAT avec les contraintes de la version dépliée du modèle Δ (lignes 17, 19 et 23).

Dans l'algorithme 2, chaque *scenario* est ajouté à une file d'attente et à chaque itération on cherche à construire tous les successeurs possibles du *scenario* traité. L'algorithme 3 est initialisé avec une liste de *scenarios* initialisée avec $(\top, 1, \gamma_1)$ (ligne 8), ce qui exprime qu'aucun choix de décision n'a été fait jusqu'alors, l'étape discrète est fixée à 1 et la prochaine préférence à appliquer est γ_1. À chaque itération, on construit les successeurs d'un état en faisant appel à l'algorithme 4 (ligne 14). Les *scenarios* pour lesquels l'attribut *decision* est incohérent avec Δ (dans sa version dépliée $\Delta^0, \ldots, \Delta^k$) sont rejetés. L'algorithme termine quand un *scenario* complet (ligne 11) est trouvé (dans ce cas Ω est un méta-diagnostic) ou alors lorsqu'il ne reste aucun *scenario* dans la file d'attente (dans ce cas Ω n'est pas un méta-diagnostic) (ligne 16).

Algorithm 2 *MetaDiagnosis*($\{impasse^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$) : retourne vrai si et seulement si Γ_Ω est un méta-diagnostic

1: **Entrees**
2: $\{impasse^t | t \in [1,k]\}$: une impasse dépliée sur l'horizon k
3: $\{\Delta^t | t \in [1,k]\}$: Δ déplié sur l'horizon k
4: Γ_Ω : un sous ensembles de préférence à tester
5: **Sorties**
6: retourne vrai si et seulement si Ω est un méta-diagnostic
7: **function** isAMetaDiagnosis($impasse^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}$, Γ_Ω)
8: *scenarios* $\leftarrow \{(\top, 1, \gamma_1)\}$
9: **while** *scenarios* $\neq \emptyset$ **do** `select` *curScen* `in` *scenarios*
10: *scenarios* $\leftarrow$ *scenarios* $- \{curScen\}$
11: **if** *curScen.step* $>$ *length*(*impasse*) **then**
12: **return** *true*
13: **else**
14: *nextScen* $\leftarrow$ *makeNextScenarios*(*curScen*, $\{impasse^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$)
15: *scenarios* $\leftarrow$ *scenarios* $\cup$ *nextScen*
16: **return** *false*

On considère que l'ordre dans lequel les différents sous-ensembles de Γ sont testés est déduit directement par la théorie du diagnostic à base de cohérence [Reiter, 1987]. Si on trouve un Ω qui est un méta-diagnostic, alors on ne teste pas ses sur-ensembles et si on trouve un Ω qui n'est pas un méta-diagnostic, on ne teste pas ses sous-ensembles.

5.3 Expérimentations

Cette section présente les différentes expérimentations effectuées pour les deux approches présentées précédemment. Pour chaque approche, on reprend les modèles utilisés pour la

Algorithm 3 makeNextScenarios($curScen, \{impasse^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$)

Entrees
  $curScen$
  $\{impasse^t | t \in [1,k]\}$
  $\{\Delta^t | t \in [1,k]\}$
  Γ_Ω
Sorties
  nextScenarios : les scénarios suivants associés à $curScen$
function MAKENEXTSCENARIOS($curScen, \{impasse^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$)
  $t \leftarrow curScen.step$
  $\gamma \leftarrow curScen.curPref$
  $formula \leftarrow curScen.décision$;
  $scenarios \leftarrow \emptyset$
  if $\gamma = e \nsim \overline{e}$ **then**
    $curScBis \leftarrow copy(incr(curScen))$
    $curScen.décision \leftarrow formula \wedge \mathtt{e}^t$
    $curScBis.décision \leftarrow formula \wedge \neg \mathtt{e}^t$
    if $isSAT((\bigwedge_{i=1}^{t}(\Delta^i \wedge impasse^i) \wedge formula \wedge \mathtt{e}^t)$ **then**
      $scenarios \leftarrow scenarios \cup \{curScen\}$
    if $isSAT((\bigwedge_{i=1}^{t}(\Delta^i \wedge impasse^i) \wedge formula \wedge \neg \mathtt{e}^t)$ **then**
      $scenarios \leftarrow scenarios \cup \{curScBis\}$
    return $scenarios$
  else $//\gamma = \mathtt{cond} : e \prec \overline{e}$
    if $isSAT(\bigwedge_{i=1}^{t}(\Delta^i \wedge impasse^i) \wedge formula \wedge (\mathtt{e}^t \leftrightarrow \mathtt{cond}^t))$ **then**
      $curScen.décision \leftarrow formula \wedge (\mathtt{e}^t \leftrightarrow \mathtt{cond}^t)$
      return $\{incr(curScen)\}$
    else
      $curScen.décision \leftarrow formula \wedge (\neg \mathtt{e}^t \leftrightarrow \mathtt{cond}^t)$
      return $\{incr(curScen)\}$

recherche de scénarios d'impasse du chapitre précédent. Lorsque les modèles présentent un scénario d'impasse, on récupère celui-ci (une séquence d'observations) et on effectue le méta-diagnostic des préférences. L'architecture utilisée est la même que pour les expérimentations du chapitre précédent. On étudie ici les modèles pour lesquels un scénario d'impasse avait été détectée.

5.3.1 Expérimentations par l'approche Electrum

Les expérimentations effectuées avec Electrum reprennent les modèles sur lesquels la recherche de scénarios d'impasse avait été effectuée dans le chapitre précédent. Les scénarios d'impasse sont automatiquement récupérés sous forme de fichiers XML et traduits ensuite vers Scala. Le méta-diagnostic des préférences est effectué sur tous les modèles présentés dans les résultats de la figure 4.7 qui présentaient un scénario d'impasse. Lorsqu'un contre exemple est trouvé, cela signifie que le Ω testé est un méta-diagnostic. On regroupe les différentes préférences par rapport aux variables que celles-ci ciblent.

On a donc :
— Γ_{grid} : préférences associées aux variables de la grille
— Γ_{bots} : préférences associées aux fautes pouvant toucher les robots
— $\Gamma_{ftemp} \in \Gamma_{bots}$: préférences associées aux fautes intermittentes pouvant affecter les robots.
— $\Gamma_{fperm} \in \Gamma_{bots}$: préférences associées aux fautes permanentes pouvant affecter les robots.

— $\Gamma_{fpermbot1} \in \Gamma_{fperm}$: préférences associées à la faute permanente pouvant affecter le robot 1.
— $\Gamma_{fpermbot2} \in \Gamma_{fperm}$: préférences associées à la faute permanente pouvant affecter le robot 2.

Les figures 5.5 et 5.6 présentent les résultats pour les modèles numérotés 1 à 5 pour lesquels il existait un scénario d'impasse.

	$\Gamma =$				
ID	$\emptyset$	Γ - Γ_{grid}	Γ - Γ_{bots}	Γ - Γ_{ftemp}	Γ - Γ_{fperm}
1	méta-diag	non méta-diag	méta-diag	non méta-diag	méta-diag
	0,3s(0)	0,6s(3)	0,1s(4)	0,6s(6)	0,1s(6)
2	méta-diag	non méta-diag	méta-diag	non méta-diag	méta-diag
	0,5s(0)	2,7s(3)	0,5s(8)	2,7s(10)	0,4s(10)
3	méta-diag	non méta-diag	méta-diag	non méta-diag	méta-diag
	7,2s(0)	44,9s(3)	7,9s(12)	44,2s(14)	7,3s(14)

FIGURE 5.5 – Analyse des préférences pour les modèles avec 1 robot, le nombre de préférences qui ne sont pas relaxées est indiqué à côté du temps de calcul.

On constate que relaxer les préférences portant sur les robots permet d'éliminer le scénario d'impasse : Γ_{bots} est un méta-diagnostic. Lorsqu'on relaxe seulement les préférences ciblant les fautes intermittentes, le scénario d'impasse subsiste : Γ_{ftemp} n'est pas un méta-diagnostic Dans la figure 5.5, relaxer la préférence ciblant la faute permanente du robot permet d'éliminer le scénario d'impasse : Γ_{fperm} est un méta-diagnostic (minimal) pour les instances 1 à 3.

	$\Gamma =$					
ID	Γ - Γ_{grid}	Γ - Γ_{bots}	Γ - Γ_{ftemp}	Γ - Γ_{fperm}	Γ - $\Gamma_{fpermbot1}$	Γ - $\Gamma_{fpermbot2}$
4	non méta-diag	méta-diag	non méta-diag	méta-diag	non méta-diag	non méta-diag
	40,7s(6)	6,8s(8)	40,8s(12)	6,8s(12)	40,9s(13)	40,8s(13)
5	non méta-diag	méta-diag	non méta-diag	méta-diag	non méta-diag	non méta-diag
	885,4s(6)	150,0s(12)	895,8s(16)	150,7s(16)	900,1s(17)	903,7s(17)

FIGURE 5.6 – Analyse des préférences pour les modèles avec 2 robots, le nombre de préférences qui ne sont pas relaxées est indiqué à coté du temps de calcul.

On constate que lorsqu'on relaxe seulement la préférence de $\Gamma_{fpermbot1}$ ou la préférence de $\Gamma_{fpermbot2}$ dans les instance 4 et 5 (avec 2 robots) le scénario d'impasse n'est pas éliminé, ces deux ensembles ne sont donc pas des méta-diagnostics. Le méta-diagnostic minimal pour toutes les versions de cet exemple est donc $\Omega = \Gamma_{fperm}$ On pourra donc s'intéresser à modifier la condition de cette et ces préférences suivant les instances afin d'éliminer les scénarios d'impasse.

On constate une nette différence des temps d'exécution lorsque le Ω testé est ou n'est pas un méta-diagnostic. Lorsque Ω est un méta-diagnostic, le temps de calcul est assez court puisque dès qu'une séquence d'estimations est trouvée celle-ci correspond à un contre exemple qui est directement renvoyé. A l'inverse lorsque le scénario d'impasse subsiste, toutes les séquences d'estimations sont envisagées par le model-checker et on constate des temps de calculs jusqu'à 5 fois supérieurs. Dans les deux cas, les temps de calculs restent exponentiels à mesure qu'on augmente la taille du modèle.

5.3.2 Expérimentations par l'approche SAT

Les résultats pour l'approche SAT montrent des temps de calcul très court pour de petits ensembles de préférences ou pour des scénarios d'impasse de petite taille. Cependant, dès

Instance	$\|\Omega\| = 1$	$\|\Omega\| = \|\Gamma\|/2$	$\|\Omega\| = \|\Gamma\| - 1$
small_3	0 [0, 0]	0 [0, 0]	0 [0, 0]
small_4	0 [0, 0]	0 [0, 0]	0 [0, 0]
small_5	0 [0, 0]	0 [0, 0]	0 [0, 1]
small_6	0 [0, 0]	0 [0, 2]	1 [0, 2]
small_7	0 [0, 0]	1 [0, 17]	5 [0, 14]
small_8	0 [0, 0]	8 [0, 134]	37 [0, 108]
small_9	0 [0, 0]	61 [0, 1129]	311 [0, 960]
large_3	0 [0, 0]	0 [0, 2]	0 [0, 1]
large_4	0 [0, 0]	0 [0, 2]	3 [1, 4]
large_5	0 [0, 0]	14 [0, 163]	87 [16, 119]
large_6	0 [0, 0]	48 [0, 444]	2691 [652, 4730]

TABLE 5.1 – Résultats du méta-diagnostic pour un ensemble de préférences Ω . Les colonnes indiquent la taille des sous-ensembles Ω testés. Pour chaque test, on réalise le test sur au moins 20 ensembles possibles pour la taille donnée. Chaque case contient le temps moyen, le minimum et maximum sous la forme $moyenne[min, max]$.

lors qu'on traite de grands ensembles de préférences ou des scénarios d'impasse de grande taille, on constate que le temps de calcul croit de manière exponentielle.

On constate que certains ensembles sont particulièrement difficiles à vérifier et que le plus difficile d'entre eux peut prendre jusqu'à dix fois plus de temps par rapport au temps de calcul moyen. Cela arrive le plus souvent lorsqu'on teste la moitié des préférences de Γ puisque plus on relaxe de préférences, plus il est facile de trouver une séquence d'estimations.

5.4 Comparaison des résultats pour les deux approches

Les performances s'avèrent satisfaisantes du fait que la mémoire n'est pas un problème puisque la vérification s'effectue hors ligne. L'approche Electrum permet d'étudier un scénario d'impasse précis mais demande la génération de nombreux fichiers. A l'inverse, l'approche SAT est moins coûteuse en mémoire mais a aussi l'avantage d'explorer l'espace de recherche de manière plus efficace avec un parcours en profondeur pour les séquences d'estimations. Dans les deux approches, les temps de calculs restent exponentiels par rapport à des modèles de grande taille, une approche décentralisée pourrait s'avérer être une solution lorsqu'on analyse des systèmes avec des architectures complexes et reste une perspective.

Cependant lors d'expérimentations effectuées qui ne sont pas présentées ici, nous avons constaté qu'il existait des modèles pour lesquels le méta-diagnostic n'aboutissait pas (le méta-diagnostic minimal était égal à l'ensemble Γ). La figure 5.7 présente un SED sous forme de machine de Moore pour lequel il n'existe aucune stratégie d'estimation à état unique qui ne rencontrerait pas d'impasse.

Exemple 12 (Scénario d'impasse sans aucun méta-diagnostic). *On considère deux stratégie d'estimation $estim_1$ et $estim_2$ à état unique pour le système illustré en figure 5.7 avec :*

- $estim_1(s_0, b) = s_1$
- $estim_2(s_0, b) = s_2$

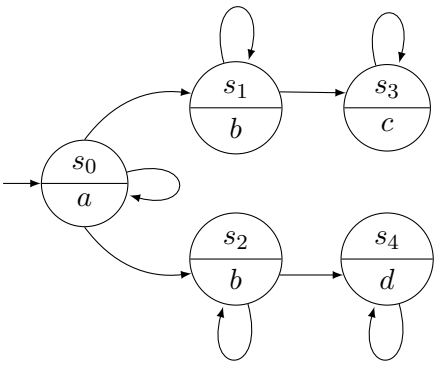
En partant de l'état initial s_0 et en recevant l'observation b, $estim_1$ estime l'état s_1 tandis que $estim_2$ estime s_2.

Lorsque la séquence d'observations produite par le système est (a, b, c)

- $cands(s_1, c) \neq \emptyset$ *: $estim_1$ est capable de sélectionner un état unique parmi les candidats (il s'agit de s_3).*
- $cands(s_2, c) = \emptyset$ *: $estim_2$ rencontre une impasse dans l'estimation.*

Lorsque la séquence d'observations produite par le système est (a, b, d)

FIGURE 5.7 – Un système dans lequel aucun méta-diagnostic n'est possible car celui-ci n'admet aucune stratégie d'estimation à état unique sans impasse.



— $cands(s_1, d) = \emptyset$ *: $estim_1$ rencontre une impasse dans l'estimation.*
— $cands(s_2, d) \neq \emptyset$ *: $estim_2$ est capable de sélectionner un état unique parmi les candidats (il s'agit de s_4).*

Il n'existe donc pas de stratégie d'estimation à état unique pour ce système qui ne rencontrerait pas d'impasse. Le méta-diagnostic minimal serait l'ensemble Γ.

Au début de ce chapitre, nous avons fait l'hypothèse que les scénarios d'impasse étaient dûs aux choix d'estimation. Cependant, l'exemple ci-dessus tend à montrer que cela peut également dépendre du modèle comportemental du système (son SED). On fait donc l'hypothèse dans le chapitre suivant qu'il existerait des systèmes pour lesquels une stratégie d'estimation à état unique sans impasse n'est pas possible. On tentera alors de définir une nouvelle propriété s'appliquant au SED pour l'estimation à état unique.

Chapitre 6

Estimabilité à état unique

Ce chapitre s'intéresse à la possibilité de construire une stratégie d'estimation à état unique pour un système donné sans que celle-ci ne rencontre d'impasse lors de l'exécution. En effet, comme illustré en fin de chapitre précédent, il arrive que pour certains systèmes l'élimination de tous les scénarios d'impasse par modification de la stratégie d'estimation soit impossible. Il est donc pertinent de pouvoir vérifier si un système est susceptible ou non d'admettre une telle stratégie.

Dans ce chapitre, la stratégie d'estimation n'est plus représentée comme une séquence de préférences mais comme une fonction d'estimation partielle qui à partir d'un état précédent et de l'observation de l'état courant, détermine ce dernier (lorsque celui-ci existe). Tout d'abord, le problème de l'estimabilité à état unique est défini comme la possibilité de construire une fonction d'estimation sans impasse à partir des spécifications du système. Afin de répondre à ce problème, la propriété d'*estimabilité à état unique* des SED est définie. On propose une condition nécessaire et suffisante pour cette propriété qui repose sur l'équivalence des langages. Une condition nécessaire est également présentée de manière à améliorer l'efficacité algorithmique.

Ensuite, deux approches permettant de résoudre le problème de l'estimabilité à état unique par construction d'une fonction d'estimation sans impasses sont présentées. On s'intéresse à la complexité algorithmique des deux approches proposées ainsi qu'à la classe de complexité du problème d'estimabilité à état unique. Enfin, il est montré que des contraintes de correction peuvent être intégrées dans la recherche d'une fonction d'estimation avec la deuxième approche afin de raffiner la qualité de l'estimation souhaité par l'utilisateur. Des expérimentations sont ensuite menées et discutées sur des modèles de taille variable.

6.1 Le problème de l'estimabilité à état unique

Dans cette section, le problème de l'estimabilité à état unique est présenté. Il s'agit d'être capable, à partir des spécifications d'un système, de savoir s'il est possible de construire une stratégie d'estimation à état unique qui ne rencontre jamais d'impasse. Si un système peut être estimé par une telle stratégie, on dira qu'il est estimable à état unique.

6.1.1 Définition du problème

Définition 41. *Système estimable à état unique*
Un système est estimable à état unique *si et seulement s'il existe une stratégie d'estimation pour ce système qui ne rencontre jamais d'impasse.*

Définition 42. *Le problème d'estimabilité à état unique*
Le problème d'estimabilité consiste à décider si un système est estimable à état unique.

Afin de formaliser le problème d'estimabilité à état unique, la stratégie d'estimation n'est plus représentée sous forme d'une séquence de préférences conditionnelles comme dans les chapitres précédents, mais désormais comme une fonction d'estimation partielle qui détermine l'état courant à partir de l'observation reçu et de l'état précédemment estimés.

Définition 43 (Fonction d'estimation). *Soit (s_0, Δ) un SED avec pour ensemble d'états S et ensemble d'observations O. Une fonction d'estimation est une fonction partielle estim : $(S \times O) \to S$ qui sélectionne un état unique parmi les candidats, c'est-à-dire pour chaque $s \in S$ et $o \in O$, si $cands(s, o) \neq \emptyset$ alors $estim(s, o) \in cands(s, o)$.*

6.1.2 Équivalence entre fonction d'estimation et préférences

À partir d'un SED (s_0, Δ), on montre qu'il y a équivalence entre système de préférences et fonction d'estimation.

Proposition 9. *Soit (s_0, Δ) un SED avec S comme ensemble d'états et O comme ensemble d'observations.*

- *Si estim est une fonction d'estimation associée à (s_0, Δ) alors il existe un système de préférences Γ pour lequel les états préférés sont ceux renvoyés par la fonction estim.*
- *Si Γ est un système de préférences, alors il existe une fonction estim qui renvoie les états préférés.*

Démonstration.

- pour une fonction *estim* donnée, on considère $P_{estim} =$ $\{(s, o) \in S \times O | estim$ est définie pour $(s, o)\}$. Pour chaque $e \in \mathtt{E}$, on créé l'ensemble des états images de *estim* dans lesquels $\mathtt{e}$ est vraie. Formellement, $True(\mathtt{e}) = \{(s, o) \in P_{estim} | estim(s, o)(e) = \top\}$. On crée ensuite pour chaque $\mathtt{e} \in \mathtt{E}$ la préférence γ_e de la manière suivante : $\gamma_e = \bigvee_{(s,o) \in True(e)} \mathtt{form}(o) \wedge \mathtt{form_{pre}}(s) : e \prec \overline{e}$. On définit $\mathtt{form}(o)$ et $\mathtt{form_{pre}}(s)$ afin de transformer une affectation en contrainte comme suit :

  - Pour une observation o, $\mathtt{form}(o) = \bigwedge_{\substack{\mathtt{vo} \in \mathtt{O} \\ o(\mathtt{vo}) = \top}} \mathtt{vo} \wedge \bigwedge_{\substack{\mathtt{vo} \in \mathtt{O} \\ o(\mathtt{vo}) = \bot}} \neg \mathtt{vo}$;
  - Pour un état précédent s, $\mathtt{form_{pre}}(s) = \bigwedge_{\substack{\mathtt{vs} \in \mathtt{P} \\ s(\mathtt{vs}) = \top}} \mathtt{pre\_vs} \wedge \bigwedge_{\substack{\mathtt{vs} \in \mathtt{P} \\ s(\mathtt{vs}) = \bot}} \neg \mathtt{pre\_vs}$.

  Si on note Γ l'ensemble de tous les γ_e pour tout e dans $\mathtt{E}$ et si pour chaque $(s, o) \in P_{estim}$ on note $\hat{s}$ l'état de S préféré par (s_0, Δ, Γ), on vérifie que $\hat{s} = estim(s, o)$
- pour un système de préférences Γ donné, on parcourt tous les couples (s, o) tels que $cands(s, o) \neq \emptyset$, on calcule l'état estimé $\hat{s}$ préféré et on construit $estim(s, o) = \hat{s}$.

■

D'après la proposition 9, si un système admet une fonction d'estimation sans impasses, alors il existe au moins une séquence de préférences conditionnelles pouvant être utilisée comme stratégie d'estimation à état unique. Cependant, s'il est impossible de construire une telle fonction d'estimation, alors le modèle de préférences ne sera pas applicable pour un tel système.

Exemple 13. *On considère le système suivant avec 1 variable observable et 1 variable à estimer. On observe la présence ou l'absence d'une alarme et on doit en déduire la présence ou l'absence d'une faute. Le comportement du système est décrit par les ensembles de variables et les ensembles d'état suivant :*

- $\mathtt{vS} = \{\mathtt{alarme}, \mathtt{faute}\}$
- $\mathtt{vO} = \{\mathtt{alarme}\}$
- $O = \{alarme, \overline{alarme}\}$
- $S = \{(alarme, faute), (\overline{alarme}, faute),$
  $(alarme, \overline{faute}), (\overline{alarme}, \overline{faute})\}$

On considère que le système évolue de la manière suivante : si l'alarme se déclenche, alors on sait que la faute est apparue dans le système. Si la faute est apparu dans le système à l'étape précédente, alors celle-ci est toujours présente dans l'état courant ; cette faute est permanente. On considère que dans son état initial, l'alarme n'est pas déclenchée et la faute est absente. Ce comportement est décrit par les contraintes suivantes :

$$\Delta = \left\{ \begin{array}{c} \mathtt{alarme} \rightarrow \mathtt{faute} \\ \mathtt{pre}(\mathtt{faute}) \rightarrow \mathtt{faute} \end{array} \right\}$$

- $s_0 = \overline{alarme}, \overline{faute}$

La figure 6.1 représente ce système sous forme de machine de Moore.

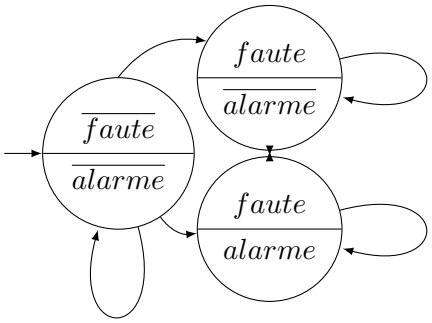


FIGURE 6.1 – Le système (s_0, Δ) représenté sous forme de machine de Moore, on ne représente que les variables estimées dans la partie haute de l'état par soucis de clarté

On considère maintenant deux séquences de préférences conditionnelles Γ_1 et Γ_2 afin d'estimer l'état du système : Puisqu'il n'y a qu'une seule variables à estimer, chacune des deux séquence contient une unique préférence conditionnelle, respectivement γ_1 et γ_2 telle que :

- $\gamma_1 = \overline{alarm} : faute \prec \overline{faute}$
- $\gamma_2 = \overline{alarm} : \overline{faute} \prec faute$

Le système présenté en figure 6.1 est estimable à état unique puisqu'avec les séquences de préférences Γ_1 et Γ_2 on ne rencontre jamais d'impasse. Cependant, chacune des deux théorie de préférences conditionelles estiment le système d'une manière différente :

- Γ_1 préfère les états ou la faute est **présente** lorsque l'alarme ne se déclenche pas
- Γ_2 préfère les états ou la faute est **absente** lorsque l'alarme ne se déclenche pas

On cherche à représenter les séquences de préférences sous forme d'une fonction d'estimation de la forme suivante :

$$estim : S \times O \to S \text{ telle que } \forall s \in S, \forall o \in O, estim(s, o) \in cands(s, o)$$

On peut extraire les fonctions d'estimations à partir de Γ_1 et Γ_2. Pour Γ_1 on obtient une fonction $estim_{\Gamma_1}$ telle que :

— $estim_{\Gamma_1}(\overline{fautealarme},\overline{alarm}) = \overline{fautealarme}$
— $estim_{\Gamma_1}(\overline{fautealarme},\overline{alarm}) = \overline{fautealarme}$
— $estim_{\Gamma_1}(\overline{fautealarme},alarm) = fautealarme$

Pour Γ_2 on obtient une fonction $estim_{\Gamma_2}$ telle que :

— $estim_{\Gamma_2}(\overline{fautealarme},\overline{alarm}) = \overline{fautealarme}$
— $estim_{\Gamma_2}(\overline{fautealarme},alarm) = fautealarme$

Notons que lorsque l'alarme apparaît, la présence de la faute est déduite avec certitude indépendamment de Γ_1 ou Γ_2.

6.1.3 Définition des langages

La résolution du problème de l'*estimabilité à état unique* repose sur l'utilisation des langages associés aux automates. On réutilise ici la définition du langage observable présentée dans le chapitre 2 (définition 9 page 12). Une fois le langage observable défini, on s'intéresse à l'ensemble des mots qui peuvent être générés avec une fonction d'estimation. Pour cela, on définit donc le langage accepté par une fonction d'estimation, c'est à dire l'ensemble des séquences d'observations cohérentes pour lesquelles il existe une séquence d'estimation associée.

Définition 44 (Langage accepté par une fonction d'estimation). *Soit (s_0, Δ) un SED avec espace d'états S et espace d'observations O, $sobs \in \mathcal{L}_{obs}(\Delta)$ une séquence d'observations et $estim : (S \times O) \to S$ une fonction d' estimation. Le langage accepté par cette fonction d'estimation, noté $\mathcal{L}_{obs}(estim)$, est l'ensemble de toutes les séquences d'observations pour lesquelles il existe une séquence d'estimations. Formellement :*

$$\begin{aligned} \mathcal{L}_{obs}(estim) = \{sobs \in \mathcal{L}_{obs}(\Delta) \mid (|sobs| > 1) \\ et\ \exists sest \in \mathcal{L}(\Delta) \\ t.q.\ \ pour\ i \in [1, |sobs|], obs(sest[i]) = sobs[i] \\ et\ estim(sest[i-1], sobs[i]) = sest[i]\} \end{aligned}$$

6.1.4 Condition nécessaire et suffisante pour l'estimabilité

On propose une condition nécessaire et suffisante pour l'estimabilité à état unique basée sur l'existence d'une fonction d'estimation dont le langage accepté est égal au langage observable du système. Cela garantit que pour chaque séquence d'observations générée par le système, il existe une séquence d'estimations. Dans une telle configuration, l'existence de scénario d'impasse devient impossible.

Proposition 10 (Condition d'estimabilité). *Un SED (s_0, Δ) est* estimable à état unique *si et seulement s'il existe une fonction d'estimation dont le langage accepté est égal au langage des observations du système :*

$$\exists\, estim : (S \times O) \to S \text{ telle que } \mathcal{L}_{obs}(estim) = \mathcal{L}_{obs}(\Delta)$$

Démonstration. Par construction de la fonction *estim*, on a $\mathcal{L}_{obs}(estim) \subseteq \mathcal{L}_{obs}(\Delta)$. L'absence d'impasse correspond à la capacité d'*estim* d'estimer toutes les séquences d'observations générées par le système. On a donc équivalence entre égalité des langages et *estimabilité à état unique.* ■

Si la condition de la proposition 10 est satisfaite, alors la fonction d'estimation fournit une séquence d'estimations pour tous les éléments de $\mathcal{L}_{obs}(\Delta)$. Il n'y a donc pas d'impasse. La principale difficulté est de trouver une telle fonction d'estimation ou de prouver qu'elle n'existe pas.

6.1.5 Condition nécessaire pour l'estimabilité

Lors des études menées pour la définition de la propriété d'*estimabilité à état unique*, une première condition nécessaire a été mise en évidence. Celle-ci porte sur l'analyse locale d'un état et de ses successeurs et définit la notion d'état non-bloquant, c'est-à-dire les états qui ne rencontrent pas d'impasse à l'instant suivant. On considère donc qu'un état est non-bloquant pour une certaine séquence d'observations, si et seulement s'il existe un ensemble de candidat non vide pour cet état et chaque continuation possible de la séquence d'observations.

Définition 45 (État non-bloquant). *Soient (s_0, Δ) un SED, et $sobs \in \mathcal{L}_{obs}(\Delta)$ une séquence d'observations. Un état $\hat{s}$ est* non-bloquant *pour $sobs$ si celui-ci est atteignable par $sobs$ et pour chaque observation ultérieure o, il existe au moins un candidat. Formellement, $\hat{s} \in reach(sobs)$ est* non-bloquant *si et seulement si :*

$$\forall o \in O \text{ si } sobs.o \in \mathcal{L}_{obs}(\Delta), \text{ alors } cands(\hat{s}, o) \neq \emptyset$$

Notons qu'un état est dit non-bloquant pour une séquence d'observations particulière. Il peut arriver qu'un état soit non-bloquant pour une certaine séquence d'observations et bloquant pour une autre. Une proposition de condition pour l'estimabilité à état unique est dérivée de cette notion d'état bloquants.

Exemple 14 (État non-bloquant et état bloquant). *On considère le système illustré en figure 6.2 et la séquence d'observations $seqObs = (a, b)$. L'ensemble des états atteignables pour cette séquence d'observations $reach(sobs)$ est $\{s_1, s_2\}$. On considère maintenant les continuations possibles de $seqObs$ de type $seqObs.o$ avec $o \in O$. Lorsque $seqObs.o = (a, b, b)$ les états s_1 et s_2 ont chacun un successeur (eux-mêmes). On peut donc dire que s_1 et s_2 sont non-bloquants pour la séquence (a, b, b). Si on considère maintenant la continuation $o' = c$ telle que $seqObs.o' = (a, b, c)$, on a :*

- *$cands(s_1, o') = \{s_3\}$: s_1 est non bloquant pour la séquence (a, b, c)*
- *$cands(s_2, o') = \emptyset$: s_2 est bloquant pour la séquence (a, b, c)*

Proposition 11 (Condition sur les états non-bloquants). *S'il existe une séquence d'observations $sobs \in \mathcal{L}_{obs}(\Delta)$ telle que $reach(sobs)$ ne contient que des états bloquants, alors le système n'est pas estimable.*

Démonstration. Soit une séquence d'observations *sobs* telle que $reach(sobs)$ ne contient que des états bloquants. Soit *estim* une fonction d'estimation et soit $\hat{s}$ l'état estimé par *estim* pour *sobs*. Par hypothèse, $\hat{s}$ est bloquant : il existe donc une continuation de *sobs*, notée $sobs.o$ qui n'a pas de successeur pour $\hat{s}$. Cela signifie que $sobs.o$ est une impasse. Le système n'est donc pas estimable à état unique. ■

Intuitivement, la proposition 11 signifie que si l'ensemble des états atteignables $reach(sobs)$ pour une séquence d'observations *sobs* de $\mathcal{L}_{obs}(\Delta)$ ne contient que des états bloquants, alors quel que soit l'état $\hat{s} \in reach(sobs)$ choisi, il existe une continuation $sobs.o \in \mathcal{L}_{obs}(\Delta)$ qui est une impasse. Il est donc impossible, dans cette configuration, de construire un estimateur qui ne rencontrerait pas d'impasse.

Proposition 12 (Transitions entre états non bloquants). *Soit $sobs \in \mathcal{L}_{obs}(\Delta)$ une séquence d'observations et $o \in O$ une observation telle que $sobs.o \in \mathcal{L}_{obs}(\Delta)$. Si le système est estimable, alors il existe une paire d'états $(s_1, s_2) \in \Delta$ telle que s_1 est non-bloquant pour $sobs$ et s_2 est non-bloquant pour $sobs.o$.*

Démonstration. Par contraposée, si pour toute paire d'états (s_1, s_2) telle que s_1 est bloquant pour une séquence d'observation *sobs* et s_2 est bloquant pour une continuation *sobs.o*. Cela signifie que quelle que soit l'état estimé par la fonction *estim*, il existera une impasse et donc le système n'est pas *estimable à état unique*. ■

Exemple 15 (Condition nécessaire sur les états non-bloquants). *Si on considère le système présenté en figure 6.2, on constate que la condition de la proposition 12 est respectée comme le montre la table des transitions de la figure 6.3. Le système est donc estimable à état unique : il suffit d'estimer l'état s_1 lorsque l'observation b est reçue depuis l'état initial pour ainsi éviter l'état bloquant s_2.*

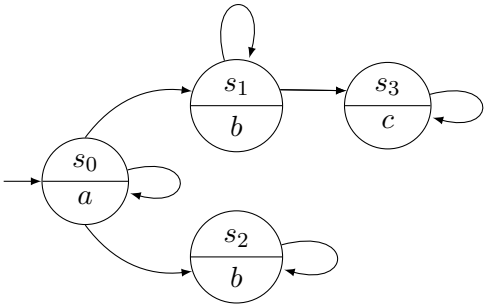


FIGURE 6.2 – Un SED estimable à état unique

sobs	*sobs.o*	$(s, s') \in \Delta$
a	$a.b$	$(s_0, s_1) \in \Delta$
ab	$ab.c$	$(s_1, s_3) \in \Delta$
ab	$ab.b$	$(s_1, s_1) \in \Delta$
..	..	...

FIGURE 6.3 – Tableau des transitions pour des séquences d'observations et leur continuation pour le système de la figure 6.2.

Il s'avère cependant que cette condition n'est pas suffisante pour garantir l'estimabilité à état unique. La figure 6.4 propose un contre-exemple dans lequel chaque transition est non-bloquante mais il est pourtant impossible de construire une fonction d'estimation pour ce système.

Exemple 16 (Condition insuffisante sur les états non-bloquants). *On considère maintenant le système présenté en figure 6.4, on constate que la condition de la proposition 12 est respectée comme le montre la table des transitions de la figure 6.5 mais le système n'est pas estimable à état unique. Il est en effet possible de vérifier la condition suffisante en trouvant des paires d'états non-bloquants pour chaque séquence d'observations et une de ses continuations. Cependant, produire une séquence d'états estimés pour la séquence d'observations (a, b, c, d, c) est impossible dans notre cadre d'estimation à état unique puisque en partant de l'état initial s_0, il faudrait d'abord emprunter les états s_1 et s_3 car ceux-ci sont non-bloquants et on serait ensuite obligé d'emprunter l'état s_5 qui lui est bloquant. Le système présenté en figure 6.4 n'est donc pas estimable à état unique.*

Cette condition ne constitue pas une condition nécessaire et suffisante pour l'*estimabilité à état unique*, on l'utilise cependant dans nos algorithme car celle-ci permet d'écarter les

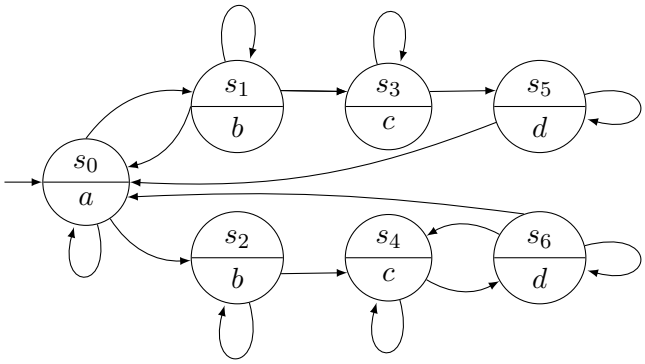


FIGURE 6.4 – Un SED dans lequel toutes les transitions sont non-bloquantes, mais qui n'est pas estimable à état unique

$sobs$	$sobs.o$	$(s, s') \in \Delta$
a	$a.b$	$(s_0, s_1) \in \Delta$
ab	$ab.c$	$(s_1, s_3) \in \Delta$
abc	$abc.d$	$(s_4, s_6) \in \Delta$
$abcd$	$abcd.c$	$(s_6, s_0) \in \Delta$
..	..	...

FIGURE 6.5 – Tableau des transitions pour des séquences d'observations et leur continuation pour le système de la figure 6.4.

états bloquants lors de la construction de l'estimateur. Lors de la recherche d'un estimateur on explorera donc uniquement des états non bloquants.

6.2 Approche par équivalence des langages

La condition nécessaire et suffisante pour le problème de l'estimabilité à état unique repose sur l'équivalence entre le langage observable du système et le langage accepté par l'estimateur. On propose dans cette section un algorithme permettant de réaliser la vérification de cette équivalence.

6.2.1 Structure générale de l'algorithme

L'approche basée langage consiste à construire récursivement des fonctions d'estimation partiellement définies et les tester.

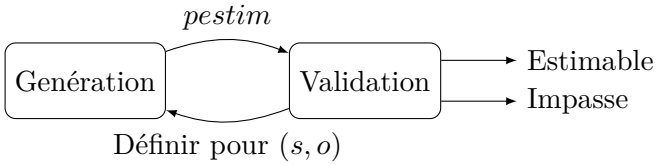


FIGURE 6.6 – Structure de l'algorithme de l'approche basée langages.

Afin de construire récursivement une fonction d'estimation, l'algorithme est initialisé avec une fonction d'estimation partielle vide. On tente ensuite de prolonger cette fonc-

tion d'estimation partielle jusqu'à couvrir l'ensemble des états atteignables du système. Le prolongement d'une fonction d'estimation partielle est définie comme suit :

Définition 46 (Prolongement d'une fonction d'estimation). *Soient estim et pestim deux fonctions d'estimation de* $(S \times O)$ *vers* S*. On dit que estim* prolonge *pestim si et seulement si pour tout couple* (s, o) *de* $S \times O$ *tel que* $pestim(s, o)$ *est définie, alors* $estim(s, o)$ *est également définie et* $pestim(s, o) = estim(s, o)$

Proposition 13 (Impasse pour une fonction d'estimation partielle). *Soient sobs une séquence d'observations de* $\mathcal{L}_{obs}(\Delta)$*, pestim et estim deux fonctions d'estimation de* $(S \times O)$ *vers* S *telles que estim prolonge pestim. Si sobs est une impasse pour pestim, alors sobs est également une impasse pour estim.*

Démonstration. Soit *sobs* une impasse pour *pestim* et *estim* une fonction qui prolonge *pestim*. Comme *pestim* et *estim* sont égales sur les couples (s, o) pour lesquels *pestim* est définie alors si *sobs* est une impasse pour *pestim*, *estim* va prendre les mêmes valeurs d'estimation et *sobs* sera également une impasse pour *estim*. ■

Afin de construire récursivement et valider des fonctions d'estimations partielles, l'algorithme est structuré à partir de deux composantes : la première produit des fonctions d'estimation partiellement définies et la seconde vérifie si une fonction d'estimation partielle donnée satisfait la proposition 10. La structure de l'algorithme est illustré en figure 6.6.

La composante de génération produit récursivement des fonctions d'estimation partielles *pestim* et les transmet à la composante de validation. La composante de validation a trois types de retours :

- « Estimable » signifie que le système peut être estimé complètement avec la fonction *pestim* courante
- « Impasse » signifie qu'il existe un scénario d'impasse pour la la fonction *pestim* courante.
- « Définir pour (s, o) » signifie que *pestim* doit être prolongée pour inclure une estimation pour (s, o).

Selon le résultat de la validation, la composante de génération peut terminer ou produire récursivement d'autres fonctions d'estimation partielles à tester. Le fonctionnement de chacune des deux composante est ensuite détaillé.

6.2.2 Borne de complexité

La vérification de l'équivalence des langage est un problème aussi appelé problème de l'équivalence de traces. Il a été prouvé que ce problème appartient à la classe de complexité P-SPACE [Huttel and Shukla, 1996]. Cela signifie qu'un algorithme capable de le résoudre nécessite un espace mémoire de taille polynomiale. La vérification par équivalence des langages fournit donc une borne de complexité pour le problème de l'estimabilité à état unique.

6.2.3 Fonctionnement de la composante de validation

La composante de validation contient une fonction CheckEstim qui vérifie si le langage accepté par une fonction d'estimation partielle est égal au langage des observations du système. L'algorithme est basé sur une légère modification de l'algorithme classique pour tester l'égalité des langages réguliers [Cassandras and Lafortune, 2009] afin de tenir compte des cas où plusieurs candidats à l'estimation existent et pour lesquelles la fonction d'estimation partielle est indéfinie.

L'approche consiste à simuler l'exécution de l'estimateur et du système, tout en s'assurant qu'ils sont synchronisés sur les mêmes séquences d'observations. Puisque pour chaque

séquence d'observations *sobs*, il existe (au plus) une séquence d'estimations unique, nous n'avons besoin de garder trace que d'un seul état dans l'estimateur. Ainsi, l'état de l'estimateur atteint via une séquence d'observations *sobs* est associé à l'ensemble des états systèmes $reach(sobs)$.

L'algorithme explore récursivement des paires $(estSt, sysSts)$ où $sysSts = reach(sobs)$ pour une $sobs \in \mathcal{L}_{obs}(\Delta)$, et où $estSt \in sysSts$. Pour s'assurer que l'algorithme termine, une variable globale « *visited* » stocke les paires qui ont déjà été explorées. L'algorithme recherche les séquences *sobs* pour lesquelles l'estimateur n'est pas défini, de sorte que la condition de terminaison n'est remplie que si une telle séquence n'existe pas. Dans ce cas, les langages sont égaux, et nous retournons « Estimable » (lignes 12 à 13).

À chaque itération, CheckEstim calcule les observations que le système peut produire (ligne 16). Ensuite, pour chaque observation produite, il appelle une fonction `NextEstStates`$(estSt, o)$ (ligne 19) définie comme suit. Si *pestim* est définie pour $(estSt, o)$, celle-ci retourne $\{pestim(estSt, o)\}$ sinon $cands(estSt, o)$. L'algorithme teste ensuite le nombre d'état possibles pour l'estimateur :

- Si $estNxts = \emptyset$ (ligne 22) : cela signifie que si *sobs* est la séquence d'observations courante (dans l'appel récursif), $sobs.o$ est une impasse,« Impasse » est retourné.
- Si $estNxts = \{estNxt\}$ (ligne 25) : cela signifie que pour la paire d'état courante $(estSt, o)$, soit il n'y a qu'un unique successeur, soit *pestim* est définie. Dans ce cas la recherche continue avec des appels récursifs.
- Si $|estNxts| > 1$ (ligne 33) : cela signifie qu'il y a plusieurs candidats à l'estimation, et que *pestim* n'est pas définie pour $(estSt, o)$. « Définir pour $(estSt, o)$ » est retourné, afin que la composante de génération produise des fonctions d'estimation définies pour cette paire.

6.2.4 Fonctionnement de la composante de génération

La composante de génération passe par une fonction GenEstim décrite dans l'algorithme 5 qui lance le processus de vérification et appelle la composante de validation. Lors de la première itération, la composante de génération produit la fonction d'estimation partielle vide (c'est-à-dire définie sur $\emptyset$), et l'envoie à la composante de validation.

- Si la composante de validation retourne « Estimable », cela signifie que la fonction d'estimation partielle courante *pestim* définit un estimateur qui accepte $\mathcal{L}_{obs}(\Delta)$, et que les paires d'états pour lesquelles *pestim* n'est pas définie ne sont pas nécessaires pour l'estimation. De ce fait, toute extension de *pestim* satisfait la proposition 10, et le système est estimable.
- Si la composante de validation retourne « Impasse », cela signifie qu'il existe un chemin sans issue pour *pestim*. Par la proposition 13, aucune extension de cette fonction ne peut satisfaire la proposition 10. La récursion se termine.
  Si la composante de validation retourne « Définir pour (s, o) », cela signifie qu'il existe une paire (s, o) telle que s est atteignable, $cands(s, o)$ contient plusieurs candidats, et *pestim* n'est pas définie pour (s, o). Dans ce cas, nous devons vérifier s'il existe une option d'estimation pour (s, o) qui satisfait la proposition 10, et les extensions de *pestim* sont alors générées récursivement.

L'algorithme explore récursivement toutes les options d'estimation, Ainsi s'il termine sans qu'aucune fonction d'estimation satisfaisant la proposition 10 n'ait été trouvée, alors le système n'est pas estimable.

6.3 Approche par simulation entre automates

L'approche basée langage fournit une vérification de la propriété d'estimabilité à état unique à partir de l'équivalence des langages des automates du système et de l'estimateur.

Algorithm 4 Composante de validation : fonction CheckEstim

Entrees
  $\Sigma = (S, \Delta, s_0)$: un SED
  estim : une fonction d'estimation partielle
  $estSt \in S$: état de l'estimateur
  $sysSts \subseteq S$: états du système
Sorties
  « Estimable » , « Impasse » ou « Définir pour (s, o) »
Global
  $visited \in S \times 2^S \leftarrow \emptyset$
function CheckEstim(Σ, *estim*, *estSt*, *sysSts*)
// Marquage des couples (état estimateur, états système)
  if $(estSt, sysSts) \in visited$ **then**
    return Estimable
  $visited \leftarrow visited \cup (estSt, sysSts)$
// Calcul de toutes les observations pouvant suivre s'
  $nextObs \leftarrow \{obs(s') | \exists s \in sysSts, (s, s') \in \Delta\}$
// Calcul des états suivants dans l'estimateur et dans le système
  for o **in** *nextObs* **do**
    $estNxts \leftarrow$ NextEstStates($estSt, o$)
    $sysNxts \leftarrow \{s' \mid obs(s') = o \wedge \exists s \in sysSts, (s, s') \in \Delta\}$
// Pas de successeurs dans l'estimateur
    if $estNxts = \emptyset$ **then**
      return Impasse
// Un seul successeur dans l'estimateur
    else if $estNxts = \{estNxt\}$ **then**
      $rec \leftarrow$ CheckEstim(Σ, *estim*, *estNxt*, *sysNxts*)
      switch *rec* **:**
        case Impasse **: return** Impasse
        case Définir pour (s, o) **:**
          return Définir pour (s, o)
        case Estimable **:** continue
// Plusieurs successeurs possibles dans l'estimateur
    else
      return Définir pour $(estSt, o)$
  end for
  return Estimable

Cependant, la vérification d'équivalence des langages est coûteuse. Dans les formalismes utilisés jusqu'à présent pour la définition et l'estimation d'un SED, l'état suivant dépend directement de l'état précédent et non de la séquence complète d'états précédents. Il est donc possible de vérifier l'estimabilité à état unique en utilisant la relation de simulation entre automates. Dans cette partie, on représente maintenant les systèmes à estimer comme une machine de Moore (S, s_0, Δ, Γ) avec un ensemble de transitions Δ de $S \times S$, s_0 l'état initial du système, et une fonction $obs : S \to O$ qui permet d'obtenir l'observation associée à un état. Comme décrit dans l'article [Roussel et al., 2020], on montre que l'on peut aussi représenter la fonction *estim* comme une machine de Moore dont l'ensemble des états est un sous-ensemble des états du système.

Algorithm 5 Composante de génération : fonction GenEstim

```
1: Entrees
2:    Σ = (S, Δ, s_0) : un SED
3: Sorties
4:    pestim : une fonction d'estimation partielle
5: function GenEstim(Σ, pestim)
6:    switch CheckEstim(Σ, pestim, s_0, {s_0}) :
7:       case Estimable : return true
8:       case Impasse : return false
9:       case Définir pour (s,o) :
10:         for c in cands(s, o) do
11:            extension ← pestim ∪ ((s, o), c)
12:            if GenEstim(Σ, extension) then
13:               return true
14:         end for
15:         return false
```

6.3.1 Relation de simulation entre automates

Définition 47 (Relation de simulation). *Étant donnés deux systèmes à évènements discrets $M = (S, s_0, \Delta, obs)$ et $M' = (S', s'_0, \Delta', obs')$, on dit que M' simule M par la relation de simulation $R \subseteq S \times S'$ si et seulement si :*

- $(s_0, s'_0) \in R$
- $\forall (s, s') \in R, \forall t \in S$ *tel que* $(s, t) \in \Delta$, *alors* $\exists t' \in S'$ *tel que* $(s', t') \in \Delta'$, $obs(t) = obs(t')$ *et* $(t, t') \in R$

Dans le problème de l'équivalence de langages dans le domaine de la théorie des langages et des automates, certains états sont considérés comme terminaux, d'autres non. Dans notre cas, aucun état n'est terminal et toutes les séquences d'observations cohérente sont des mots du langage observable du système. Par définition, l'espace d'état atteignable par l'estimateur (définit par une fonction d'estimation) est inclus dans l'espace d'état atteignables du système. En effet, avec l'approche basée langage, on cherche à construire une fonction d'estimation définie sur l'ensemble des couples (s, o) atteignables dans le système. L'estimateur d'un système à évènements discrets est donc un sous-automate du système. On peut montrer que si un sous automate d'un système à évènements discrets simule celui-ci, alors il y a équivalence entre leurs langages observables.

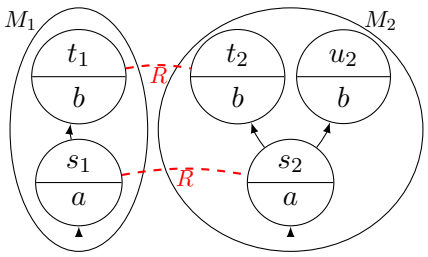


Figure 6.7 – Relation de simulation entre deux automates

Proposition 14. *Soient 2 systèmes à évènements discrets $M = (S, s_0, \Delta, obs)$ et $M' = (S', s'_0, \Delta', obs')$ tels que :*

- $S' \subseteq S$,
- $s'_0 = s_0$
- $\Delta' \subseteq \Delta$

Si M' simule M par la relation $R \subseteq S \times S'$ alors $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(M')$

Démonstration. Montrons que $\mathcal{L}_{obs}(M') \subseteq \mathcal{L}_{obs}(M)$. Soit $obs \in \mathcal{L}_{obs}(M')$. Il existe $seq' = (s'_0, ..., s'_1) \in \mathcal{L}_{obs}(M')$ tel que $obs = \text{obs}(seq), \forall i \in [0, n-1], (s'_i, s'_{i+1})$ appartient à Δ' donc appartient à Δ. Comme $s'_0 = s_0$ alors seq' appartient à $\mathcal{L}(M)$. Donc obs(seq') appartient à $\mathcal{L}_{obs}(M)$. Montrons que $\mathcal{L}_{obs}(M) \subseteq \mathcal{L}_{obs}(M')$. Soit $obs \in \mathcal{L}_{obs}(M)$. Il existe $seq = (s_0, ..., s_n) \in \mathcal{L}_(M)$ tel que $obs = \text{obs}(seq)$. On montre par récurence sur la taille de obs que $obs \in \mathcal{L}_{obs}(M')$.

- On a $s_0 = s'_0$ donc $\text{obs}(s_0) = \text{obs}(s'_0)$ et $(s_0, s'_0) \in R$.
- Soit $k \leq n$. Supposons qu'il existe $(s'_0, .., s'_k) \in \mathcal{L}_(M')$ et $(s_0, ..., s_k) \in \mathcal{L}_(M)$ tels que $\text{obs}((s'_0, ..., s'_k)) = \text{obs}((s_0, ..., s_k))$ et $(s_k, s'_k) \in R$. Comme $(s_k, s_{k+1}) \in \Delta$ et $(s_k, s'_k) \in R$, alors d'après la définition de la simulation il existe $s'_{k+1} \in S'$ tel que $(s'_k, s'_{k+1}) \in \Delta'$, $\text{obs}(s_{k+1} = \text{obs}(s'_{k+1}$ et $(s_{k+1}, s'_{k+1}) \in R$. Il existe donc $(s'_0, .., s'_k, s'_{k+1}) \in \mathcal{L}(M')$ et $(s_0, .., s_k, s_{k+1}) \in \mathcal{L}(M)$ tels que $\text{obs}((s'_0, .., s'_k, s'_{k+1})) = \text{obs}((s_0, .., s_k, s_{k+1}))$ et $s_{k+1}, s'_{k+1}) \in R$

Donc on a bien $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(M')$ ■

En plus d'être un sous-automate du système, avec l'approche d'estimation à état unique, l'estimateur a la particularité d'être déterministe sur les observations. On définit dans un premier temps les systèmes déterministes sur les observations qui représenterons les estimateurs.

Définition 48. *Système déterministe sur les observations Un système $M = (S, s_0, \Delta, obs)$ est déterministe sur les observations si et seulement si : $\forall s \in S, \forall (t, t') \in S$ tels que $(s, t) \in \Delta$ et $(s, t') \in \Delta$, si obs(t) = obs(t') alors $t = t'$*

Pour un système déterministe, chaque séquence d'observations correspond à une unique séquence d'états dans ce système.

Proposition 15. *Soit $M = (S, s_0, \Delta, obs)$ un SED déterministe sur les observations. Pour toute séquence d'observation obs de $\mathcal{L}_{obs}(M)$, il existe une unique séquence seq dans $\mathcal{L}(M)$ telle que obs(seq) = obs*

Démonstration. Soit $M = (S, s_0\ \Delta, obs)$ un SED déterministe sur les observations. Soit obs une séquence d'observation de $\mathcal{L}_{obs}(M)$ de longueur n, et soient seq et seq' deux séquences d'états de $\mathcal{L}(M)$ telles que $\texttt{obs}(seq) = \texttt{obs}(seq') = obs$. Montrons par récurrence que $\forall i \in [0..n]$, $seq[i] = seq[i]$.

- Pour $i = 0$, $seq[0] = seq'[0] = s_0$ car toute séquence de $\mathcal{L}(M)$ est préfixée par s_0.
- Soit k tel que $1 \leq k \leq n$. On suppose que pour tout $i \leq k$, $seq[i] = seq'[i]$. On a $(seq[k], seq[k+1]) \in \Delta$, $(seq'[k], seq'[k+1]) \in \Delta$, $\text{obs}(seq[k+1]) = \text{obs}(seq[k+1]) = obs(k+1)$. Donc d'après la définition 48, on a $seq[k+1] = seq'[k+1]$.

■

On montre ensuite que si il y a équivalence des langages observables entre un système et un sous automate de celui ci déterministe, alors il existe une relation de simulation entre les deux.

Proposition 16. *Soient 2 systèmes à évènements discrets $M = (S, s_0, \Delta, obs)$ et $M' = (S', s'_0, \Delta', obs')$ tels que :*

- *$S' \subseteq S$,*
- *$s'_0 = s_0$,*
- *$\Delta' \subseteq \Delta$,*
- *M' est déterministe.*

Si $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(M')$ alors il existe une relation $R \subseteq S \times S'$ telle que M' simule M par R

Démonstration. On construit d'abord R. Pour tout $s \in S$ et $s' \in S'$,$(s, s') \in R$ si et seulement s'il existe $obs \in \mathcal{L}_{obs}(M), seq \in \mathcal{L}_{(}M), seq' \in \mathcal{L}_{(}M')$ tels que $\texttt{obs}(seq) = \texttt{obs}(seq') = obs$, $last(seq) = s$ et $last(seq') = s'$ Soit $(s, s') \in R$. Il existe $obs \in \mathcal{L}_{obs}(M)$,$seq \in \mathcal{L}_{(}M), seq' \in \mathcal{L}_{(}M')$ tels que $\texttt{obs}(seq) = \texttt{obs}(seq') = obs$, $last(seq) = s$ et $last(seq') = s'$. On note k la taille de obs. Soit $t \in S$ tel que $(s, t) \in \Delta$. $seq.t$ appartient à $\mathcal{L}_{(}M)$ donc $obsT = obs.\texttt{obs}(t)$ appartient à $\mathcal{L}_{obs}(M)$. Comme $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(M')$, on sait qu'il existe $seq'' \in \mathcal{L}_{(}M')$ telle que $\texttt{obs}(seq'') = obsT$. Comme M' est déterministe, $\forall i \leq k, seq'[i] = seq''[i]$. On a notamment $seq''[i] = s'$. On note t' l'état $seq''[k+1]$. Comme $seq'' \in \mathcal{L}_{(}M')$, on a $(s', t') \in \Delta$. Comme $obsT \in \mathcal{L}_{obs}(M)$, $seq \in \mathcal{L}_{(}M)$, $seq'' \in \mathcal{L}_{(}M')$ avec $\texttt{obs}(seq.t) = \texttt{obs}(seq'') = obsT$, $last(seq.t) = t$ et $last(seq'') = t'$, on a $(t, t') \in R$ ■

À partir des propositions 14 et 16, on peut conclure que l'égalité des langages observables et l'existence d'une relation de simulation sont équivalentes.

Proposition 17. *Soient 2 systèmes à évènements discrets M = (S, s_0, Δ, obs) et M' = (S', s'_0, Δ', obs') tels que :*
- *$S' \subseteq S$,*
- *$s'_0 = s_0$,*
- *$\Delta' \subseteq \Delta$,*
- *M' est déterministe.*

$\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(M')$ si et seulement si il existe une relation $R \subseteq S \times S'$ telle que M' simule M par R.

Démonstration. On prouve trivialement la proposition 17 à partir des propositions 14 et 16. ■

6.3.2 Encodage de la vérification d'un estimateur

On a donc montré que la relation de simulation entre automate pouvait être utilisée pour résoudre le problème de l'estimabilité à état unique. On s'intéresse dans un premier temps à la possibilité de vérifier si il existe une relation de simulation pour un système à évènements discrets et un estimateur donnés. On propose un encodage basé sur la proposition 17. Pour un système $M = (S, s_0, \Delta, obs)$ et son estimateur déterministe sur les observations $M = (S', s'_0, \Delta', obs')$, on introduit les variables booléennes suivantes. Pour tout $s \in reach(M), s' \in reach(M')$, on créé la variable $r_{s,s'}$ qui est vraie si et seulement si (s, s') appartient à la relation de simulation. On créé ensuite les clauses suivantes :
- r_{s_0,s'_0}
- $\forall s \in S, \forall s' \in S', \forall t \in S$ tel que $(s, t) \in \Delta$, $r_{s,s'} \rightarrow \bigvee_{t' \in S', (s',t') \in \Delta, obs(t) = obs(t')} r_{t,t'}$

Ces contraintes peuvent être encodées et vérifiées grâce à un solver SAT. Si on trouve une affectation possible sur les variables $r_{s,s'}$, alors chaque variables à vrai indique que le couple (s, s') appartient à la relation de simulation trouvée. On peut donc vérifier avec cette méthode qu'il existe (ou non) une relation de simulation entre un système et son estimateur.

6.3.3 Borne de complexité

L'utilisation de la relation de simulation entre automate permet de raffiner la complexité algorithmique théorique du problème de l'*estimabilité à état unique*. En effet, vérifier si un système en simule un autre s'effectue en temps polynomial. On peut donc vérifier qu'un estimateur est solution au problème de l'estimabilité dans un temps polynomial ce qui nous permet de statuer que le problème *estimabilité à état unique* appartient à la classe de complexité NP (voir chapitre 2).

6.3.4 Encodage de la recherche d'un estimateur

On propose maintenant un encodage permettant à partir d'un système M, de trouver une relation de simulation et un estimateur associé répondant au problème d'estimabilité à état unique. À partir d'un système à évènements discrets $M = (S,\ s_0,\ \Delta,\ obs)$, on créé les variables de décision suivantes pour représenter l'estimateur $M' = (S',\ s'_0,\ \Delta',\ obs')$ recherché et la relation de simulation entre les deux :

— $\forall s \in S$, V_s : vraie si et seulement si $s \in S'$

— $\forall (s, s') \in \Delta$, $E_{s,s'}$: vraie si et seulement si $(s, s') \in \Delta'$

— $\forall (s, s') \in S^2$ avec $\text{obs}(s) = \text{obs}(s')$, $R_{s,s'}$ vraie si et seulement si s' simule s.

On encode ensuite la recherche d'une solution avec les contraintes suivantes :

$$\forall (s,t) \in \Delta,\ E_{s,t} \rightarrow V_s \wedge V_t \tag{6.1}$$

$$\forall s' \in S,\ V_{s'} \leftrightarrow reachable(\widehat{M'}, s_0, s') \tag{6.2}$$

$$\forall s \in S, \forall o \in nextObs(s),\ \sum_{t \in S | obs(t) = o} E_{s,t} \leq 1 \tag{6.3}$$

$$\forall s \in S, \forall o \in nextObs(s),\ V_s \rightarrow \bigvee_{t \in S | obs(t) = o} E_{s,t} \tag{6.4}$$

$$R_{s_0,s_0} \tag{6.5}$$

$$\forall s \in S,\ \bigvee_{s' \in S, obs(s) = obs(z')} R_{s,s'} \tag{6.6}$$

$$\forall (s, s') \in S^2 \text{ s.t. } obs(s) = obs(s'), \forall t \in S \ s.t.\ (s,t) \in \Delta,$$
$$R_{s,s'} \rightarrow \bigvee_{(s',t') \in \Delta, o(t) = obs(t')} (R_{t,t'} \wedge E_{s',t'}) \tag{6.7}$$

$$\forall s' \in S,\ V_{s'} \leftrightarrow \bigvee_{s \in S | o(s) = o(s')} R_{s,s'} \tag{6.8}$$

La contrainte 6.1 indique qu'une transition de l'estimateur relie deux états de celui-ci. Chaque état de l'estimateur doit être atteignable depuis l'état initial 6.2. Les contraintes 6.3 et 6.4 indique que pour chaque couple (s, o) il y a exactement 1 état estimé dans l'estimateur. Cela signifie non seulement que l'estimateur est déterministe sur les observations mais aussi que celui-ci ne rencontre pas d'impasse. Les contraintes 6.5 et 6.7 reprennent la définition de la relation de simulation entre automates et indiquent que si un état en simule un autre, alors il appartient à l'estimateur. Enfin, la contrainte 6.6 contraint chaque état du système à être simulé et la contrainte 6.8 indique que chaque état de l'estimateur apparaît dans la relation de simulation (ce qui permet de rendre la relation de simulation trouvée minimale).

6.3.5 Ajout de contraintes de correction

On est capable de vérifier grâce à la simulation entre automate qu'un estimateur est solution au problème de l'estimabilité à état unique. Cependant, les estimateurs sans impasses ne sont pas tous intéressants. Par exemple, un estimateur qui reste en permanence dans l'état initial du système n'est pas très intéressant du point de vue du diagnostic. On s'intéresse donc à des estimateurs estimant le plus correctement possible l'état du système. Plus généralement, la correction est une propriété qui indique que l'état de l'estimateur correspond à l'état actuel du système. L'estimabilité à état unique fourni une forme très faible de correction, l'estimation est toujours cohérente avec les observations reçue, la dynamique du système est donc toujours respectée. On peut toutefois s'intéresser à différentes formes de

correction supplémentaires. Suivant le contexte, une correction totale ou encore des estimateurs plus ou moins optimistes peuvent êtres requis [Bouziat et al., 2018, Couto et al., 2020]. On définit dans cette section deux formes de correction paramétrables par l'utilisateur. On modélise ici la correction sous forme de contraintes mais aussi à partir de critères d'optimisation afin de générer des estimateurs intéressants. Les contraintes permettent de restreindre les états de l'estimateur par rapport à ceux du système tandis que les critères d'optimisation intègre une notion de coût pour chaque estimateur possible. On définit la première forme de correction permettant de contraindre les états estimés. Pour cela, on peut forcer certains états possible de l'estimateur à apparaître dans la relation de simulation.

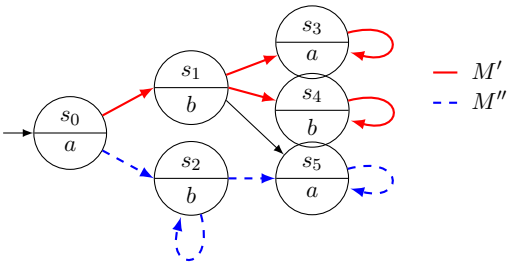


FIGURE 6.8 – Un système M, avec un estimateur possible M' en rouge, et un autre M'' en bleu.

Définition 49 ((ϕ_1, ϕ_2)-correction). *Soit M un système à évènements discrets, soient $\phi_1 \subseteq S$ et $\phi_2 \subseteq S$ deux ensembles d'états, M' un estimateur sans impasse pour M, et R la relation de simulation associée. M' est dit (ϕ_1, ϕ_2)-correct si et seulement si lorsque le système est dans un état de ϕ_1, alors l'estimateur est dans un état de ϕ_2, c.a.d. $\forall (s, s') \in R$, if $s \in \phi_1$ alors $s' \in \phi_2$.*

Avec de telles contraintes, on peut par exemple forcer l'estimation d'une faute critique lorsque cela est cohérent avec les observations. Cela rend l'estimateur correct pour cette faute.

Exemple 17. *On considère le système M présenté en Figure 6.8. Soit f_1 une faute présente dans les états $\phi^{f_1} = \{s_1, s_3, s_4\}$ et f_2 une faute seulement présente dans $\phi^{f_2} = \{s_5\}$. L'estimateur M' est (ϕ^{f_1}, ϕ^{f_1})-correct car tous les états dans ϕ^{f_1} sont estimés par les états dans ϕ^{f_1}. Cependant M' n'est pas (ϕ^{f_2}, ϕ^{f_2})-correct car s_5 est estimé par s_4 et ce dernier n'appartient pas à ϕ^{f_2}.*

Cependant, la (ϕ_1, ϕ_2)-correction réduit l'espace des estimateurs candidats et peut être dans certains cas trop restrictive. Par exemple, dans l'exemple 17, il n'existe aucun estimateur satisfaisant la (ϕ^{f_1}, ϕ^{f_1}) et la (ϕ^{f_2}, ϕ^{f_2})-correction. On peut donc s'intéresser à une autre forme de correction. Par exemple énumérer tous les estimateurs possibles et choisir le meilleur. Cela pourrait être laborieux mais une autre approche consiste à définir un critère d'optimisation dans la recherche des estimateurs. Par exemple, la(ϕ_1, ϕ_2)-correction pourrait être satisfaite (plus faiblement) « le plus souvent possible ». On introduit donc une fonction de coût pour les estimateurs afin de modéliser ce critère de correction. On définit pour cela un ordre sur les estimateurs en fonction d'une métrique définie par l'utilisateur représentant la distance entre un système et son estimateur.

Définition 50 (Coût de correction). *Soient M un système à évènements discrets, M' un estimateur sans impasses pour M, et R leur relation de simulation. Soit $cost : S^2 \to \mathbb{N}$ une fonction de coût telle que $cost(s, s')$ représente à quel point il est correcte d'estimer s par s'. Le coût de correction pour M' est $cost(M') = \sum_{(s,s') \in R} cost(s, s')$*

Définition 51 (Distance de correction). *Soit M un système à évènements discrets, M' et M'' deux estimateurs sans impasse pour M, et $cost$ une fonction de côut. M' est plus correcte que M'' selon $cost$ si et seulement si $cost(M') \leq cost(M'')$.*

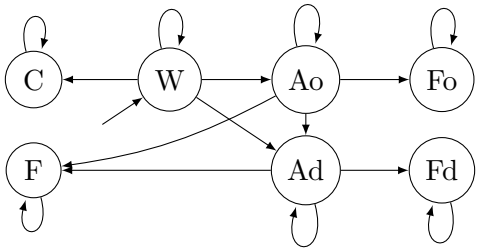


FIGURE 6.9 – Le workflow des états des actions qui peuvent être (W)aiting, (C)ancelled, (A)ctive (o)k, (F)inished (o)k, (F)ailed, (A)ctive (d)elayed, ou (F)inished (d)elayed.

On peut définir cette fonction de coût de différentes manières pour des paires d'états. Par exemple vérifier qu'un état de l'estimateur correspond ou non à l'état réel du système. Une telle fonction $c_=$ serait donc définit comme suit : $\forall (s, s') \in S^2, c_=(s, s') = 0$ si $s = s'$, sinon 1. On peut aussi considérer une fonction basée sur la (ϕ_1, ϕ_2)-correction : $\forall (s, s') \in S^2, c_{\phi_1,\phi_2}(s, s') = 0$ si $s \in \phi_1$ et $s' \in \phi_2$, sinon 1.

Si les états sont représentés par des variables booléennes, la distance de Hamming [Hamming, 1950] fournit une fonction de coût adéquate. Des poids de coût pondérés ou une agrégation lexicographique des coût peuvent aussi être utilisés pour considérer différentes criticités de fautes par exemple.

Exemple 18. *Soit M le système à évènements discrets présenté en Figure 6.8. On définit l'estimateur sans impasse M''' similaire à M' excepté que $estim(s_1, a) = s_5$. On considère une fonction de coût c telle que $\forall i, c(s_i, s_i) = 0$, $c(s_1, s_2) = c(s_2, s_1) = 1$, $c(s_3, s_5) = 1$ et $c(s_5, s_3) = 2$. Ce coût indique qu'il est préférable d'estimer que le système est dans l'état s_5 quand celui-ci est dans s_3 plutôt que l'inverse. On a $c(M') = 3$ et $c(M''') = 2$. Cela signifie que M''' est plus correct que M' selon la fonction de coût c.*

6.4 Expérimentations

6.4.1 Expérimentations pour l'approche par l'équivalence des langages

Nous avons testé notre approche sur un exemple inspiré du langage de robotique autonome PLEXIL. [V.Verma et al., 2005]. Ce cadre est organisé autour du concept d'actions, qui ont un workflow hiérarchique complexe. Nous utilisons un workflow d'actions séquentielles simplifié décrit dans l'exemple 19.

Exemple 19. *Nous considérons un cadre robotique où les plans des robots sont représentés sous forme de séquence d'actions. La figure 6.9 illustre le workflow des actions. Nous considérons un robot avec un plan constitué de deux actions séquentielles : move et inspect, dont les états sont représentés par les variables mv et ins. L'état de santé du robot est décrit par trois variables booléennes $hnav$, $hsens$ et $hpow$ représentant respectivement si les fonctions de navigation, de capteur et d'alimentation électrique fonctionnent normalement. Une autre variable booléenne, $pert$, indique si le robot est sujet à des perturbations (terrain glissant, obstacle, vent) à chaque pas de temps. Nous notons $move = W$ pour exprimer que l'action move passe dans l'état W. Nous notons $Y(v)$ la valeur à l'étape précédente pour la variable $v \in \{mv, ins, hnav, hsens, hpow, pert\}$. Nous utilisons la fonction $start(v) = K$ qui signifie $Y(v) \neq K \wedge v = K$. Le comportement du système est décrit par l'automate pour*

chaque action, associé aux contraintes suivantes :

$$mv \in \{W, Ao, Ad\} \rightarrow ins = W \tag{6.9}$$
$$mv \in \{C, F\} \rightarrow ins = C \tag{6.10}$$
$$start(mv = Fo) \rightarrow ins = Ao \tag{6.11}$$
$$start(mv = Fd) \rightarrow ins = Ad \tag{6.12}$$
$$start(mv = Ad) \rightarrow (\neg hnav \vee \neg hpow \vee pert) \tag{6.13}$$
$$start(mv = F) \rightarrow \neg hpow \tag{6.14}$$
$$start(ins = Ad) \rightarrow (\neg hsens \vee \neg hpow \vee pert) \tag{6.15}$$
$$start(ins = F) \rightarrow \neg hpow \tag{6.16}$$
$$hnav \vee hsens \rightarrow hpow \tag{6.17}$$

L'action ins doit rester dans W lorsque l'action mv s'exécute (6.9)*. Si mv échoue ou est annulé, ins est annulé* (6.10)*. ins commence au moment où mv finit* (6.11)*,* (6.12)*. Un retard dans mv (resp. ins) peut s'expliquer par un problème de navigation (resp. capteur), d'alimentation, ou une perturbation* (6.13)*. (resp.* (6.15)*). Une défaillance dans mv ou ins ne peut s'expliquer que par un problème d'alimentation* (6.14)*,* (6.16)*. Un problème d'alimentation électrique se propage au capteur et à la navigation* (6.17)*.*

Les variables mv et ins sont observables, c'est-à-dire que l'état de chaque action est connu. Les variables hnav, hsens, hpow et pert sont estimées. L'ensemble d'états S est l'ensemble des évaluations pour toutes les variables, la fonction obs limite une évaluation aux variables mv et ins. Par exemple, l'état initial $(mv = W, ins = W, hnav, hsens, hpow, \overline{pert})$ *produit l'observation* $(mv = W, ins = W)$*.*

Notre algorithme est implémenté en Scala, et exécuté sur un processeur Intel® Xeon(R) W-2123, 3.60GHz 8 coeurs, avec une mémoire limitée à 4 Go. Le système tel que décrit dans l'exemple 19 est estimable, nous avons donc introduit quelques modifications pour le rendre non estimable. Nous avons fait en sorte que les actions produisent la même observation dans leurs états « Ao » et « Ad ». Nous avons aussi modélisé les systèmes d'exemple de [Sampath et al., 1995] (Valve Controller) et [Williams and Nayak, 1996] (Valve Driver).

Les résultats présentés dans la table 6.1 montrent que les systèmes non estimables sont détectés très rapidement. Ceci est dû à la vérification préliminaire sur les états bloquants qui capte les impasses plus tôt. Bien que les propositions 11 et 12 ne suffisent pas à assurer l'estimabilité, elles sont utiles dans une grande majorité des cas. Dans le cas des systèmes estimables, le temps de calcul est lié au nombre de fonctions d'estimation partielles testées. Notons qu'il peut être améliorer en tenant compte des besoins opérationnels pour limiter l'espace des fonctions d'estimation à examiner.

Modèle	États	Succ.	Estimable à état unique	Temps(s)
Exemple 19	112	13.7	oui	66
Exemple 19-v2	112	13.7	non	0.4
Valve Controller	209	15.5	non	0.4
Valve Driver	51	22.3	oui	56

TABLE 6.1 – Temps de calcul pour vérifier l'*estimabilité à état unique* avec l'équivalence des langages. La colonne « États » indique le nombre d'états du système, « Succ. » le nombre moyen de transitions pour chaque état, « Estimable à état unique » indique si le système est estimable, « Temps » le temps de calcul en secondes.

6.4.2 Expérimentations pour l'approche par la simulation d'automates

Afin de tester l'encodage de la simulation entre automates, on utilise différents systèmes issus de la littérature mais aussi des systèmes générés de manière aléatoire. La taille de

certains modèles étant importante, l'utilisation d'un serveur de calcul à été nécessaire. Pour toutes les expérimentations la mémoire alloué à été poussée à 64 Go. L'implémentation repose toujours sur l'utilisation de SCALA et les expérimentations ont été effectué sur des processeurs Intel® Xeon® CPU E5-2699 v3 @ 2.30GHz. On présente ci-dessous les différents modèles testés.

Exemples issus de la littérature.

- *valve controller* [Sampath et al., 1995] : le modèle utilisé dans l'article qui définit la diagnosticabilité d'un système. Ce modèle représente un système de ventilation d'air conditionné. Le système est équipé d'une valve qui peut être ouverte, fermée ou bloquée, d'une pompe et d'un contrôleur. Une pompe peut être touchée par une faute permanente dans les modes *on* et *off*. Chaque composants peut être dans 4 états possibles ; ce système n'est pas estimable à état unique.
- *valve driver* [Williams and Nayak, 1996] un modèle qui représente le comportement d'un réacteur de fusée équipé de plusieurs réservoirs. On y retrouves 4 modes, 6 commandes d'entrées et 3 sorties ; ce système est estimable à état unique.
- *baggage transfer* [Pencolé et al., 2018] Ce modèle représente un tapis d'aéroport sur lequel circulent des bagages. On y retrouve 2 aiguillages, 1 piston, 1 contrôleur et 4 capteurs. En modifiant ce modèle, on parvient à générer 218 systèmes ; tous les systèmes sont estimables à état unique.

Pour le modèle *baggage transfer*, aucun modèle de faute n'existait à l'origine, on modifie le modèle en supposant qu'une faute (non-observable) peut affecter chacun des composants.

Exemples générés aléatoirement Produire aléatoirement un système estimable à état unique est une tâche difficile à mesure qu'on souhaite obtenir un grand nombre d'états et de transitions. On génère donc un ensemble de systèmes étant le produit cartésien de plusieurs petits systèmes estimables à état unique. On génère donc des systèmes avec 5 états et 3 observations avec une probabilité de 50% de créer une transition entre chaque paire d'états. Le produit cartésien est ensuite effectué avec une probabilité de 90% de créer une transition entre chaque paires d'états possibles. On agrège ainsi entre 4 et 5 petits systèmes pour créer un système.

Chacun de ses systèmes ainsi créé n'est pas forcément intéressant pour les expérimentations. En effet dans la plupart des cas lorsque le système produit est estimable à état unique, l'estimateur correspond au système lui-même, on parle alors de système trivialement estimable à état unique.

On sélectionne pour les tests les systèmes intéressant suivant 3 critères :

1. Le système obtenu est estimable à état unique ;
2. L'estimateur parcourt moins de la moitié des états possibles du système. (on rejette ainsi les systèmes trivialement estimables).
3. Il existe un état s tel que le système n'admet pas d'estimateur $(\{s\}, S \setminus \{s\})$-correct.

Ensuite, pour chaque système ainsi généré, on génère aussi une version non-estimable à état unique de ce système en modifiant une des contraintes de Δ. Avec cette méthode de génération, on génère des systèmes de 10 à 1461 états. La figure 6.10 présente les expérimentations pour tous ces systèmes.

6.4.3 Résultats

Lors des expérimentations, la majeur partie du temps de calcul est liée à la génération des clauses SMT tandis que le temps de résolution par le solver *Monosat* est négligeable. On constate aussi qu'il n'y a que peu de différence de temps de calcul pour les systèmes estimables à état unique et ceux qui ne le sont pas. La figure 6.10 présente les résultats

expérimentaux pour l'approche par simulation d'automates et on constate que le temps de calcul croit de manière exponentielle par rapport au nombre d'états du système. Les exemples de la littérature *valve driver* et *valve controller* montrent des temps de calcul similaires à ceux des exemples générés de même taille. Pour les exemples du baggage transfer, les temps de calculs sont cependant plus importants, ceci est du à de nombreuses symétries présentes dans ces types de systèmes.

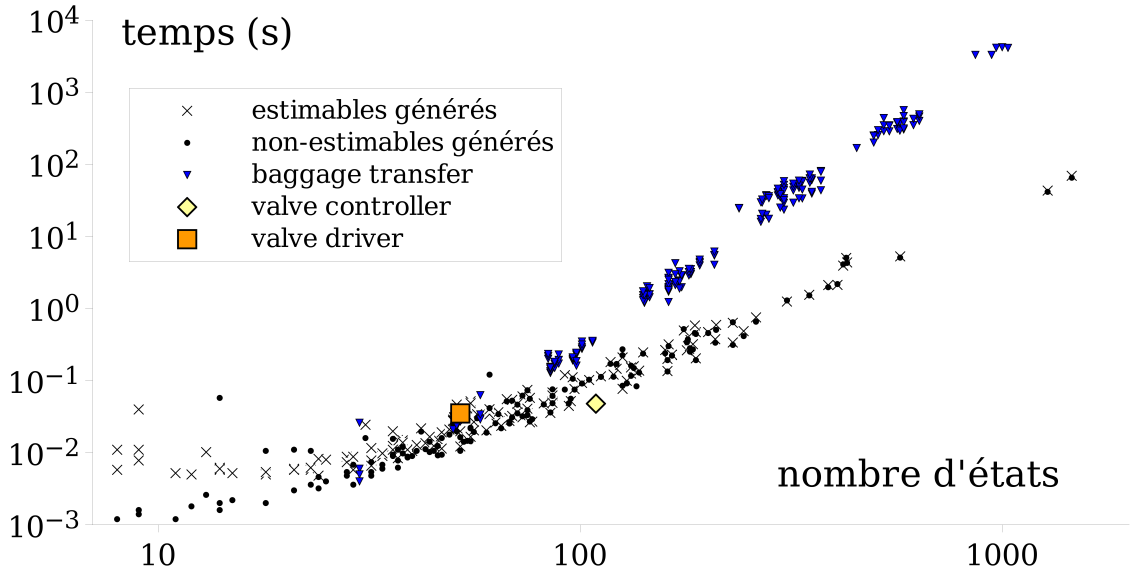


FIGURE 6.10 – Temps de calcul pour l'*estimabilité à état unique* avec l'approche par simulation d'automates. Notez que les échelles sont logarithmiques.

Pour les systèmes avec un nombre important d'états, il est possible de vérifier l'estimabilité en quelques heures. Ce phénomène n'est pas rédhibitoire puisqu'il s'agit d'une analyse hors ligne. On constate aussi que pour des systèmes de l'ordre de la centaine d'états, il existe une certaine variation pour l'ordre de grandeur du temps de calcul. Cela montre que des systèmes de taille similaire peuvent être plus ou moins difficiles à résoudre.

On peut comparer les résultats des deux approches sur les deux exemples de la littérature *valve driver* et *valve controller*. On constate que l'approche par simulation d'automate est plus efficace, on passe en effet d'un temps de vérification de l'ordre de la minute à un temps de l'ordre de la seconde. Contrairement aux travaux présentés précédemment, les deux approches offrent une vérification non-bornée pour le problème de l'estimabilité à état unique.

Chapitre 7

Conclusion

Les travaux de thèse présentés dans ce document s'intéressent au problème de l'estimation dans les systèmes à évènements discrets (SED). La première partie présente un état de l'art du domaine du diagnostic des SED. Le deuxième chapitre décrit les concepts de base pour de tels système et les approches de diagnostic de la littérature. Un formalisme d'estimation à état unique [Pralet et al., 2016], qui repose sur l'utilisation d'une théorie de préférences conditionnelles, est présenté dans le chapitre 3. Bien que ce formalisme ait des avantages par rapport aux autres approches de la littérature, celui-ci est sujet à certaines limites notamment le problème de l'impasse. Le problème de l'impasse affecte les modèles d'estimation à état unique lorsque pendant la phase de diagnostic, le système produit des observations incompatibles avec les précédentes estimations effectuées. Différentes solutions au problème de l'impasse sont explorées dans la deuxième partie de ce document.

7.1 Synthèse des contributions

Le chapitre 4 s'intéresse à la détection de scénario d'impasse à partir du modèle d'un système et d'une stratégie d'estimation à état unique. Pour cela, le phénomène de l'impasse est dans un premier temps formalisé ainsi que la construction d'une machine à états inspirée de la technique du Twin-Plant [Cimatti et al., 2003] sur laquelle une analyse d'atteignabilité permet de détecter les scénarios d'impasse. Deux approches sont ensuite proposées, la première repose sur l'utilisation d'*Electrum* un outil de model-checking, la deuxième sur l'utilisation de solver SAT. Chacune des deux approches permet la vérification de scénario d'impasse d'une taille donnée, permettant ainsi de valider ou non le bon fonctionnement de l'estimateur à état unique.

Dans le chapitre 5, on cherche à analyser un scénario d'impasse préalablement trouvé. On y fait l'hypothèse que les scénarios d'impasse sont dus à une mauvaise conception de la stratégie d'estimation à état unique. On cherche donc à blâmer certaines préférences dans la stratégie d'estimation à état unique en effectuant un méta-diagnostic des préférences. Pour cela, la notion de préférence relaxée est définie afin de désactiver certaines préférences, relâchant ainsi l'hypothèse d'un état unique estimé, mais permettant dans certains cas de retrouver la cohérence dans l'estimation et ainsi éliminer le scénario d'impasse. Tout comme le chapitre précédent, deux approches sont étudiées, l'une utilisant *Electrum*, l'autre un solver SAT. Des expérimentations sont effectuées à partir des scénarios d'impasse trouvés dans le chapitre 4.

On remarque cependant que pour certains systèmes, le méta-diagnostic des préférences n'aboutit pas. Il existe donc des systèmes à évènements discrets pour lesquels il n'est pas possible de réaliser l'estimation à état unique sans rencontrer de scénarios d'impasse. Au travers du chapitre 6, le problème de l'estimabilité à état unique est donc formalisé. On s'intéresse donc à la construction d'estimateurs à états unique sans impasse à partir des spécifications d'un système. La propriété d'*estimabilité à état unique* pour les SED est

proposée et une condition nécessaire et suffisante est définie. Cette condition repose sur l'équivalence des langages dans les automates et une première approche est développée autour de la vérification de cette équivalence. Ensuite, une deuxième approche utilise la relation de simulation entre automates. Cette approche repose sur l'utilisation du solver SMT *Monosat*. Il est montré que des contraintes de correction sur l'état estimé peuvent être ajoutées dans la construction de la relation de simulation. Chacune des deux approches est accompagnée d'expérimentations à la fois sur des modèles générés mais aussi sur quelques exemples de la littérature.

7.2 Perspectives

La recherche de scénario d'impasse dans le chapitre 4 s'effectue sur un horizon borné. On peut donc valider le bon fonctionnement d'un estimateur uniquement pour une certaine fenêtre temporelle. Il serait intéressant d'étendre cette méthode afin de pouvoir détecter les scénarios d'impasse de taille non bornée en identifiant par exemple certains types de cycles dans l'exécution.

Le méta-diagnostic des préférences est présenté dans le chapitre 5, permettant ainsi de blâmer les préférences pouvant être responsables d'un scénario d'impasse. Cependant la manière dont il faudrait modifier les préférences afin d'éliminer le scénario d'impasse n'est pas étudiée en détail. Il existe plusieurs manières de modifier le modèle de préférences conditionnelles ; on peut par exemple modifier la condition de la préférence ou encore son rang dans la séquence.

Une borne de complexité pour le problème de l'*estimabilité à état unique* est obtenue dans le chapitre 6. Une perspective de ce travail consiste à prouver que le problème de l'estimabilité à état unique est un problème NP-COMPLET. Une première condition pour cette preuve est la vérification d'une solution candidate au problème dans un temps polynomial : cela est possible grâce à l'approche par relation de simulation entre automates présentée dans la deuxième partie du chapitre. Ensuite, il faudrait parvenir à transformer par réduction polynomiale un problème connu de la classe NP-COMPLET vers le problème de l'*estimabilité à état unique*.

Lors de ces travaux, l'hypothèse d'un estimateur ne gardant qu'un seul état estimé à chaque étape discrète est posée. Cela permet notamment de pallier le problème de passage à l'échelle pour des systèmes complexes dont on souhaite effectuer l'estimation. Il serait intéressant de transposer le problème d'estimation à état unique pour des estimateurs capables de garder en mémoire un nombre fini d'états. Cette hypothèse a notamment été étudiée par le couplage de deux estimateurs à état unique dans [Coquand et al., 2020].

Troisième partie

Annexes

Annexe A

Preferential Discrete Model-based Diagnosis for Intermittent and Permanent Faults

Preferential Discrete Model-based Diagnosis for Intermittent and Permanent Faults

Valentin Bouziat[1] and **Xavier Pucel**[1] and **Stéphanie Roussel**[1] and **Louise Travé-Massuyès**[2]
[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France
e-mail: firstname.name@onera.fr
[2]LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
e-mail: louise@laas.fr

Abstract

In this paper we consider the diagnosis of intermittent and permanent faults in discrete event systems. We present a logic based modeling approach associated with conditional preferences in order to produce a single diagnosis at each time step. Like all incomplete diagnosis approaches, ours is subject to deadlocks between the system and its diagnoser. In this paper, we address the detection of such deadlocks at design time with the rich semantic model-checker ELECTRUM.

1 Introduction

Using robots in situations where teleoperation is impossible or difficult requires the robots to have some level of decisional autonomy. In particular, an autonomous robot needs to detect and respond appropriately to abnormal losses of performance, due to degradations of the robotic system, or to external disturbances. This requires an elaborate fault management strategy, which can be challenging to design, especially when the mission is complex and the robot is subject to many faults. We are therefore interested in techniques that facilitate the identification of non-nominal operation modes, the detection of faults, and the estimation of the remaining capabilities of the robot.

We propose a modeling formalism constructed so that it meets three requirements. First, it should incrementally calculate a single diagnosis at each time step. This requirement stems from constraints on system performance: one can not afford to compute all possible diagnoses on a large system for a long period of time. Moreover, considering the complete robotic system, the diagnostic module is in our case associated with a deterministic planner which expects a single diagnosis as input. Second, we require our formalism to take into account both permanent and intermittent faults, in particular to distinguish disturbances (noise, false contact, etc.), supposedly intermittent, from degradations (breakdowns, etc.), often permanent. Third, since the diagnosis is used to make autonomous decisions, we want to avoid gooing back and modifying a diagnosis previously issued. We also forbid from producing a sequence of diagnoses that does not correspond to a possible evolution of the system (for example to remove a permanent fault). The principle is that the diagnosis should be a reliable piece of information, from which the robot can make a safe autonomous decision. If it is impossible to always produce such a diagnosis, then we want to detect it at design time so that the robot, the mission, or the fault management strategy can be modified.

In this paper, we describe how such requirements can create situations where the robotic system produces an observation sequence for which the diagnoser has no explanation. In this kind of situation, which we call a *deadlock*, the entire decision process supported by the diagnostic engine fails. This is why we are interested in detecting deadlock scenarios at the design time.

The paper is sturctured as follows. We start with a presentation of the state of the art in section 2. We then introduce our modeling formalism in section 3, and describe the deadlock issue in section 4. We propose a verification method at design time based on the model-checker ELECTRUM [1] in section 5, and present experimental results on small multi-robot systems in section 6. Perspectives are discussed in section 7.

2 Related Work

Our work addresses the diagnosis of discrete event systems. In [2] the authors define a framework for the diagnosis of systems modeled by finite state machines subject to permanent faults. The authors describe the construction of a diagnoser that allows an incremental computation of the diagnosis. However, their diagnoser requires memorizing all the possible state-diagnosis pairs, which severely limits its scalability. Their definition of diagnosability illustrates the importance of validating the performance of a diagnoser at design time.

Our formalism associates propositional logic constraints with a conditional preference theory from [3]. It is inspired by the formalism used in [4] to produce incremental diagnoses. In this related work, the authors just point at the risk of encountering a deadlock. In [5], the same authors propose to detect deadlock scenarios between a system and its diagnoser by an iterative model-checking approach but this approach is difficult to implement and no associated experimentations are provided.

The problem of deadlocks between a system and its diagnoser occurs in [6] where the diagnoser goes back in the execution of the diagnoser in order to find a consistent explanation. This solution violates our third requirement explained above. We want to produce reliable diagnoses, or no diagnoses at all.

A classic way to order diagnoses is to prefer the minimal ones, this approach knows several variants compared in [7]. Other approaches like [8] order faults according to some criterion, and deduce an order on diagnoses. In all

these approaches, the diagnosis ordering is unconditional, *i.e.* observations have no effect on the order of diagnoses. Our model uses conditional preferences precisely to remove this limitation, and to make it possible to prefer certain diagnoses based on present and past observations.

The diagnosis of intermittent faults is discussed in the literature from different points of view. In [9], the same function is called multiple times, but while faults manifest themselves in an intermittent manner, the underlying diagnosis is constant from one test to another. In [10], a repair event is associated with each fault, and the diagnosis task consists in detecting for each fault which event (fault or repair) happened last. A diagnoser is built to allow incremental evaluation, which involves the previously mentioned scaling problems since all diagnoses are kept in memory.

3 Diagnosis model

Our diagnosis model is a tuple (s_0, Δ, Γ), where s_0 is the initial system state, Δ is the behavioral model of the system and Γ is the conditional preference model. We assume that the system evolves with discrete events dynamics and that all time steps have the same duration.

3.1 Variables

We use a set of propositional variables $\mathtt{P}$ to describe the state of the system. $\mathtt{P}$ is partitioned into two subsets $\mathtt{O}$ and $\mathtt{E}$, which respectively represent the elements of the system that are observed and those to be estimated. At each discrete time step, given a truth assignment to the variables from $\mathtt{O}$, our goal is to estimate the value for the variables from $\mathtt{E}$.

Notations For a variable set $\mathtt{X}$, we define an *assignment* as a function from $\mathtt{X}$ to $\{\top, \bot\}$ that associates to each variable $\mathtt{x}$ from $\mathtt{X}$ the boolean value *true* ($\top$) or *false* ($\bot$). We write x and $\overline{x}$ the assignments to $\{\mathtt{x}\}$ such that $x(\mathtt{x}) = \top$ and $\overline{x}(\mathtt{x}) = \bot$. For a set of variables $\mathtt{X} = \{\mathtt{x_1}, \ldots, \mathtt{x_n}\}$, if for $i \in [1, n]$, f_i is an assignment $\{\mathtt{x_i}\} \to \{\top, \bot\}$, then $f_1\ f_2 \ldots f_n$ denotes the f assignment on $\mathtt{X}$ such that $\forall \mathtt{x_i} \in \mathtt{X}, f(\mathtt{x_i}) = f_i(\mathtt{x_i})$. For example, $a\ \overline{b}\ \overline{c}$ is the assignment to $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ that assigns true to $\mathtt{a}$ and false to $\mathtt{b}$ and $\mathtt{c}$.

Definition 1 (Observation)**.** *An observation, denoted o, is an assignment of the variables of $\mathtt{O}$.*

Definition 2 (State)**.** *A state, denoted s, is an assignment of the variables of $\mathtt{P}$. $\mathcal{S}$ denotes the set of states.*

3.2 Behavioral Model

The behavioral model $\Delta \subseteq \mathcal{S}^2$ is the transition relation of the system. In order to refer to the state of the system at the previous time step, we introduce a bijective function pre on the set $\mathtt{P}$. For a variable $\mathtt{p}$ in $\mathtt{P}$, $pre(\mathtt{p})$ represents the value of the variable $\mathtt{p}$ at the previous time step. Formally, we define a variable set $\mathtt{P_{pre}} = \{\mathtt{pre\_p} \,|\, \mathtt{p} \in \mathtt{P}\}$ such as $\forall \mathtt{p} \in \mathtt{P}, pre(\mathtt{p}) = \mathtt{pre\_p}$. Δ is represented by a set of propositional logic formulas Δ_p that can relate to both the variables of $\mathtt{P}$ and those of $\mathtt{P_{pre}}$.

A pair of states (s_{pre}, s_{now}) belongs to the transition relation Δ when the system can be in the state s_{pre} at time $t-1$ and in the state s_{now} at time t. To formally link Δ to Δ_p, for any pair of states (s_{pre}, s_{now}), we define the assignment $\sigma_{s_{pre}, s_{now}}$ on variables from $\mathtt{P} \cup \mathtt{P_{pre}}$ such as $\forall \mathtt{p} \in \mathtt{P}, \sigma_{s_{pre}, s_{now}}(\mathtt{p}) = s_{now}(\mathtt{p})$ and $\sigma_{s_{pre}, s_{now}}(pre(\mathtt{p})) = s_{pre}(\mathtt{p})$. We consider that a pair of states belongs to the transition relation Δ if and only if the corresponding assignment satisfies the formulas of Δ_p.

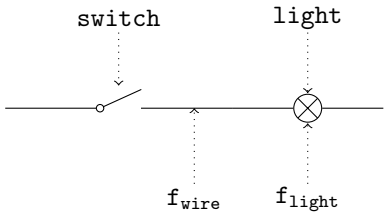


Figure 1: A simple system composed of a lamp and a switch

Definition 3 (Transition)**.** *A pair of states $(s_{pre}, s_{now}) \in \mathcal{S}^2$ is a transition, denoted $(s_{pre}, s_{now}) \in \Delta$, if and only if $\sigma_{s_{pre}, s_{now}} \models \Delta_p$.*

In the remaining of this paper, we do not distinguish Δ et Δ_p.

In order to reason about the possible evolutions of the system state, and the associated observation sequences, we define consistent state sequences and consistent observation sequences as follows.

Definition 4 (Consistent state sequence)**.** *A state sequence $(s_0, s_1, \ldots, s_n) \in \mathcal{S}^n$ is consistent if and only if $\forall i \in [1, n]$, $(s_{i-1}, s_i) \in \Delta$.*

Definition 5 (Consistent observation sequence)**.** *An observation sequence $(o_0, o_1, \ldots, o_n)$ is consistent if and only if there exist a consistent state sequence $(s_0, s_1, \ldots, s_n)$ such that $\forall i \in [0, n], \forall \mathtt{o} \in \mathtt{O}, s_i(\mathtt{o}) = o_i(\mathtt{o})$.*

In the following, when there is no ambiguity, state and observation sequences are assumed to be consistent.

Given a previous state and an observation of the system, we define the set of candidates for diagnosis as the set of states that are compatible with the previous state, the observation and the behavioral model Δ.

Definition 6 (Diagnosis candidates)**.** *The set $\mathcal{S}_\Delta(s_{pre}, o)$ of diagnosis candidates for a previous state s_{pre} and an observation o is defined by:*

$$\mathcal{S}_\Delta(s_{pre}, o) = \big\{ s_{now} \in \mathcal{S} \,|\, (s_{pre}, s_{now}) \in \Delta \text{ and } \forall \mathtt{o} \in \mathtt{O}, s_{now}(\mathtt{o}) = o(\mathtt{o}) \big\}$$

Example 1. *The system illustrated in Figure 1 is composed of a lamp controlled by a switch. This system may be subject to a permanent fault and an intermittent fault, our goal is to estimate their presence or absence. The set $\mathtt{O}$ is composed of* `light` *which is true when the light is on, false otherwise, and* `switch` *which is true when the switch is closed (current flows), false otherwise. The set $\mathtt{E}$ contains two variables representing the two faults that may occur: $\mathtt{f_{wire}}$ represents an intermittent loose contact in the wire and $\mathtt{f_{light}}$ a permanent lamp failure.*

The two rules of Δ represent the following behavior: the lamp glows when the switch is closed and when there is no fault on the wire nor in the lamp (δ_1). If the bulb of the lamp was broken at the previous state, then it is at the present time (δ_2), i.e. the fault $\mathtt{f_{light}}$ is permanent. In the initial state $s_0 =$ light switch $\overline{f_{light}}\ \overline{f_{wire}}$, the lamp glows and there is no fault.

$$\Delta = \left\{ \begin{array}{cc} \texttt{light} \leftrightarrow \texttt{switch} \wedge \neg \texttt{f}_{\texttt{light}} \wedge \neg \texttt{f}_{\texttt{wire}} & (\delta_1) \\ \texttt{pre\_f}_{\texttt{light}} \rightarrow \texttt{f}_{\texttt{light}} & (\delta_2) \end{array} \right\}$$

From the behavioral model Δ, we can calculate diagnosis candidates. For instance, at time step 1, s_0 is the previous system state, let us consider the observation $o_1 = \overline{light}$ switch. The candidate states set is $\mathcal{S}_\Delta(s_0, o_1) = \{s_{now}, s'_{now}, s''_{now}\}$ with:

$$\begin{array}{rcl} s_{now} & = & \overline{light}\ switch\ \overline{f_{light}}\ f_{wire} \\ s'_{now} & = & \overline{light}\ switch\ f_{light}\ \overline{f_{wire}} \\ s''_{now} & = & \overline{light}\ switch\ f_{light}\ f_{wire} \end{array}$$

In the state s_{now}, the intermittent fault is present alone; in s'_{now}, the permanent fault is present alone; both faults are present in s''_{now}.

3.3 Single diagnosis choice

For a given previous state and observation, in order to produce a single diagnosis, we have to choose a single state from all the candidates of $\mathcal{S}_\Delta(s_{pre}, o)$. This choice is dictated by the conditional preference model Γ that consists in an ordered set of *conditional preferences*.

A conditional preference relates to a variable to be estimated and indicates under which condition we prefer the diagnoses that assign true of false to this variable to the other diagnoses. A preference is a form of "soft constraint", it is only applied when there exists diagnoses with both values for the variable. A formal definition follows.

Definition 7 (Conditional preference)**.** *A conditional preference γ on a variable* e *of* E*, is denoted* $\texttt{cond} : e \prec \overline{e}$*. The preference's condition* cond *is a propositional formula on* $\texttt{P} \cup \texttt{P}_{\texttt{pre}}$*. The variable* e *is called the preference's target.*

Note that $\texttt{cond} : \overline{e} \prec e$ is equivalent to $\neg\texttt{cond} : e \prec \overline{e}$. For a variable e from E, the conditional preference $\texttt{cond} : e \prec \overline{e}$ expresses a preference for states in which e is true to those where e is false if and only if cond is satisfied.

Formally, a preference $\gamma = \texttt{cond} : e \prec \overline{e}$ defines a partial order $\prec_\gamma$ an equivalence relation $\approx_\gamma$ between pairs of transitions as follows. For all triples $s_{pre}, s, s' \in \mathcal{S}^3$, γ strictly prefers the transition (s_{pre}, s) to transition (s_{pre}, s') (denoted $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$) if and only if $\sigma_{s_{pre},s} \models \texttt{cond} \leftrightarrow \texttt{e}$ and $\sigma_{s_{pre},s'} \not\models \texttt{cond} \leftrightarrow \texttt{e}$. Transitions (s_{pre}, s) et (s_{pre}, s') are equivalent to γ (denoted $(s_{pre}, s) \approx_\gamma (s_{pre}, s')$) if and only if $(\sigma_{s_{pre},s} \models \texttt{cond} \leftrightarrow \texttt{e}) \Leftrightarrow (\sigma_{s_{pre},s'} \models \texttt{cond} \leftrightarrow \texttt{e})$.

Example 2. *Let us consider the preference $\gamma = \neg\texttt{light} : f_{light} \prec \overline{f_{light}}$. This preference indicates that if both f_{light} and $\overline{f_{light}}$ are part of some diagnosis candidate, then f_{light} is preferred is and only if* $\neg\texttt{light}$ *holds. Formally, we prefer diagnoses that satisfy $\neg\texttt{light} \leftrightarrow \texttt{f}_{\texttt{light}}$. For the states s_{now}, s'_{now} and s''_{now} in example 1, as* $\neg\texttt{light}$ *holds, we prefer the states in which $\texttt{f}_{\texttt{light}}$ holds, i.e. the states s'_{now} and s''_{now}. Formally, $s'_{now} \prec_\gamma s_{now}$, $s''_{now} \prec_\gamma s_{now}$ and $s''_{now} \approx_\gamma s'_{now}$.*

We assume that in Γ, all estimated variables are the target of a conditional preference. This raises the question of the order in which preferences are applied, that we address now.

Definition 8 (Conditional preference model)**.** *A conditional preference model Γ is a sequence of conditional preferences $(\gamma_1, \gamma_2, .., \gamma_n)$ with $\gamma_i = \texttt{cond}_\texttt{i} : e_i \prec \overline{e_i}$ for $i \in [1, n]$ such that each variable* e *from* E *is the target of exactly one preference.*

We require Γ to be an acyclic preference model, which guarantees its consistence (see [11]). For example, the following two preferences form a cycle: $\texttt{a} : b \prec \overline{b}$ et $\texttt{b} : a \prec \overline{a}$. In the first preference the value of a depends on that of b, and in the second preference the exact opposite happens. While the literature contains work on cyclic preference networks [11], in this paper we assume that Γ is acyclic. This means that the condition of a preference γ_i cannot use variables that are the target of the following preferences in the sequence. Formally, $\forall i \in [1, n]$, the scope of the condition $\texttt{cond}_\texttt{i}$ is a subset of $\texttt{P}_{\texttt{pre}} \cup \texttt{O} \cup \{\texttt{e}_\texttt{j} | 1 \leq j < i\}$.

From a preference model Γ, it is possible to define a partial order $\prec_\Gamma$ between pairs of states as follows. Intuitively, as in a lexicographic order, we consider preferences γ_i in their index order in Γ and we apply at each index the order relation $\prec_{\gamma_i}$ defined above. This order is partial because it compares only pairs of transitions (s_{pre}, s_{now}) and $(s_{pre}{}', s_{now}{}')$ that have the same previous state (*i.e.* $s_{pre} = s_{pre}{}'$) and whose successor states produce the same observation (*i.e.* $\forall \texttt{o} \in \texttt{O} s_{now}(\texttt{o}) = s_{now}{}'(\texttt{o})$).

Formally, $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) is strictly preferred to (s_{pre}, s') by Γ (denoted $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$) if and only if there exists $i \in [1, n]$ such that for all $j < i$, $(s_{pre}, s) \approx_{\gamma_j} (s_{pre}, s')$ and $(s_{pre}, s) \prec_{\gamma_i} (s_{pre}, s')$.

Although the $\prec_\Gamma$ order is partial on $\mathcal{S}^2$, it is complete on any set of diagnosis candidates. We use this order relation to define the preferred diagnosis at each time step.

Proposition 1. *Let s_{pre} be a state, o an observation, Δ a behavioral model and Γ a preference model. If s and s' are two candidate states for s_{pre} et o $(s, s' \in \mathcal{S}_\Delta(s_{pre}, o)^2)$ such that $s \neq s'$ then we have either $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$ or $(s_{pre}, s') \prec_\Gamma (s_{pre}, s)$.*

Proof By definition 6, s and s' belong to $\mathcal{S}_\Delta(s_{pre}, o)$ implies that $\forall \texttt{o} \in \texttt{O}, s(\texttt{o}) = s'(\texttt{o})$. Therefore, $s \neq s'$ means that there is a variable $\texttt{e} \in \texttt{E}$ such that $s(\texttt{e}) \neq s'(\texttt{e})$. So, there exists a preference $\gamma \in \Gamma$ such that $s \approx_\gamma s'$ does not hold. Let i be the index of the first preference of the sequence in this case. Formally, γ_i is such that $\forall j < i$, $(s_{pre}, s) \approx_{\gamma_j} (s_{pre}, s')$, $(s_{pre}, s) \approx_{\gamma_i} (s_{pre}, s')$ does not hold. This means that $(s_{pre}, s) \prec_{\gamma_i} (s_{pre}, s')$ or $(s_{pre}, s') \prec_{\gamma_i} (s_{pre}, s)$. From the order $\prec_\Gamma$, we then have $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$ or $(s_{pre}, s') \prec_\Gamma (s_{pre}, s)$. □

3.4 Estimation process

At each step, given the set of candidate states $\mathcal{S}_\Delta(s_{pre}, o)$ we select the state preferred by the conditional preference model Γ.

Definition 9 (Estimated state)**.** *The estimated state $\hat{s} = estim(s_{pre}, o)$ for a previous state s_{pre} and an observation o is the element of $\mathcal{S}_\Delta(s_{pre}, o)$ preferred by $\prec_\Gamma$. Formally, $\hat{s} = estim(s_{pre}, o)$ if and only if:*

1. *$\hat{s} \in \mathcal{S}_\Delta(s_{pre}, o)$, and*
2. *$\forall s_{now} \in \mathcal{S}_\Delta(s_{pre}, o)$ such that $s_{now} \neq \hat{s}$, we have $(s_{pre}, \hat{s}) \prec_\Gamma (s_{pre}, s_{now})$.*

From the initial state s_0, we now define the sequence of estimated states that is produced by the diagnoser for a given observation sequence.

Definition 10 (Estimated state sequence)**.** *A state sequence* $(s_0, \hat{s}_1, \hat{s}_2, ..., \hat{s}_k)$ *is the estimated state sequence for the observation sequence* $(o_0, o_1, o_2,...,o_k)$ *if and only if* $\forall i \in [1, k], \hat{s}_i = estim(\hat{s}_{i-1}, o_i)$.

Example 3. *Let us associate the lamp model* Δ *from example 1 to the following preferences:*

$$\Gamma = \left(\begin{array}{rl} \texttt{pre\_f}_{\texttt{wire}} : f_{wire} \prec \overline{f_{wire}} & (\gamma_1) \\ \bot : f_{light} \prec \overline{f_{light}} & (\gamma_2) \end{array} \right)$$

The first preference (γ_1) *declares that we prefer the value for* $\texttt{f}_{\texttt{wire}}$ *that was estimated at the previous time step. This mechanism makes it possible to bring some stability to the diagnosis since* $\texttt{f}_{\texttt{wire}}$ *is an intermittent fault: if* Δ *allows both diagnoses for* $\texttt{f}_{\texttt{wire}}$*, we prefer to maintain the diagnosis that was chosen at the previous time step.*

The preference (γ_2) *indicates that we always prefer to assume that* $\texttt{f}_{\texttt{light}}$ *is absent.*

In example 1, with $s_0 = light\ switch\ \overline{f_{light}}\ \overline{f_{wire}}$*) as the previous state and* $o_1 = \overline{light}$ *switch as the observation, the set of diagnosis candidates* $\mathcal{S}_\Delta(s_{pre}, o)$ *contains the three states* $s_{now} = \overline{light}$ *switch* $\overline{f_{light}}\ f_{wire}$*,* $s_{now}{}' = \overline{light}$ *switch* $f_{light}\ \overline{f_{wire}}$ *and* $s''_{now} = \overline{light}$ *switch* $f_{light}\ f_{wire}$*.*

By applying (γ_1) *first, as* $\overline{pre\_f_{wire}}$ *holds in the three states, we prefer the states in which* $\overline{f_{wire}}$*. In our case, only* $s_{now}{}'$ *satisfies this criterion. As there is only one diagnosis left, we do not need to apply preference* (γ_2)*. The preferred single state is* $\hat{s}_1 = estim(s_0, o_1) = s_{now}{}' = \overline{light}\ switch\ f_{light}\ \overline{f_{wire}}$*.*

4 Deadlock

A deadlock situation occurs when the diagnoser has previously estimated a state $\hat{s}$ different from the current system state s, and receives an observation o in contradiction with $\hat{s}$ and Δ. In such a situation the candidates set for $\hat{s}$ and o is empty and the diagnoser is unable to return a diagnosis.

Definition 11 (Deadlock)**.** *An estimation model* (s_0, Δ, Γ) *is in a deadlock situation for an observation sequence* $(o_0, o_1, o_2, \ldots, o_k)$ *with* $k > 1$ [1] *if and only if:*

1. *there exists an estimated state sequence for* $(o_0, \ldots, o_{k-1})$*, and*
2. *there exist no estimated state sequence for* $(o_0, \ldots, o_k)$

Example 4. *Let* (o_0, o_1, o_2, o_3) *be the observation sequence produced by the system and along with the associated state sequence* (s_0, s_1, s_2, s_3) *described in Figure 2.*

[1] There can be no deadlock at $\hat{s}_1$ because we assume that the diagnoser is correctly initialized at s_0.

Step i	o_i	s_i	$\hat{s}_i$
0	*light switch*	$\overline{f_{light}}\ \overline{f_{wire}}$	$\overline{f_{light}}\ \overline{f_{wire}}$
1	*light switch*	$\overline{f_{light}}\ \overline{f_{wire}}$	$\overline{f_{light}}\ \overline{f_{wire}}$
2	$\overline{light}$ *switch*	$\overline{f_{light}}\ f_{wire}$	$f_{light}\ \overline{f_{wire}}$
3	*light switch*	$\overline{f_{light}}\ \overline{f_{wire}}$	/

Figure 2: Deadlock scenario for example 4. In columns s_i and $\hat{s}_i$, we omit variables from O whose values are identical to the column o_i, and we only represent variables from E).

When observation o_1 *is received,* Γ *selects the preferred state* $\hat{s_1}$ = *light switch* $\overline{f_{light}}\ \overline{f_{wire}}$*. Then for observation* o_2*, we are in the situation described in Example 3 and the preferred state is* $\hat{s}_2$ = $\overline{light}$ *switch* $f_{light}\ \overline{f_{wire}}$*. Observation* o_3 *is in contradiction with the previously estimated state. In fact the estimator has estimated* f_{light} *in the previous state, meaning that variable* $\texttt{pre\_f}_{\texttt{light}}$ *is true. Rules of* Δ *are not consistent with such a configuration:* (δ_1) *requires* $\texttt{f}_{\texttt{light}}$ *to be false because the lamp glows while* (δ_2) *requires* $\texttt{f}_{\texttt{light}}$ *to be true since the associated fault is permanent.*

Therefore, there is an estimated state sequence for (o_0, o_1, o_2)*: this is the sequence* $(s_0, \hat{s}_1, \hat{s}_2)$*. However, since there is no estimated state sequence for* (o_0, o_1, o_2, o_3)*, the pair system-diagnoser is in a deadlock situation for this observation sequence.*

In the previous example, the deadlock situation could be easily avoided by changing the order or conditions of preferences. However, as soon as one is interested in more complex systems, it becomes difficult to anticipate the deadlock situations and even more to solve them. In this paper, we focus on detecting these deadlocks and leave their resolution for future work.

5 Deadlock Checking

In this section, we show how to use a model-checker to identify deadlock scenarios at the design phase of the diagnostic model.

5.1 Deadlock checker

The verification method we propose is inspired from the Twin-Plant [12] technique, which makes it possible to verify the diagnosability of a system by constructing the synchronous product of two finite state machines.

Our method is similar since it involves constructing a deadlock verifier by synchronizing two state machines. The first state machine represents the system and is constrained only by the transition relation Δ. We use it to generate consistent observation sequences. The second state machine is the diagnoser built from the whole estimation model. It is also constrained by Δ, but it deterministically selects the next state by applying Γ preferences. The verifier is the product of these two state machines synchronized on observable variables at each time step. Deadlock checking then consists in checking whether there exists an observation sequence that leads the verifier to a state in which the system has a successor state, but the diagnoser has none.

In order to distinguish the states of the two state machines presented above, *i.e.* the state of the system and that of the diagnoser, we introduce two variable sets $\texttt{P}^{\texttt{sys}}$ and $\texttt{P}^{\texttt{est}}$ that are direct copies of P. We also use a est_sat variable that indicates whether the diagnoser has a successor state.

Definition 12 (Verifier variables)**.** *The set of the verifier variables is defined by* $\texttt{P}^{\texttt{verif}} = \texttt{P}^{\texttt{sys}} \cup \texttt{P}^{\texttt{est}} \cup \{\texttt{est\_sat}\}$*, where :*

- $\texttt{P}^{\texttt{sys}} = \{\texttt{p\_sys} \mid \texttt{p} \in \texttt{P}\}$ *is the set of variables describing the system state ;*
- $\texttt{P}^{\texttt{est}} = \{\texttt{p\_est} \mid \texttt{p} \in \texttt{P}\}$ *is the set of variables describing the diagnoser state;*
- est_sat *indicates whether there is an estimated state for the diagnoser.*

A state of the verifier is an assignment on variables of $\texttt{P}^{\texttt{verif}}$*.*

In order to define the state machine resulting from the synchronous product, we successively define the initial state of the verifier (definition 13) and its transition relation (definition 14).

Definition 13 (Verifier initial state)**.** *The initial state s_0^{verif} of the verifier is such that:*

- $\forall \mathtt{p} \in \mathtt{P}$, $s_0^{verif}(\mathtt{p\_sys}) = s_0(\mathtt{p})$,
- $\forall \mathtt{p} \in \mathtt{P}$, $s_0^{verif}(\mathtt{p\_est}) = s_0(\mathtt{p})$,
- $s_0^{verif}(\mathtt{est\_sat}) = \top$.

Definition 14 (Verifier transition relation)**.** *Let s_{pre}^{verif} and s_{now}^{verif} be two verifier states. The pair $(s_{pre}^{verif}, s_{now}^{verif})$ is a verifier transition if and only if:*

(1) variables associated with observations take the same value on system and diagnoser sides,

(2) the system transition described by variables of $\mathtt{P^{sys}}$ satisfies Δ,

(3) the variable est_sat *indicates whether there exists a possible estimated state (i.e. if the candidates set is not empty), and*

(4) if est_sat *is true, then the diagnoser transition described by variables of $\mathtt{P^{est}}$ satisfies Δ and Γ.*

Formally :

$$\begin{array}{l}\forall \mathtt{o} \in \mathtt{O}, s_{pre}^{verif}(\mathtt{sys\_o}) = s_{pre}^{verif}(\mathtt{est\_o}) \text{ and} \\ \forall \mathtt{o} \in \mathtt{O}, s_{now}^{verif}(\mathtt{sys\_o}) = s_{now}^{verif}(\mathtt{est\_o})\end{array} \quad (1)$$

$$\begin{array}{l}\exists (s_{pre}, s_{now}) \in \Delta, \forall \mathtt{p} \in \mathtt{P}, \\ \quad s_{pre}(\mathtt{p}) = s_{pre}^{verif}(\mathtt{sys\_p}) \text{ and} \\ \quad s_{now}(\mathtt{p}) = s_{now}^{verif}(\mathtt{sys\_p})\end{array} \quad (2)$$

$$\mathtt{est\_sat} \leftrightarrow \left(\begin{array}{l}\exists (s_{pre}, s_{now}) \in \Delta, \\ \quad \forall \mathtt{p} \in \mathtt{P}, s_{pre}(\mathtt{p}) = s_{pre}^{verif}(\mathtt{est\_p}) \text{ and} \\ \quad \forall \mathtt{p} \in \mathtt{O}, s_{now}(\mathtt{p}) = s_{now}^{verif}(\mathtt{est\_p})\end{array}\right) \quad (3)$$

$$\mathtt{est\_sat} \rightarrow \left(\begin{array}{l}\exists (s_{pre}, s_{now}) \in \Delta, \forall \mathtt{p} \in \mathtt{P}, \\ \quad s_{pre}(\mathtt{p}) = s_{pre}^{verif}(\mathtt{est\_p}) \text{ and} \\ \quad s_{now}(\mathtt{p}) = s_{now}^{verif}(\mathtt{est\_p}) \text{ and} \\ \quad \left(\begin{array}{l}\nexists s_{best} \in \mathcal{S}, \\ \quad \forall \mathtt{o} \in \mathtt{O}, s_{best}(\mathtt{o}) = s_{now}(\mathtt{o}), \text{ and} \\ \quad (s_{pre}, s_{best}) \in \Delta, \text{ and} \\ \quad (s_{pre}, s_{best}) \prec_\Gamma (s_{pre}, s_{now})\end{array}\right)\end{array}\right) \quad (4)$$

Proposition 2. *An estimation model (s_0, Δ, Γ) is subject to a deadlock if and only if the associated verifier contains a path in which* est_sat *is false. The deadlock scenario is the observation sequence corresponding to the states of the verifier.*

Proof Suppose that the estimation model is subject to a deadlock. According to Definition 11, there exists an observation sequence $(o_0, o_1, \ldots, o_n)$ generated by a consistent state sequence $(s_0, s_1, \ldots, s_n)$, such that there is an estimated state sequence $(\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_{n-1})$ for the partial sequence $(o_0, o_1, \ldots, o_{n-1})$ and that there does not exist an estimated state sequence $(\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_n)$ for the complete sequence. We then build the verifier's state sequence $(s_0^{verif}, s_1^{verif}, \ldots, s_n^{verif})$ as follows:

- $\forall i \in [0, n], \forall p \in \mathtt{P}, s_i^{verif}(\mathtt{p\_sys}) = s_i(\mathtt{p})$,
- $\forall i \in [0, n-1], \forall p \in \mathtt{P}, s_i^{verif}(\mathtt{p\_est}) = \hat{s}_i(\mathtt{p})$,
- $\forall i \in [0, n-1], s_i^{verif}(\mathtt{est\_sat}) = \top$,
- $\forall \mathtt{p} \in \mathtt{O}, s_n^{verif}(\mathtt{p\_est}) = o_n(\mathtt{p})$,
- $\forall \mathtt{p} \in \mathtt{E}, s_n^{verif}(\mathtt{p\_est}) = \top$,
- $s_n^{verif}(\mathtt{est\_sat}) = \bot$.

We show that for all $i \in [0, n-1]$, $(s_i^{verif}, s_{i+1}^{verif})$ is a transition of the checker: let i be an integer in $[0, n-1]$, we show that the four parts of Definition 14 are satisfied:

- $\forall \mathtt{o} \in \mathtt{O}$, $\forall p \in \mathtt{P}, s_i(\mathtt{p}) = \hat{s}_i(\mathtt{p})$ and $s_{i+1}(\mathtt{p}) = \hat{s}_{i+1}(\mathtt{p})$. Equation (1) holds;
- equation (2) holds by construction of the verifier;
- for $i < n-1$, $s_i^{verif}(\mathtt{est\_sat}) = \top$ and $(\hat{s}_i, \hat{s}_{i+1})$ is the pair of states in Δ satisfying equation (3);
- for $i = n-1$, the definition of a deadlock implies that there does not exist a pair of states satisfying Δ: equation (3) is satisfied;
- for $i < n-1$, $s_i^{verif}(\mathtt{est\_sat}) = \top$ and $(\hat{s}_i, \hat{s}_{i+1})$ is the pair of states in Δ satisfying equation (4);

For the reciprocal, let us assume that the checker has a path in which est_sat is false. We consider such a path $(s_0^{verif}, s_1^{verif}, \ldots, s_n^{verif})$ in wich s_n^{verif} is the only state in which false is assigned to est_sat. We then build two states sequences $(s_0, s_1, \ldots, s_n)$ and $(s_0, \hat{s}_1, \ldots, \hat{s}_{n-1})$ as follows:

- $\forall i \in [0, n], \forall \mathtt{p} \in \mathtt{P}, s_i(\mathtt{p}) = s_i^{verif}(\mathtt{p\_sys})$,
- $\forall i \in [0, n-1], \forall \mathtt{p} \in \mathtt{P}, \hat{s}_i(\mathtt{p}) = s_i^{verif}(\mathtt{p\_est})$

We prove that these state sequences are consistent with Δ. From definition 14, est_sat is false (last time step) implies that no state s_{now} allows to satisfy the right part of the condition (equation 3) for $s_{pre} = \hat{s}_{n-1}$. This means that no state meets the definition 6. There is a deadlock for the observation sequence generated by $(s_0, s_1, \ldots, s_n)$. □

5.2 Choice of the model-checker

Model-checking is a set of techniques and computer tools for checking properties on systems. Among the many model-checkers in the literature, NuSMV [13] and NuXMV are known for their effectiveness in checking temporal properties on dynamic systems. They are based on temporal logic like LTL or CTL. Another family of model checkers is specialized in checking properties on structurally complex models, but without temporal dynamics. This is the case of the Alloy Analyzer model-checker [14] in which the models are based on the Alloy language, itself based on first order logic.

In the case of deadlock verification, we need to express the temporal dynamics of the system but also the preferences of the diagnoser, which are not easily expressed in propositional logic. We therefore chose to use the model-checker ELECTRUM [1] which is based on the Alloy language and integrates temporal dynamics as in NuSMV.

5.3 Quantified form of preference

The concept of preference is part of the verifier definition. This means that the model-checker language must allow to express them. To do so, we define a *quantified form* for preferences.

Definition 15 (Preference quantified form)**.** *Let be* $\Gamma = (\gamma_1, \ldots, \gamma_n)$ *a preference model in which each preference has form* $\gamma_i = \mathtt{cond_i} : e_i \prec \overline{e_i}$*. The quantified form of* γ_i *is the formula* ψ_i *defined by:*

$$\psi_i = (\forall \mathtt{e_i}, \exists \mathtt{e_{i+1}}, \ldots, \mathtt{e_n},\ \Delta) \rightarrow (\mathtt{e_i} \leftrightarrow \mathtt{cond_i})$$

The quantified form ψ_i is composed of two parts. The left part is only satisfied when for both truth values variable $\mathtt{e_i}$, there is a way to assign the target variables of the following preferences and still satisfy Δ. It represents the cases when the preference is actually applied. The right part expresses the effect of the preference, *i.e* that $\mathtt{e_i}$ is assigned to true if and only if the condition $\mathtt{cond_i}$ is satisfied. Through the following proposition, we show that Γ preferences and their quantified form are equivalent from the point of view of the estimated state for a previous state and an observation.

Proposition 3. *For a previous estimated state* $\hat{s}_{pre}$*, an observation* o*, a candidate* $\hat{s} \in \mathcal{S}_\Delta(\hat{s}_{pre}, o)$*,* $\hat{s} = estim(\hat{s}_{pre}, o)$ *if and only if the assignment* $\sigma_{\hat{s}_{pre},\hat{s}}$ *satisfies conjunction* $\bigwedge_{i\in[1,n]} \psi_i$*.*

Proof. Let $\hat{s}_{pre}$ be a previous estimated state, o an observation and $\hat{s}$ a state in $\mathcal{S}_\Delta(\hat{s}_{pre}, o)$. We inductively show that $\hat{s}$ is preferred for the preference sequence $(\gamma_1, \ldots, \gamma_k)$ if and only if the assignment $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfies $\bigwedge_{i\in[1,k]} \psi_i$.

For $k = 1$, the set of free variables in formula $lhs_1 = \forall \mathtt{e_1}, \exists \mathtt{e_2}, \ldots \mathtt{e_n}, \Delta$ is the set $\mathtt{O}$, since all the variables in $\mathtt{E}$ are quantified. The formula lhs_1 is satisfied by an observation o if and only if both oe_1 and $o\overline{e_1}$ have an extension in $\mathtt{P} \cup \mathtt{P_{pre}}$ that satisfies Δ. This means $\mathcal{S}_\Delta(\hat{s}_{pre}, o)$ contains at least two states s and s' with $s(\mathtt{e_1}) \neq s'(\mathtt{e_1})$. Among these two states, the one that satisfies $\mathtt{cond_1} \leftrightarrow \mathtt{e_1}$ is the preferred state by preference γ_1, and also satisfies ψ_1. As the states preferred by γ_1 are exactly the ones for which the assignment $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfies ψ_1, then the proposition holds for $k = 1$.

Let $k > 1$ and let us assume that the induction is valid at index $k-1$, *i.e.* that $\hat{s}$ is preferred for the preference sequence $(\gamma_1, \ldots, \gamma_{k-1})$ if and only if the assignment $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfies $\bigwedge_{i\in[1,k-1]} \psi_i$. Since s and s' belong to $\mathcal{S}_\Delta(\hat{s}_{pre}, o)$, since s is a state preferred for $(\gamma_1, \ldots, \gamma_{k-1})$ and since $\sigma_{\hat{s}_{pre},s'}$ satisfies $\bigwedge_{i\in[1,k-1]} \psi_i$, then for all $i < k$, $s(\mathtt{e_i}) = s'(\mathtt{e_i})$.

The set of free variables in formula $lhs_k = \forall \mathtt{e_k}, \exists \mathtt{e_{k+1}}, \ldots \mathtt{e_n}, \Delta$ is the set $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$. An assignment *op* to these variables satisfies lhs_k if and only if *op* e_k and *op* $\overline{e_k}$ both have an extension to $\mathtt{P} \cup \mathtt{P_{pre}}$ that satisfies Δ.

If for a given assignment *op* to $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$ the formula lhs_k does not hold, then for all s and s' in $\mathcal{S}_\Delta(\hat{s}_{pre}, o)$, $s(\mathtt{e_k}) = s'(\mathtt{e_k})$. Thus, the preference γ_k is not applied and the preferred states for $(\gamma_1, \ldots, \gamma_{k-1})$ are the also preferred for $(\gamma_1, \ldots, \gamma_k)$. In this case, $\psi_k = \top$ and the states s' such that $\sigma_{\hat{s}_{pre},s'}$ satisfy $\bigwedge_{i\in[1,k-1]} \psi_i$ are the same ones for which $\sigma_{\hat{s}_{pre},s'}$ satisfy $\bigwedge_{i\in[1,k]} \psi_i$ and the induction is valid.

If for a given assignment *op* to $\mathtt{O} \cup \{\mathtt{e_i} \mid i < k\}$ the formula lhs_k holds, then $\mathcal{S}_\Delta(\hat{s}_{pre}, o)$ contains at least two states s and s' such that for all $i < k$, $s(\mathtt{e_i}) = s'(\mathtt{e_i})$ and $s(\mathtt{e_k}) \neq s'(\mathtt{e_k})$. Between these two states, the one in which $\mathtt{cond_k} \leftrightarrow \mathtt{e_k}$ is true is the preferred state by preference γ_k. As the states preferred by γ_k are exactly the ones for which the assignment $\sigma_{\hat{s}_{pre},\hat{s}}$ satisfies ψ_k, thus the induction is valid at index k. □

```
pred delta[fwire,light,flight,
           switch,preF_flight : Bool] {
  (isTrue[preF_flight] => isTrue[flight])
  and
  (isTrue[light] <=> ( isTrue[switch] and
                       !isTrue[flight] and
                       !isTrue[fwire]))
}
var one sig RealSystem {
  var fwire : Bool,
  var preF_fwire : Bool,
  var light : Bool,
  var flight : Bool,
  var switch : Bool,
  var preF_flight : Bool,
}
fact always_delta_RealSystem {
  always {
    delta[RealSystem.fwire,
    RealSystem.preF_fwire,
    RealSystem.light,
    RealSystem.flight,
    RealSystem.switch,
    RealSystem.preF_flight]}
}
fact synch_obs {
  always {
    Estimator.light = RealSystem.light
    Estimator.switch = RealSystem.switch}
}
```

Figure 3: The observed system and observation synchronization in ELECTRUM.

```
fact next_Estimator {
  always {
    Estimator.preF_flight' =
            Estimator.flight
    Estimator.preF_fwire' =
            Estimator.fwire }
  }
```

Figure 4: Temporal dynamics in ELECTRUM.

5.4 Model Encoding with ELECTRUM

In this section, we illustrate how the verifier defined in Definition 13 and 14 is encoded in ELECTRUM. We illustrate the encoding of the system described in examples 1 and 3.

To model the system's state machine, we declare a predicate corresponding exactly to the model Δ then we declare a constraint (a `fact` in ELECTRUM) that specifies that the system respects the predicate at all time steps (`always` in ELECTRUM), as illustrated in figure 3. A similar constraint is declared for the diagnoser's state machine. Finally, we synchronize the variables from `O` of the observed system with those of the second state machine representing our estimator, as expressed in condition (1) of definition 14.

To represent the temporal aspect of our formalism, we use the operator `'` from ELECTRUM which describes a variable at the next time step. Figure 4 reproduces the intention of the bijective function pre (`p_pre` value at next state is equal to `p` value at current state).

ELECTRUM supports the use ofrelational algebra operators. It is possible for each preference to write a predicate

```
pred pref_fwire_possible{
  all fwire : Bool |some flight : Bool |
    delta[fwire,
    Estimator.preF_fwire,
    Estimator.light,
    flight,
    Estimator.switch,
    Estimator.preF_flight]
}
pred pref_fwire_applied{
  isTrue[Estimator.fwire] <=>
    isTrue[Estimator.preF_fwire]
}
pred pref_flight_possible{
  all flight : Bool |
    delta[Estimator.fwire,
    Estimator.preF_fwire,
    Estimator.light,
    flight,
    Estimator.switch,
    Estimator.preF_flight]
}
pred pref_flight_applied{
  isTrue[Estimator.flight] <=>
    isTrue[False]
}
fact prefs {
  always {
    pref_flight_possible implies
                  pref_flight_applied
    pref_fwire_possible implies
                  pref_fwire_applied
  }
}
```

Figure 5: Implementing preferences in ELECTRUM

telling us if a preference γ_i is applicable, that is, if the two valuations for variable $\mathtt{e_i}$ (operator `all`), and at least one possible valuation for variables $\mathtt{e_j}, i < j \leq n$ (operator `some`) are consistent with Δ an the observation. This corresponds to the left part of the quantified form of a preference. Figure 5 illustrates the definition of predicates that implement these conditions with ELECTRUM operators. The rest of the preferences are described by a fact.

The order in which preferences are applied in Γ, is completely induced by the quantification overlays in the quantified forms of preferences (see definitions 8 and 15).

6 Experimental results

We experimented our approach on the model of a multi-robot mission of customizable size and complexity.

In this mission, one or more robots move on a rectangular grid of variable size, each robot moves towards its destination. The destination can change during the mission according to unspecified dynamics.

Behavioral constraints are declared in Δ for each robot. robots should move unless they are at their destination. Moreover, robots may be subject to permanent and / or intermittent faults. An intermittent fault slows down or immobilizes the robot for a few moments which is modelled by the fact that a robot takes several time steps to cross a cell. A permanent fault immobilizes the robot permanently. We also estimate the state of the terrain. A robot crosses a *normal* cell in one time step, but we introduce *difficult* cells (the robot takes several time steps to go through the cell) and *dangerous* cells (robot stays forever on the cell).

We observe at every moment the position of robots on the grid and their destination. We estimate the presence of intermittent and permanent faults on each robot, a variable indicating if it can move, as well as the dangerousness of each grid cell. We implement two estimation strategies, one subject to deadlock, the other not (for example always preferring the absence of a permanent fault).

Models are written in Scala language and are automatically translated into ELECTRUM. ELECTRUM proceeds to the verification through different solvers. For our experiments, we set ELECTRUM to use the Sat4j solver [15]. Verifications were performed on a 3.5GHz Intel Core i5-7600 quad-core processor. Figure 6 shows the results of our experiments for different mission parameters.

The first conclusion is that verification is faster for models that are subject to deadlock scenarios than for those that do not encounter such scenarios. This difference is directly related to the fact that when the model-checker finds a deadlock scenario in the model, the search ends immediately and it returns a counterexample corresponding to the sequence of observations. On the contrary, to prove that a model contains no deadlock scenarios, ELECTRUM explores many more possible executions paths for the verifier.

We can then see that the verification is very fast for the first models, but as we enrich P, Δ et Γ, the number of variables generated for Sat4j and the time required for verification increases exponentially. For examples of larger sizes, it is interesting to note that Sat4j did not have enough memory to process them.

Deadlock detection as proposed in the paper is carried out in the design phase. Hence, high calculation times are not necessarily unacceptable and do not call into question the approach we propose. Let us notice that the results we present here are preliminary results and we are currently working on different ways to improve them. Specifically, we aim to integrate ELECTRUM more closely to avoid the creation of boolean variables where ELECTRUM could handle enumerated variables.

7 Conclusion and future work

We have developed a generic method to validate the proper behavior of a diagnoser at design stage. In particular, we have showed that it is possible to anticipate deadlock situations.

In this paper, we do not introduce any mechanism to identify the causes and modify the estimation model in order to eliminate deadlock scenarios, but this remains a goal for future work.

In addition, we plan to reuse the automation of model-checking allowing us to verify the existence of deadlock scenarios for our diagnosers to check other properties related to diagnosis such as diagnosability.

Finally, progress on model-checker performance is needed to apply it to multi-robot missions involving large models. QBF solvers [16] are potential tools for finding deadlock scenarios of bounded length.

References

[1] Nuno Macedo, Julien Brunel, David Chemouil, Alcino Cunha, and Denis Kuperberg. Lightweight specification and analysis of dynamic systems with rich con-

ID	nb of robots	grid size	$\|\mathtt{P}\|$	$\|\Delta\|$	$\|\Gamma\|$	CNF vars	CNF clauses	deadlock	no deadlock
1	1	1 x 2	13	17	7	1083	1586	0,1s	0,8s
2	1	2 x 2	21	27	11	3400	3925	0,5s	4,8s
3	1	2 x 3	29	37	15	13757	10593	7,3s	52,9s
4	2	2 x 2	34	50	14	12631	13561	6,7s	60,5s
5	2	2 x 3	46	68	18	65 030	47 669	144,6s	1001,4s

Figure 6: ELECTRUM model checking. Columns $\|\mathtt{P}\|$, $\|\Delta\|$ et $\|\Gamma\|$ indicate respectively the number of variables of P, the number of rules of Δ and the number of preferences in Γ. Columns 7 and 8 indicate respectively the number of variables and clauses sent to solver Sat4j by ELECTRUM. The last two columns indicate the verification time for a version of the model subject to a deadlock scenario and another version for which such a scenario does not exist.

figurations. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 373–383, New York, NY, USA, 2016.

[2] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995.

[3] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Preference-based constrained optimization with cp-nets. *Computational Intelligence*, 20(2):137–157, 2004.

[4] Cedric Pralet, Xavier Pucel, and Stéphanie Roussel. Diagnosis of intermittent faults with conditional preferences. In *Proceedings of the 27th International Workshop on Principles of Diagnosis (DX'16)*, 2016.

[5] Xavier Pucel and Stéphanie Roussel. Intermittent fault diagnosis as discrete signal estimation: Trackability analysis. In *Proceedings of the 28th International Workshop on Principles of Diagnosis (DX'17)*, 2017.

[6] James Kurien and P Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 370–377, 2000.

[7] Marie-Odile Cordier, Philippe Dague, François Lévy, Jacky Montmain, Marcel Staroswiecki, and Louise Travé-Massuyès. Conflicts versus analytical redundancy relations: a comparative analysis of the model based diagnosis approach from the artificial intelligence and automatic control perspectives. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(5):2163–2177, 2004.

[8] Alexander Felfernig and Monika Schubert. Fastdiag: A diagnosis algorithm for inconsistent constraint sets. In *Proceedings of the 21st International Workshop on the Principles of Diagnosis (DX 2010), Portland, OR, USA*, pages 31–38, 2010.

[9] Johan De Kleer. Diagnosing multiple persistent and intermittent faults. In *IJCAI*, pages 733–738, 2009.

[10] Olivier Contant, Stéphane Lafortune, and Demosthenis Teneketzis. Diagnosis of intermittent faults. *Discrete Event Dynamic Systems*, 14(2):171–202, 2004.

[11] Craig Boutilier, Ronen I Brafman, Carmel Domshlak, Holger H Hoos, and David Poole. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.(JAIR)*, 21:135–191, 2004.

[12] Shengbing Jiang, Zhongdong Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8):1318–1321, Aug 2001.

[13] Alessandro Cimatti, Edmund Clarke, Fausto Giunchiglia, and Marco Roveri. Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425, Mar 2000.

[14] Daniel Jackson. *Software Abstractions: Logic, Language, and Analysis*. The MIT Press, 2006.

[15] Daniel Le Berre and Anne Parrain. The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[16] Massimo Narizzano, Luca Pulina, and Armando Tacchella. The qbfeval web portal. In *Logics in Artificial Intelligence*, pages 494–497. Springer Berlin Heidelberg, 2006.

Annexe B

Meta-diagnosis via preference relaxation for state trackability

Meta-diagnosis via preference relaxation for state trackability

Valentin Bouziat[1], Xavier Pucel[1], Stéphanie Roussel[1], Louise Travé-Massuyès[2] *
[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

Abstract

In autonomous systems, planning and decision making rely on the estimation of the system state across time, i.e. state tracking. In this work, a preference model is used to provide non ambiguous estimates at each time point. However, this strategy can lead to deadlocks. Our goal is to anticipate deadlock scenarios at design time and to blame a subset of preferences for each of them, so that these preferences can be revised. To do so, we define the preference-based state estimation approach as well as a method for detecting deadlock scenarios, and we apply a consistency-based meta-diagnosis strategy based on preference relaxation. We build upon SAT solvers for the detection of deadlocks and for consistency checking during the meta-diagnosis.

1 Introduction

Intelligent autonomous systems require an elaborate fault management strategy in order to cope with unexpected aleas, both internal (failures, wear) and external (perturbations, environmental changes). While planning and scheduling algorithms can cope to some extent with uncertainty, we focus on approaches that separate, on the one hand, diagnosis and situation assessment, and on the other hand, decision making, and we consider a state estimation technique that produces a unique output, for several reasons. First, it supports integration with deterministic planners and schedulers; second, it scales better; and last, it eases the explanation of the current system status to an operator.

The requirements of our diagnoser are as follows. First, it should handle systems with discrete dynamics, as well as non-determinism due to partial measurements of the system state. Second, it should produce a unique diagnosis at each time step. Third, the diagnoses must be a reliable grounds for decision making. A consequence of the first two requirements is that there are situations where, due to non-determinism, the diagnoser must choose one diagnosis among all those that are consistent with the observations. The diagnosis model that supports this approach, described in section 2, is composed of two parts: a constraint model that defines the possible explanations for a given observed scenario, and a conditional preference model that specifies which diagnosis is preferred, and under which conditions.

As explained in detail in our previous papers (Pralet, Pucel, and Roussel 2016; Pucel and Roussel 2017; Bouziat et al. 2018), this modeling approach is original. It uses a totally ordered conditional preference model (Boutilier et al. 2004a; 2004b) to let the designer select diagnoses. Most diagnosis approaches for discrete-event systems (Sampath et al. 1995; Grastien et al. 2007; Pencolé and Cordier 2005; Jéron et al. 2006; Benveniste et al. 2003; Qiu and Kumar 2006) do not support diagnosis preferences. Neither do intermittent fault diagnosis approaches (Contant, Lafortune, and Teneketzis 2004; De Kleer 2009). Stochastic models support sorting diagnoses by probability (Thorsley and Teneketzis 2005) but transition probabilities are difficult to assess when constructing a robotic system.

As detailed in section 3, these assumptions inevitably allow for situations where the diagnoser has chosen an execution path different from that of the system. This may result in the diagnoser being incapable of providing an diagnosis consistent with the latest system observations, losing track of the system state.Finding such deadlocking paths cannot be done using classical model-checking tools such as NuXMV (Cavada et al. 2014), because the preference model cannot be efficiently translated into a symbolic model. In (Pucel and Roussel 2017) we described a detection approach based on iterative queries to a symbolic model-checker. In some related work (Bouziat et al. 2018), we explore the capabilities of the ELECTRUM model-checker (Macedo et al. 2016). In this paper, we follow a more direct approach: we detect deadlocking paths by enumerating the possible osbervable sequences, and checking if the estimator accepts them.

There are several ways to deal with a deadlock online: backtracking in time and finding an appropriate branching path (Kurien and Nayak 2000) or allowing a transition for the estimator that violates the system model. Both options violate our third requirement: the fact that we use the diagnoses for autonomous and possibly critical decision making means that we do not accept a use case where the diagnoser changes its diagnosis for a previous state, or produces a sequence of diagnoses that is inconsistent with the system model. This requirement does not impose the diagnoser to be strictly correct at all times, however it sets some minimal quality level for the diagnosis.

*Authors are listed in alphabetical order

The contribution of this paper consists in an approach for dealing with deadlocking paths *at design time*. We suggest a set of preferences that, if modified appropriately, will elmiminate the deadlock. The approach implements a consistency-based diagnosis approach over the relaxation of the preferences. Since we use a diagnosis approach to analyse a diagnosis model, we call it meta-diagnosis to avoid ambiguity.

The paper is structured as follows. First, the diagnosis model and the diagnosis algorithm are described (Sect. 2). Second, deadlocking paths are defined (Sect. 3). Then, the semantic for relaxing conditional preferences is defined, and an algorithm for finding minimal relaxations is described (Sec. 4). In Section 5, we describe a realistic robotic functional architecture on which we have tested our approach. Experimental results are presented in Section 6.

2 Diagnosis model

The diagnosis model is a tuple (s_0, Δ, Γ) in which s_0 is the initial system state, Δ the behavioural model and Γ the preference model.

We suppose that the system is governed by discrete dynamics where each time step lasts the same duration. The system, i.e. its internal state, inputs and outputs, are described at a given time step by a finite set of propositional symbols $\mathtt{P}$. A *system state* is an assignment of truth values to $\mathtt{P}$ and we note S the set of all possible system states. $s_0 \in S$ is the initial system state. We suppose that $\mathtt{P}$ is partitioned into two subsets $\mathtt{O}$ and $\mathtt{E}$ that respectively represent the elements of the system that are directly observed and the ones to be estimated. An assignment of truth values to $\mathtt{O}$ is called an *observation*. We note $\mathcal{O}$ the set of possible observations.

Notation For a variable set $\mathtt{X}$ and a variable $\mathtt{x} \in \mathtt{X}$, x (resp. $\overline{x}$) denotes the assignment in which $\mathtt{x}$ is assigned *true* (resp. *false*). For two assignments x and y on the respective variable sets $\mathtt{X}$ and $\mathtt{Y}$, xy is the assignment on $\mathtt{X} \cup \mathtt{Y}$ such that $\forall \mathtt{x} \in \mathtt{X}$, $xy(\mathtt{x}) = x(\mathtt{x})$ and $\forall \mathtt{y} \in \mathtt{Y}$, $xy(\mathtt{y}) = y(\mathtt{y})$.

2.1 Behavioral Model Δ

The behavioral model $\Delta \subseteq \mathcal{S}^2$ is the transition relation of the system. In order to refer to the state of the system at the previous time step, we define a variable set $\mathtt{P}_{\mathtt{pre}} = \{\mathtt{pre\_p} \,|\, \mathtt{p} \in \mathtt{P}\}$ such as $\forall \mathtt{p} \in \mathtt{P}, pre(\mathtt{p}) = \mathtt{pre\_p}$. Δ is represented by a set of propositional logic formulas Δ_p that can relate to both the variables of $\mathtt{P}$ and those of $\mathtt{P}_{\mathtt{pre}}$.

A pair of states (s_{pre}, s_{now}) belongs to the transition relation Δ when the system can be in the state s_{pre} at time $t-1$ and in the state s_{now} at time t. To formally link Δ to Δ_p, for any pair of states (s_{pre}, s_{now}), we define the assignment $\sigma_{s_{pre},s_{now}}$ on variables from $\mathtt{P} \cup \mathtt{P}_{\mathtt{pre}}$ such that $\forall \mathtt{p} \in \mathtt{P}, \sigma_{s_{pre},s_{now}}(\mathtt{p}) = s_{now}(\mathtt{p})$ and $\sigma_{s_{pre},s_{now}}(pre(\mathtt{p})) = s_{pre}(\mathtt{p})$. We consider that a pair of states belongs to the transition relation Δ if and only if the corresponding assignment satisfies the formulas of Δ_p.

Definition 1 (Transition)**.** *A pair of states $(s_{pre}, s_{now}) \in \mathcal{S}^2$ is a transition, denoted $(s_{pre}, s_{now}) \in \Delta$, if and only if $\sigma_{s_{pre},s_{now}} \models \Delta_p$.*

In the remaining of this paper, Δ denotes indifferently the relation and the formula. In order to reason about the possible evolutions of the system state, and the associated observation sequences, we define consistent state sequences and consistent observation sequences as follows.

Definition 2 (Consistent state sequence)**.** *A state sequence $(s_0, s_1, \ldots, s_n) \in \mathcal{S}^n$ is consistent if and only if $\forall i \in [1, n]$, $(s_{i-1}, s_i) \in \Delta$.*

Definition 3 (Consistent observation sequence)**.** *An observation sequence $(o_0, o_1, \ldots, o_n)$ is consistent if and only if there exist a consistent state sequence $(s_0, s_1, \ldots, s_n)$ such that $\forall i \in [0, n], \forall \mathtt{o} \in \mathtt{O}, s_i(\mathtt{o}) = o_i(\mathtt{o})$.*

In the following, when there is no ambiguity, state and observation sequences are assumed to be consistent.

Given a previous state and an observation of the system, we define the set of candidates for diagnosis as the set of states that are consistent with the previous state, the observation and the behavioral model Δ.

Definition 4 (Diagnosis candidates)**.** *The set $\mathcal{S}_\Delta(s_{pre}, o)$ of diagnosis candidates for a previous state $s_{pre} \in \mathcal{S}$ and an observation o is:*

$$\mathcal{S}_\Delta(s_{pre}, o) = \{s_{now} \in \mathcal{S} | \; \forall \mathtt{o} \in \mathtt{O}, s_{now}(\mathtt{o}) = o(\mathtt{o}) \text{ and } (s_{pre}, s_{now}) \in \Delta\}$$

We illustrate this modelling approach on a small model that is a simplified version of the experimental benchmark of section 5.

Example 1 (Behavioural model)**.** *We consider a simple robot functional architecture with three functions: movement, communication and power supply. The health status of each function is represented by the three respective variables $\mathtt{h}_{\mathtt{move}}$, $\mathtt{h}_{\mathtt{com}}$ and $\mathtt{h}_{\mathtt{pow}}$. Two alarms $\mathtt{al}_{\mathtt{move}}$ and $\mathtt{al}_{\mathtt{com}}$ can be raised when movement and communication fail. Diagnosis consists in estimating the value of health variables from the sequence of alarms, i.e. $\mathtt{O} = \{\mathtt{al}_{\mathtt{move}}, \mathtt{al}_{\mathtt{com}}\}$, and $\mathtt{E} = \{\mathtt{h}_{\mathtt{move}}, \mathtt{h}_{\mathtt{com}}, \mathtt{h}_{\mathtt{pow}}\}$.*

Δ is the conjunction of the following formulas:

$$\neg \mathtt{pre\_h}_{\mathtt{pow}} \rightarrow \neg \mathtt{h}_{\mathtt{pow}} \quad (\delta_1)$$
$$\neg \mathtt{h}_{\mathtt{pow}} \rightarrow (\mathtt{al}_{\mathtt{move}} \wedge \mathtt{al}_{\mathtt{com}}) \quad (\delta_2)$$
$$\neg \mathtt{h}_{\mathtt{move}} \rightarrow \mathtt{al}_{\mathtt{move}} \quad (\delta_3)$$
$$\neg \mathtt{h}_{\mathtt{com}} \rightarrow \mathtt{al}_{\mathtt{com}} \quad (\delta_4)$$

Δ expresses that the fault in the power supply is permanent (δ_1), and causes both alarms to be raised (δ_2), movement faults cause movement alarms (δ_3) and communication faults cause communication alarms (δ_4). Note that alarms can also occur without any fault being present (false positives, external perturbations, etc), and that movement and communication faults are intermittent, i.e. they are independent from their previous values.

The initial state is $s_0 = \overline{al_{move}}\ \overline{al_{com}}\ h_{move}\ h_{com}\ h_{pow}$.

At time step 1, let us assume we receive observation $o_1 = al_{move}\ \overline{al_{com}}$. The set of diagnosis candidates is then:

$$\mathcal{S}_\Delta(s_0, o_1) = \left\{ \begin{array}{l} s_{11} = al_{move}\ \overline{al_{com}}\ h_{move}\ h_{com}\ h_{pow}, \\ s_{12} = al_{move}\ \overline{al_{com}}\ \overline{h_{move}} h_{com} h_{pow} \end{array} \right\}$$

In the first explanation, the al_{move} alarm is explained as a false alarm or caused by some unknown event, while in the second it is explained by a fault in the movement module.

2.2 Preference model Γ

The preference model Γ allows the diagnoser to choose one unique diagnosis among the candidates at each time step. In order to formally define Γ, we first introduce the concept of conditional preference and define the order induced by a sequence of preferences.

Informally, a conditional preference relates to a variable to be estimated and indicates under which condition we prefer the diagnoses that assign true of false to this variable to the other diagnoses. A preference is a form of "soft constraint", it is only applied when there exists diagnoses with both values for the variable. Note that it is a simple case of the conditional preference theory ((Wilson 2011)).

Definition 5 (Conditional preference)**.** *A conditional preference γ on a variable* e *of* E*, is denoted $\langle \texttt{cond} : e \prec \overline{e} \rangle$. The preference's condition* cond *is a propositional formula on* $\texttt{P} \cup \texttt{P}_{\texttt{pre}}$*. The variable* e *is called the preference's target.*

For e in E, the conditional preference $\langle \texttt{cond} : e \prec \overline{e} \rangle$ expresses a preference for transitions in which e is *true* if and only if cond is satisfied. Note that preferences $\langle \texttt{cond} : e \prec \overline{e} \rangle$ and $\langle \neg\texttt{cond} : \overline{e} \prec e \rangle$ are equivalent.

Formally, a preference $\gamma = \langle \texttt{cond} : e \prec \overline{e} \rangle$ defines a partial order $\prec_\gamma$ and an equivalence relation $\approx_\gamma$ between transitions the following way: $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) is strictly preferred to (s_{pre}, s') according to γ (denoted $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$) if and only if $\sigma_{s_{pre},s} \models \texttt{cond} \leftrightarrow \texttt{e}$ and $\sigma_{s_{pre},s'} \not\models \texttt{cond} \leftrightarrow \texttt{e}$.

The equivalence relation is defined as follows: (s_{pre}, s) and (s_{pre}, s') are equivalent regarding γ (denoted $(s_{pre}, s) \approx_\gamma (s_{pre}, s')$) if and only if $(\sigma_{s_{pre},s} \models \texttt{cond} \leftrightarrow \texttt{e}) \Leftrightarrow (\sigma_{s_{pre},s'} \models \texttt{cond} \leftrightarrow \texttt{e})$.

Definition 6 (Conditional preference model)**.** *A conditional preference model Γ is a sequence of preferences $(\gamma_1, \gamma_2, \ldots, \gamma_n)$ with $\gamma_i = \langle \texttt{cond}_\texttt{i} : e_i \prec \overline{e_i} \rangle$ for $i \in [1, n]$, such that each variable of* E *is the target of exactly one preference ($n = |\texttt{E}|$), and for all $i \in [1, n]$, the condition $\texttt{cond}_\texttt{i}$ only uses variables from $\texttt{P}_\texttt{pre} \cup \texttt{O} \cup \{\texttt{e}_\texttt{j} | 1 \leq j < i\}$.*

Informally, if γ_i appears before γ_j in Γ, then the condition of γ_j can depend on the outcome of γ_i, but the reverse is forbidden. In other words, we require Γ to be acyclic.

From a conditional preference model Γ, we define a partial order $\prec_\Gamma$ between transitions. Intuitively, as in a lexicographic order, we consider preferences γ_i in their index order in Γ and we apply at each index the order relation $\prec_{\gamma_i}$ defined above. This order is partial because it compares only pairs of transitions (s_{pre}, s_{now}) and $(s_{pre}{}', s_{now}{}')$ that have the same previous state (*i.e.* $s_{pre} = s_{pre}{}'$) and whose successor states produce the same observation (*i.e.* $\forall \texttt{o} \in \texttt{O} s_{now}(\texttt{o}) = s_{now}{}'(\texttt{o})$). Formally, $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) is strictly preferred to (s_{pre}, s') according to Γ (denoted $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$) if and only if there exists $i \in [1, n]$ such that for all $j < i$, $(s_{pre}, s) \approx_{\gamma_j} (s_{pre}, s')$ and $(s_{pre}, s) \prec_{\gamma_i} (s_{pre}, s')$.

Example 2 (Preference model)**.** *We enrich the behavioural model Δ from example 1 with a preference model that implements the following strategy. If both alarms are raised simultaneously, we blame their common cause, i.e. the power supply. Otherwise we blame the respective functions. Moreover, for the communication function, we dismiss the first alarm as noise, and only diagnose a communication fault when the alarm persists during several time steps. This strategy is implemented by the following preferences:*

$$\langle \begin{array}{r} \neg\texttt{pre\_al}_\texttt{move} \wedge \texttt{al}_\texttt{move} \\ \wedge \neg\texttt{pre\_al}_\texttt{com} \wedge \texttt{al}_\texttt{com} \end{array} : \overline{h_{pow}} \prec h_{pow} \rangle \qquad (\gamma_1)$$

$$\langle \texttt{al}_\texttt{move} \wedge \texttt{h}_\texttt{pow} : \overline{h_{move}} \prec h_{move} \rangle \qquad (\gamma_2)$$

$$\langle \texttt{pre\_al}_\texttt{com} \wedge \texttt{al}_\texttt{com} \wedge \texttt{h}_\texttt{pow} : \overline{h_{com}} \prec h_{com} \rangle \qquad (\gamma_3)$$

We define Γ as the ordered sequence $(\gamma_1, \gamma_2, \gamma_3)$.

The two transitions (s_0, s_{11}) and (s_0, s_{12}) ,where s_{11} and s_{12} are the two states of $\mathcal{S}_\Delta(s_0, o_1)$ defined in Example 1 can be compared using $\prec_\Gamma$. We have $(s_0, s_{11}) \approx_{\gamma_1} (s_0, s_{12})$ as s_{11} and s_{12} have the same valuation for $\texttt{h}_\texttt{pow}$. $(s_0, s_{12}) \prec_{\gamma_2} (s_0, s_{11})$ as the condition preference is satisfied by $\sigma_{(s_0,s_{12})}$ and $\texttt{h}_\texttt{move}$ is false in s_{12}. Thus, we have $(s_0, s_{12}) \prec_\Gamma (s_0, s_{11})$.

2.3 State tracking

State tracking is performed step by step as new observations are acquired from the system. It relies on the system's behavioral model Δ and applies the preference model Γ when Δ would leave several possible diagnoses. We first show that the preference model structure guarantees that the preferred transition is always unique. State tracking is achieved by computing the preferred transition at each time step.

Proposition 1. *Let s_{pre} be a state and o an observation, let Δ be a behavioral model, let $s, s' \in \mathcal{S}_\Delta(\hat{s}_{pre}, o)^2$ be such that $s \neq s'$, and let Γ be a conditional preference model.We have either $(s_{pre}, s) \prec_\Gamma (s_{pre}, s')$ or $(s_{pre}, s') \prec_\Gamma (s_{pre}, s)$.*

Sketch of Proof Since $s \neq s'$, there exists a preference $\gamma \in \Gamma$ such that $s \approx_\gamma s'$ does not hold. We have either $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$ or $(s_{pre}, s') \prec_\gamma (s_{pre}, s)$. □

Definition 7 (Diagnosis step)**.** *The diagnosis $\hat{s} = diagnose(s_{pre}, o)$ for a previous state s_{pre} and an observation o is the element of $\mathcal{S}_\Delta(s_{pre}, o)$ preferred by $\prec_\Gamma$. Formally, $\hat{s} = diagnose(s_{pre}, o)$ if and only if:*

1. *$\hat{s} \in \mathcal{S}_\Delta(s_{pre}, o)$, and*
2. *$\nexists s_{best} \in \mathcal{S}_\Delta(s_{pre}, o)$ such that $(s_{pre}, s_{best}) \prec_\Gamma (s_{pre}, \hat{s})$.*

By proposition 1, for a given pair (s_{pre}, o), the set of candidates $\mathcal{S}_\Delta(s_{pre}, o)$ is totally ordered by $\prec_\Gamma$. Thus, there exists a unique solution to the diagnosis step, that is the *diagnosis*.

Definition 8 (Diagnosis sequence)**.** *Let $(o_0, o_1, \ldots, o_k)$ be a consistent observation sequence. The diagnosis sequence $diagnose\_seq\big((o_0, o_1, \ldots, o_k)\big)$ is the state sequence $(s_0, \hat{s}_1, \hat{s}_2, \ldots, \hat{s}_k)$ such that $\forall i \in [1, k], \hat{s}_i = diagnose(\hat{s}_{i-1}, o_i)$.*

Example 3 (Diagnosis sequence)**.** *Following Examples 1 and 2, diagnose_seq* $\big((o_0, o_1)\big) = (s_0, \hat{s}_1)$*, where* $\hat{s}_1 = s_{12}$.

The problem of diagnosis at each time step consists in finding a valuation of E that satisfies Δ, consistent with the current observation and the previous diagnosis, and that is optimal with respect to Γ. It can be reduced to a boolean optimization problem, and in particular to a (weighted partial) MAX-SAT query as described in (Pucel and Roussel 2017).

3 Deadlocks

3.1 Definition

Often there are several estimation candidates, because an observation can be explained by several system states. As a consequence, the diagnoser may select a diagnosis that differs from the system's actual state. We call this phenomenon a *divergence*. In some cases, divergences solve themselves as more observations are received from the system, and the diagnoser converges to the correct diagnosis. However, it can happen that an observation is inconsistent with the diagnosis previously selected by the diagnoser. In this case, the set of diagnosis candidates is empty and the diagnoser cannot produce any output. We call this situation a *deadlock*, and the sequence of observations that led the diagnoser into it is called a *deadlocking path*.

Definition 9 (Deadlocking path)**.** *A* deadlocking path *for an estimation model* (s_0, Δ, Γ) *is a consistent observation sequence* $(o_1, o_2, \ldots, o_k)$ *with* $k > 1$ *such that there exists a diagnosis sequence for* $(o_1, o_2, \ldots, o_{k-1})$ *but there does not exist a diagnosis sequence for* $(o_1, o_2, \ldots, o_k)$[1].

Example 4 (Deadlocking path)**.** *In the system described in Examples 1 and 2, let us consider the diagnosis scenario depicted in Figure 1. At time step 0, the diagnoser is correctly initialized.*

At time step 1, two faults occur simultaneously in the movement and communication functions in the system, causing both alarms to activate. The diagnoser explains it with a fault in the battery, due to preference (γ_1)*, which constitutes a* divergence *between the real system and the diagnosis.*

At time step 2, the faults disapear in the real system, causing both alarms to stop. The assignment $\sigma_{(s_1,s_2)}$ *must be such that:* $\texttt{pre\_h}_{\texttt{pow}}$ *is true,* $\texttt{al}_{\texttt{move}}$ *and* $\texttt{al}_{\texttt{com}}$ *are false, and rules* (δ_1) *and* (δ_2) *must be satisfied. Since no assignment can satisfy all these, the set of diagnosis candidates is empty and we have a deadlock.*

3.2 Detecting deadlocks

In (Bouziat et al. 2018), we have built a specific transition system called the *verifier*, and used the model-checker ELECTRUM to find deadlocking paths. The main advantage of that approach is the possibility to prove the absence of deadlocks of any length. However, this is still work in progress especially with respect to scalability.

[1]There cannot be any deadlock at the first diagnosis step (for $k = 1$) because the initial diagnosis is correctly initialized at s_0.

	SYS			OBS		DIAG		
Time	$\texttt{h}_{\texttt{move}}$	$\texttt{h}_{\texttt{com}}$	$\texttt{h}_{\texttt{pow}}$	$\texttt{al}_{\texttt{move}}$	$\texttt{al}_{\texttt{com}}$	$\texttt{h}_{\texttt{move}}$	$\texttt{h}_{\texttt{com}}$	$\texttt{h}_{\texttt{pow}}$
0	T	T	T	F	F	T	T	T
1	F	F	T	T	T	T	T	F
2	T	T	T	T	T	–	–	–

Figure 1: Deadlock scenario for the model of Examples 1 and 2. The SYS part contains the values of the variables of E in the real system. The OBS part shows the values of the variables of O, shared between the system and the diagnoser. The DIAG part shows the variables of E in the diagnoser.

In this paper, we follow a two-step approach based on unfolding the model and using SAT solvers to detect deadlocking paths of some predefined length. Contrary to the approach of (Bouziat et al. 2018), we cannot prove the absence of deadlocks in general but our bounded approach scales up to large benchmarks. It allows to find all the deadlocking paths of a given length, or prove their absence.

Let k denote the maximum length for the deadlocking paths we are looking for. Unfolding the model starts with creating variables associated with each time step, from 0 to k. More precisely, for each variable p in P and for each time step t in $[0, k]$, we create a variable $\texttt{p}^t$ representing the variable p at time step t. For a formula f over variables of P and $\texttt{P}_{\texttt{pre}}$ and a time step t in $[1, k]$, we define the instantiation of the formula f at t, denoted f^t, by syntactically replacing in f all variables p of P by $\texttt{p}^t$ and each variable $pre(\texttt{p})$ of $\texttt{P}_{\texttt{pre}}$ by $\texttt{p}^{t-1}$. At time step 0, each variable $\texttt{p}^0$ of P is replaced by its value in s_0. We instantiate the formula Δ for each time step $t > 0$ into the formula Δ^t. In this setting, we can easily represent consistent state and observation sequences.

Proposition 2 (Unfolded consistent sequences)**.** *Let* u *be an assignment to all the unfolded variables* $\{\texttt{p}^t | t \in [0, k]\}$*. It represents the state sequence* $(s_0, s_1, \ldots, s_k)$ *such that* $u(\texttt{p}^t) = s_t(\texttt{p})$ *for* $\texttt{p} \in \texttt{P}$ *and* $t \in [0, k]$.

u *satisfies* $\Delta^1 \wedge \ldots \wedge \Delta^k$ *if and only if* u *represents a consistent state sequence (as of Definition 2).*

Let v *be an assignment to the observable unfolded variables* $\{\texttt{o}^t | t \in [0, k], \texttt{o} \in \texttt{O}\}$*. It represents the observation sequence* $(o_0, \ldots, o_k)$ *such that* $v(\texttt{o}^t) = o_t(\texttt{o})$ *for* $\texttt{o} \in \texttt{O}$ *and* $t \in [0, k]$.

v *is consistent with* $\Delta^1 \wedge \ldots \wedge \Delta^k$ *if and only if* v *represents a consistent observation sequence (as of Definition 3).*

Sketch of Proof This proposition is quite easily deduced from Def. 1, 2 and 3, and from the fact that syntactically renaming a variable does not impact the satisfiability of a formula.

Our algorithm relies on an existing diagnoser implementation to check if a given consistent observation sequence is a deadlock, such as the MAX-SAT implementation described in (Pucel and Roussel 2017). We wrap this implementation in a `isADeadlock` function that simply indicates whether the diagnoser can or cannot produce a diagnosis sequence for a given diagnosis model (s_0, Δ, Γ) and observa-

tion sequence $seqObs$.

Algorithm 1 finds one deadlock with maximum length k. It relies on SAT model enumeration (Morgado and Marques-Silva 2005) to produce all the consistent observation sequences for Δ and checks each observation sequence with the `isADeadlock` function. For each time step t in increasing order, we define the formula $toCheck$ as the conjunction of all formulas Δ^i for $i \in [1, t]$.

We enumerate all the submodels of the formula $toCheck$ for the observable variables, *i.e.* the models in which only observable variables are retained (line 3). For each sequence of observations, if it is a deadlocking path (line 5), then we return it (line 6). Else, there is no deadlock with length less or equal to k (line 7).

Algorithm 1: `FindOneDL`$(k, \{\Delta^t | t \in [1,k]\}, s_0, \Delta, \Gamma)$: returns a deadlocking path of length less or equal to k

```
1 for t ← 1 : k do
2     toCheck ← ⋀_{i=1}^{t} Δ^i
3     subM ← subModels(toCheck, {o^i | i ∈ [1,t], o ∈ O})
4     for seqObs ← subM do
5         if isADeadlock(s_0, Δ, Γ, seqObs) then
6             return seqObs
7 print("No DL ≤ k found")
```

Note that this algorithm can easily be extended in order to find all deadlocking paths of length less or equal to k. This is out of scope for this paper.

This algorithm is quite simplistic but has satisfying performance, as detailed in section 6. However, the main contribution of this paper is to blame a set of preferences for a given deadlocking path.

4 Consistency based diagnosis of preferences

In this diagnosis approach, one can consider that a deadlocking path exists because the preference model is inappropriate. In particular, some deadlocks can be eliminated by modifying the preference conditions in Γ. We address the problem of assessing whether a given deadlock can be eliminated in such a way with a consistency based diagnosis approach (Hamscher and others 1992). More precisely, given a deadlocking path and a subset of preferences, we want to know whether the diagnoser would accept the deadlocking path if the given preferences were "relaxed". When this is the case, we can indicate to the designer that the deadlock may be eliminated by modifying the conditions for this particular set of preferences. Correcting the preference conditions to eliminate deadlock is left for future work.

We first describe the semantics for relaxing a set of preferences from Γ, then provide a definition of a consistency-based diagnosis of a set of preferences. For this, we generalise the concept of conditional preference to that of (general) preference as follows.

4.1 Relaxed preference model

For e in E, a *relaxed preference* $\gamma^{\nsim}$ declares that no valuation of e can be preferred in any context, they are incomparable.

Definition 10 (Relaxed preference)**.** *A relaxed preference for* e *in* E *(the target of the relaxed preference), denoted* $\gamma^{\nsim}$*, has the form* $\langle e \nsim \overline{e} \rangle$.

We define a (general) preference as follows.

Definition 11 (Preference)**.** *A preference for* $\mathtt{e} \in \mathtt{E}$*, denoted* φ*, is either a conditional preference* $\langle \mathtt{cond} : e \prec \overline{e} \rangle$ *or a relaxed preference* $\langle e \nsim \overline{e} \rangle$.

Preferences generalize conditional preferences, in that they induce a possibly weaker ordering on transitions. For each preference φ targeting a variable e in E, the associated partial order is defined as follows: $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) is strictly preferred to (s_{pre}, s') according to φ (denoted $(s_{pre}, s) \prec_\varphi (s_{pre}, s')$) if and only if φ is a conditional preference γ and $(s_{pre}, s) \prec_\gamma (s_{pre}, s')$.

The associated equivalence relation is also different. Let φ be a preference targeting a variable e in E, $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) and (s_{pre}, s') are equivalent according to φ (denoted $(s_{pre}, s) \approx_\varphi (s_{pre}, s')$) if and only if φ is a conditional preference γ and $(s_{pre}, s) \approx_\gamma (s_{pre}, s')$, or if φ is a relaxed preference and $s(\mathtt{e}) = s'(\mathtt{e})$.

Intuitively, replacing a conditional preference with a relaxed preference tends to remove pairs from both the partial order relation and the equivalence relation. It creates incomparable pairs of transitions.

Definition 12 (Relaxed preference model)**.** *Given a conditional preference model* $\Gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$ *and a subset*[2] $\Omega \subseteq \Gamma$*, a* relaxed preference model Γ_Ω *is a sequence of preferences* $(\varphi_1, \varphi_2, \ldots, \varphi_n)$ *such that* $\varphi_i = \gamma_i$ *if* $\gamma_i \notin \Omega$*, and* $\varphi_i = \langle e_i \nsim \overline{e_i} \rangle$ *otherwise.*

Note that $\Gamma_\emptyset = \Gamma$ *is a conditional preference model.*

From a relaxed preference model Γ_Ω, we define the partial order $\prec_{\Gamma_\Omega}$ between pairs of states in the same way as for conditional preference models: $\forall s_{pre}, s, s' \in \mathcal{S}^3$, (s_{pre}, s) is strictly preferred to (s_{pre}, s') according to Γ_Ω (denoted $(s_{pre}, s) \prec_{\Gamma_\Omega} (s_{pre}, s')$) if and only if there exists $i \in [1, n]$ such that for all $j < i$, $(s_{pre}, s) \approx_{\varphi_j} (s_{pre}, s')$ and $(s_{pre}, s) \prec_{\varphi_i} (s_{pre}, s')$.

We now generalize the notions of diagnosis (Definitions 7 and 8) to relaxed preference models. In this process, we lose the property of Proposition 1: in a relaxed model, for a given previous state and an observation, the "preferred" transition is not always unique. We prefer to speak of *non-dominated* transitions when a transition has no lesser element in $\prec_{\Gamma_\Omega}$. In fact, by relaxing preferences, one simply creates non-dominated transitions, as stated in the following proposition.

Proposition 3 (Preference relaxation)**.** *Let* (s_0, Δ, Γ) *be a diagnosis model, and let* $\Gamma_{\Omega 1}$ *and* $\Gamma_{\Omega 2}$ *be two relaxations of* Γ *such that* $\Omega 1 \subseteq \Omega 2$*. If a transition is non-dominated in* $\Gamma_{\Omega 1}$*, then it is also non-dominated in* $\Gamma_{\Omega 2}$*:*

$$\forall (s_{pre}, s) \in \mathcal{S}^2, \forall o \in \mathcal{O},$$
$$\left(\nexists s_{best} \in \mathcal{S}_\Delta(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)\right)$$
$$\Rightarrow \left(\nexists s_{best} \in \mathcal{S}_\Delta(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)\right)$$

[2] When there is no ambiguity, we consider Γ as a set of preferences.

Sketch of Proof We prove that if there was a dominating state s_{best} such that $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)$, then we would have $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)$. Let $\mathtt{e}$ be the first variable (by preference ordering) such that $s_{best}(\mathtt{e}) \neq s(\mathtt{e})$ and let $\gamma \in \Gamma$ be the associated conditional preference. Then, for each preference γ' prior to γ in Γ, (s_{pre}, s_{best}) and (s_{pre}, s) are equivalent. Let φ_1 (resp. φ_2) be the preferences targetting $\mathtt{e}$ in $\Gamma_{\Omega 1}$ (resp. $\Gamma_{\Omega 2}$). Then, we show that φ_1 and φ_2 are either both conditional or both relaxed. As a consequence, $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 2}} (s_{pre}, s)$ is equivalent to $(s_{pre}, s_{best}) \prec_{\Gamma_{\Omega 1}} (s_{pre}, s)$. □

As a result, in a relaxed preference model, the diagnosis step should return not a unique state, but rather a set of states corresponding to the set of non-dominated transitions.

Definition 13 (Diagnosis step for Γ_Ω)**.** *Let Δ be a transition relation, Γ_Ω a relaxed preference model, $\hat{s}_{pre} \in \mathcal{S}$ a previous diagnosis and o an observation, a* diagnosis step *consists in finding the set $\mathcal{S}_\Delta^{\Gamma_\Omega}(s_{pre}, o) \subseteq \mathcal{S}_\Delta(s_{pre}, o)$ containing states that are non-dominated in $\prec_{\Gamma_\Omega}$. Formally:*

$$\mathcal{S}_\Delta^{\Gamma_\Omega}(s_{pre}, o) = \{\hat{s} \in \mathcal{S}_\Delta(s_{pre}, o) | \nexists s_{best} \in \mathcal{S}_\Delta(s_{pre}, o), (s_{pre}, s_{best}) \prec_{\Gamma_\Omega} (s_{pre}, \hat{s})\}$$

Since at each time step, the output of the diagnosis step for Γ_Ω is not unique, then for a given sequence of observations, the diagnosis sequence for Γ_Ω is not unique either.

Definition 14 (Diagnosis sequence for Γ_Ω)**.** *Given an initial state $s_0 \in \mathcal{S}$, a transition relation Δ, a relaxed preference model Γ_Ω, diagnosis sequences are defined as follows:*

- *(s_0) is the only diagnosis sequence for the empty observation sequence $()$*
- *$(s_0, \hat{s}_1, \ldots, \hat{s}_k)$ is a diagnosis sequence for the observation sequence $(o_1, \ldots, o_k)$ if and only if $(s_0, \hat{s}_1, \ldots, \hat{s}_{k-1})$ is a diagnosis sequence for $(o_1, \ldots, o_{k-1})$ and $\hat{s}_k \in \mathcal{S}_\Delta^{\Gamma_\Omega}(\hat{s}_{k-1}, o_k)$*

A corrolary of Proposition 3 is that for a given observation sequence, for two sets of preferences $\Omega 1 \subseteq \Omega 2 \subseteq \Gamma$, $\Gamma_{\Omega 2}$ accepts a superset of diagnosis sequences with respect to $\Gamma_{\Omega 1}$. This is why we talk about preference *relaxation*. As a consequence, if we consider an observation sequence that is a deadlocking path for $\Gamma_{\Omega 1}$, $\Gamma_{\Omega 2}$ may have a diagnosis sequence for it.

Example 5 (Deadlocking path elimination)**.** *Let us consider the observation sequence $dl = (\overline{al_{move}}\ \overline{al_{com}},\ al_{move}\ al_{com},\ \overline{al_{move}}\ \overline{al_{com}})$. We have seen in Example 4 that dl is a deadlock for the diagnosis model (s_0, Δ, Γ). It is thus a deadlock for the relaxed model $\Gamma_\emptyset$ as well.*

Let us consider the relaxed model $\Gamma_{\{\gamma_1,\gamma_2\}} = (\langle \overline{h_{pow}} \nsim h_{pow} \rangle, \langle \overline{h_{move}} \nsim h_{move} \rangle, \langle \mathtt{pre\_al_{com}} \wedge \mathtt{al_{com}} \wedge \mathtt{h_{pow}} : \overline{h_{com}} \prec h_{com} \rangle)$ that relaxes the preferences targetting $\mathtt{h_{pow}}$ and $\mathtt{h_{move}}$.

At time step 1, the set of diagnosis candidates $\mathcal{S}_\Delta(s_0, al_{move}\ al_{com})$ contains all the possible combinations for the 3 health variables. Among them, in $\Gamma_{\gamma_1,\gamma_2}$, there are four non-dominated states that correspond to the four valuations of variables $\mathtt{h_{pow}}$ and $\mathtt{h_{move}}$, whose preferences have been relaxed.

$$\begin{array}{ll} al_{move}\, al_{com}\, \overline{h_{pow}}\, \overline{h_{move}}\, h_{com} & (s_{11}) \\ al_{move}\, al_{com}\, \overline{h_{pow}}\, h_{move}\, h_{com} & (s_{12}) \\ al_{move}\, al_{com}\, h_{pow}\, \overline{h_{move}}\, h_{com} & (s_{13}) \\ al_{move}\, al_{com}\, h_{pow}\, h_{move}\, h_{com} & (s_{14}) \end{array}$$

At time step 2, with observation $\overline{al_{move}}\ \overline{al_{com}}$, from s_{11} and s_{12}, there is no diagnosis candidate:

$$\mathcal{S}_\Delta(s_{11}, \overline{al_{move}\, al_{com}}) = \mathcal{S}_\Delta(s_{12}, \overline{al_{move}\, al_{com}}) = \emptyset$$

However, from either s_{13} or s_{14} there is only one single candidate:

$$\mathcal{S}_\Delta(s_{11}, \overline{al_{move}\, al_{com}}) = \mathcal{S}_\Delta(s_{12}, \overline{al_{move}\, al_{com}}) = \{s_2\}$$

With $s_2 = \overline{al_{move}}\ \overline{al_{com}}\ h_{pow}\ h_{move}\ h_{com}$. As the unique candidate, s_2 is automatically non-dominated. We conclude that there are two diagnosis sequences for $\Gamma_{\{\gamma_1,\gamma_2\}}$ and the observation sequence dl. These sequences are (s_0, s_{13}, s_2) and (s_0, s_{14}, s_2). Hence, the observation sequence dl is a deadlock for Γ but not for $\Gamma_{\{\gamma_1,\gamma_2\}}$.

4.2 Consistency-based preference diagnosis

Checking whether a relaxed preference model accepts some diagnosis sequences for a given observation sequence can be done by a series of consistency checks described in this section. Then, searching for the smallest set(s) of preferences that eliminate a deadlock can be done with a classical consistency-based diagnosis algorithm (Reiter 1987).

Definition 15 (Preference Meta-Diagnosis)**.** *Let (s_0, Δ, Γ) be a diagnosis model, and $(o_1, \ldots, o_k)$ a deadlocking path for this model. A set of preferences $\Omega \subseteq \Gamma$ is a* preference meta-diagnosis *if and only if there exists a diagnosis sequence for the relaxed model $(s_0, \Delta, \Gamma_\Omega)$.*

A meta-diagnosis Ω is a minimal meta-diagnosis *if and only if there is no meta-diagnosis $\Omega' \subseteq \Gamma$ such that $\Omega' \subset \Omega$.*

A meta-diagnosis Ω is interpreted as follows: it is possible to modify the conditions for the preferences in Ω so that the diagnoser will not deadlock on the associated observation sequence. It does not guarantee anything with respect to other potential deadlocking paths. Due to Proposition 3, if Ω is a meta diagnosis, then all supersets of Ω are meta-diagnoses as well.

Example 6 (Minimal meta-diagnosis)**.** *For the observation sequence dl described in Example 4, we have seen in Example 5 that it is a deadlocking path for Γ and $\Gamma_\emptyset$, but not for the relaxed model $\Gamma_{\{\gamma_1,\gamma_2\}}$. Thus, $\{\gamma_1, \gamma_2\}$ is a preference meta-diagnosis for this deadlock. However, it is not minimal.*

In fact, for this observation sequence, the relaxed model $\Gamma_{\{\gamma_1\}}$ accepts a diagnosis sequence, while $\Gamma_{\{\gamma_2,\gamma_3\}}$ deadlocks. Consequently, there is a unique minimal meta-diagnosis, $\{\gamma_1\}$. To eliminate the deadlocking path dl, the designer must modify the condition of preference γ_1.

For example, the preference model $\Gamma' = (\gamma_1', \gamma_2, \gamma_3)$ with $\gamma_1' = \langle \top : h_{pow} \prec \overline{h_{pow}} \rangle$ does not deadlock for the observation sequence dl. However it implements a very different fault management strategy, that must ba validated against the robotic mission requirements.

To check whether a set of preferences Ω is a meta-diagnosis, we perform a series of SAT queries that recreate the possible non-dominated diagnosis candidates at each time step as described in Algorithms 2 and 3. We adopt the unfolded model approach described in section 3, and unfold the model on the number k of time steps of the observation sequence dl.

We use a data structure named *scenario* to represent a partially constructed diagnosis sequence. A *scenario* is an object with 3 attributes: *decisions* is a formula that represents the diagnosis decisions made so far, $step \in [1, k]$ is a time step, and *pref* is the next preference to apply. The algorithm uses a work list of *scenario* objects, and each iteration consists in constructing the successor(s) of one *scenario*. The `incr` function mutates the *pref* attribute to the next preference if there is one, otherwise it increases the *step* attribute and resets *pref* to the first preference. The *decisions* attribute of the successor(s) aggregates that of the current *scenario* object with the diagnosis decision dictated by the current *pref*. When *pref* is relaxed, there are two possible decisions, which gives two successors.

Each *scenario* object is tested for consistency with the model Δ and dl up to the current time step. Δ is represented in the unfolded way by $\Delta^0, \ldots, \Delta^k$, and the deadlock is represented at each time step by assignments $dl^0, \ldots, dl^k$. *scenario* objects whose *decision* field is inconsistent with those are discarded. The algorithm terminates when a complete *scenario* object is found ($step > k$, Ω is a meta-diagnosis) or when no *scenario* object remains (Ω is not a meta-diagnosis).

Algorithm 2: `isAMetaDiagnosis`($\{dl^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$): returns true if and only if Ω is a meta-diagnosis

```
1  scenarios ← {(⊤, 1, γ1)}
2  while scenarios ≠ ∅ do
3      select curSc in scenarios
4      scenarios ← scenarios − {curSc}
5      if curSc.step > length(dl) then
6          return true
7      else
8          nextSc ← makeNextScenarios(curSc,
9                       {dl^t | t ∈ [1,k]}, {Δ^t | t ∈ [1,k]}, Γ_Ω)
10         scenarios ← scenarios ∪ nextSc
11 return false
```

The order in which Ω sets should be tested is dictated by the classical consistency-based diagnosis strategy from (Reiter 1987). If we find a meta-diagnosis we prune its supersets, and if we find a non-meta-diagnosis, we prune its subsets.

5 Experiments

We have experimented our approach on a realistic functional robotic architecture. We have considered two versions of this architecture: the first remains quite small and refines the model of Examples 1 to 6 with even more complex behaviour. The second one contains additional functions and more complex behaviour. We look for deadlocking paths in each model, and enumerate the minimal meta-diagnoses for them.

Algorithm 3: `makeNextScenarios`($curSc, \{dl^t | t \in [1,k]\}, \{\Delta^t | t \in [1,k]\}, \Gamma_\Omega$): returns next scenarios associated with $curSc$

```
1  t ← curSc.step;   γ ← curSc.pref
2  form ← curSc.assign ;   scenarios ← ∅
3  if γ = e ⇝ ē then
4      curScBis ← copy(incr(curSc))
5      curSc.assign ← form ∧ e^t
6      curScBis.assign ← form ∧ ¬e^t
7      if isSAT((⋀_{i=1}^{t} (Δ^i ∧ dl^i)) ∧ form ∧ e^t) then
8          scenarios ← scenarios ∪ {curSc}
9      if isSAT((⋀_{i=1}^{t} (Δ^i ∧ dl^i)) ∧ form ∧ ¬e^t) then
10         scenarios ← scenarios ∪ {curScBis}
11     return scenarios
12 else //γ = cond : e ≺ ē
13 if isSAT(⋀_{i=1}^{t} (Δ^i ∧ dl^i)) ∧ form ∧ (e^t ↔ cond^t)) then
14     curSc.assign ← form ∧ (e^t ↔ cond^t)
15     return {incr(curSc)}
16 else
17     curSc.assign ← form ∧ (¬e^t ↔ cond^t)
18     return {incr(curSc)}
```

5.1 Simple architecture

We consider the same three functions as in Example 1. However, we explicitly account for a number of possible causes for the alarms: environmental perturbations that impede movement and communications (obstacles, slippery terrain, distance to antenna, etc), represented respectively by variables $\texttt{fenv}_{\texttt{move}}$ and $\texttt{fenv}_{\texttt{com}}$, and a low voltage in the power supply, represented by $\texttt{low}_{\texttt{pow}}$. Our goal is to estimate whether each function is available for autonomous operation. Rather than the real physical health status of the functions, we estimate the trust we put in them. Variables $\texttt{t}_{\texttt{move}}$, $\texttt{t}_{\texttt{com}}$, and $\texttt{t}_{\texttt{pow}}$ represent that we still trust each respective function for autonomous operation. The estimated value for these variables is sent to the task planner. Figure 2 details the model and illustrates this architecture.

Δ represents the following behaviour. We cannot trust the movement and communication functions unless we trust the power supply function as well (δ_1). Movement perturbations or low voltage cause a movement alarm (δ_2), the same goes for communication (δ_3). Once we lose trust in the power supply, we never trust it again (δ_4). The communication alarm is a perfect indicator of our trust in the communication function (δ_5).

Rules (δ_1) and (δ_4) do not represent the actual behaviour of the robot, they enforce an interface contract between the diagnoser and the task planner. The diagnoser is required to never publish diagnosis sequences that violate these rules.

In the preferences associated with this model, we use the temporal logic operators from PtLTL to express formula more compactly. As shown in (Havelund and Rosu 2002), one can efficiently translate a PtLTL formula into a propo-

$$(\mathtt{t}_{\mathtt{move}} \vee \mathtt{t}_{\mathtt{com}}) \rightarrow \mathtt{t}_{\mathtt{pow}} \quad (\delta_1)$$

$$(\mathtt{fenv}_{\mathtt{move}} \vee \mathtt{low}_{\mathtt{pow}}) \rightarrow \mathtt{al}_{\mathtt{move}} \quad (\delta_2)$$

$$(\mathtt{fenv}_{\mathtt{com}} \vee \mathtt{low}_{\mathtt{pow}}) \rightarrow \mathtt{al}_{\mathtt{com}} \quad (\delta_3)$$

$$\neg \mathtt{pre\_t}_{\mathtt{pow}} \rightarrow \neg \mathtt{t}_{\mathtt{pow}} \quad (\delta_4)$$

$$\mathtt{t}_{\mathtt{com}} \leftrightarrow \neg \mathtt{al}_{\mathtt{com}} \quad (\delta_5)$$

$$\neg \mathtt{pre\_al}_{\mathtt{move}} \wedge \mathtt{al}_{\mathtt{move}} \wedge \neg \mathtt{pre\_al}_{\mathtt{com}} \wedge \mathtt{al}_{\mathtt{com}} : \mathit{low}_{pow} \prec \overline{\mathit{low}_{pow}} \quad (\gamma_1)$$

$$\mathbf{O}(\mathbf{H}_{\kappa}(\mathtt{low}_{\mathtt{pow}})) : \overline{t_{pow}} \prec t_{pow} \quad (\gamma_2)$$

$$\mathbf{H_2}(\mathtt{al}_{\mathtt{move}}) \wedge \neg \mathtt{low}_{\mathtt{pow}} : \mathit{fenv}_{move} \prec \overline{\mathit{fenv}_{move}} \quad (\gamma_3)$$

$$\mathbf{H_4}(\mathtt{al}_{\mathtt{move}}) : t_{move} \prec \overline{t_{move}} \quad (\gamma_4)$$

$$\mathtt{al}_{\mathtt{com}} \wedge \neg \mathtt{low}_{\mathtt{pow}} : \mathit{fenv}_{com} \prec \overline{\mathit{fenv}_{com}} \quad (\gamma_5)$$

$$\top : t_{com} \prec \overline{t_{com}} \quad (\gamma_5)$$

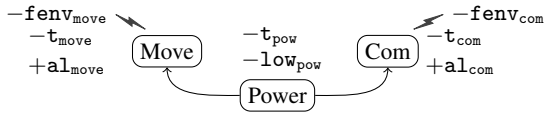


Figure 2: Δ, Γ and schema of the simple architecture model. Variables labelled with $+$ are observable, $-$ estimated. Arrows represent functional dependency.

sitional logic formula. We use the operator $\mathbf{H}_{\kappa}$ with $\kappa > 0$. $\mathbf{H}_{\kappa}(\mathtt{f})$ is true if and only if the formula $\mathtt{f}$ has been true for the last κ time steps, including now. For example $\mathbf{H_2}(\mathtt{f})$, we introduce a new variable $\mathtt{alias\_f}$. We replace any occurrence of $\mathbf{H_2}(\mathtt{f})$ by $\mathtt{alias\_f} \wedge \mathtt{pre\_alias\_f}$ and add the formula $\mathtt{alias\_f} \leftrightarrow \mathtt{f}$ to Δ. The new variable needs to estimated, however its value is totally determined by that of the other variables, thus, provided its preference is placed last in Γ the condition does not matter.

We also use the once operator: $\mathbf{O}(\mathtt{f})$ is true if and only if formula $\mathtt{f}$ has been true at least once in the experiment (including the current time step).

Γ is the ordered sequence $(\gamma_1, \gamma_2, \gamma_3, \gamma_4, \gamma_5, \gamma_6)$ and implements the following strategy. We favour low voltage if and only if both alarms fire simultaneously (γ_1). If there has been a continuous period of size κ of low voltage, we lose trust in the power supply (γ_2). When a movement alarm not explained by low voltage is on for two time steps, we blame the associated environmental perturbations (γ_3). We trust the movement function when there has been no movement alarm for 4 time steps (γ_4). Communication alarms not explained by low voltage are explained by environmental perturbations (γ_5). In doubt, we trust the communication function (γ_6).

This model deadlocks in a similar scenario as that of Example 4 when both movement and communication are raised simultaneously then turn off. We can control the minimal length of the deadlocking paths with the κ parameter in γ_2. For any $\kappa \geq 1$, the shortest deadlocking path has $\kappa + 2$ time steps. We use this feature to produce experimental benchmarks for our algorithm.

There are two minimal meta-diagnoses for the shortest deadlock: $\{\gamma_1\}$ and $\{\gamma_2\}$. If we replace γ_1 with $\gamma_1' = \langle \top : \overline{\mathit{low}_{pow}} \prec \mathit{low}_{pow} \rangle$ or γ_2 by $\gamma_2' = \langle \top : t_{pow} \prec \overline{t_{pow}} \rangle$, the deadlock is eliminated.

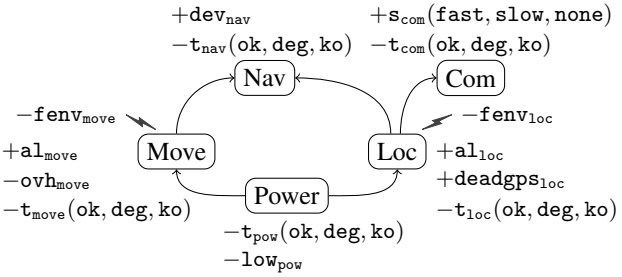


Figure 3: Illustration of the complete functional architecture.

5.2 Complete architecture

The complete architecture is illustrated on Figure 3. We consider two additional functions: localisation and navigation. The trust in each functions now values over the set $\{\mathtt{ok}, \mathtt{deg}, \mathtt{ko}\}$, representing trust in nominal, degraded and no trust in each function. Failure causes are still modelled, such as low voltage, movement environmental pertubations, but also engine overheating ($\mathtt{ovh}_{\mathtt{move}}$). Trajectory deviations are detected in the navigation component ($\mathtt{dev}_{\mathtt{nav}}$). The communication can be either fast, slow or lost (variable $\mathtt{s}_{\mathtt{com}}$ values over $\{\mathtt{fast}, \mathtt{slow}, \mathtt{none}\}$). Finally, the localisation function undergoes environmental perturbations ($\mathtt{fenv}_{\mathtt{loc}}$), and can raise alarm ($\mathtt{al}_{\mathtt{loc}}$) if the gps signal is poor, or alarm ($\mathtt{deadgps}_{\mathtt{loc}}$) if the gps is out of service.

The complete architecture model is composed of 27 variables amongst which 20 are to be estimated, 48 formulas in Δ[3], and 20 preferences. The full detail of this model is not shown for the sake of brevity.

As in the small architecture, we have several sets of preferences that let us control the presence of deadlocking paths and their minimal length. The deadlocks can be eliminated by the same minimal meta-diagnoses $\{\gamma_1\}$ and $\{\gamma_2\}$.

6 Results

We use Sat4j (Le Berre and Parrain 2010) as a SAT solver for checking the satisfiability of formulas, exhibiting a model of formulas, and its MAX-SAT version for checking whether a sequence of observations is a deadlocking path. Experiments have been conducted on a four-core Xeon 2.80GHz processor with 8GiB of RAM.

In the following results, "small_i" and "large_i" respectivly denote the small and large models with minimal deadlock length i. Also, "small_noDL" and "large_noDL" refer to the model versions that do not deadlock.

6.1 Detection of deadlocking paths

Table 1 presents the results for finding the deadlocking path of Algorithm 1. On the simple version of the architecture,

[3]Excluding the variables and formulas generated by PtLTL compilation to propositional logic.

Instance	Time (s)	nPaths
small_3	0	16
small_4	0	58
small_5	0	222
small_6	0	870
small_7	4	3446
small_8	20	13718

Instance	Time (s)	nPaths
small_9	171	54743
small_10	1694	218718
large_3	1	68
large_4	2	1649
large_5	108	39488
large_6	+3600	–

Table 1: Results of experiments for finding one deadlock. Columns are defined as follows: name of the instance (*Instance*); time of computation in seconds (*Time*); the number of $seqObs$ (see Algo 1) that have been tested (*nPaths*).

Instance	Time (s)
small_3	0
small_4	0
small_5	0
small_6	3
small_7	26

Instance	Time (s)
small_8	190
small_9	1498
large_3	6
large_4	16
large_5	581

Table 2: Results for computing minimal meta-diagnoses for deadlock returned by Algo. 1.

a deadlock of size less or equal to 7 is found in less than 5 seconds. However, for longer deadlocks, the computation time grows exponentially. For the complete architecture, the computation times out for a deadlock of size 6, because the number of observation sequences grows exponentially with their length. Also, to find a deadlock of size 6, we first test all observations sequences of size 5.

6.2 Relaxation of preferences

The next experiments are about enumerating the minimal meta-diagnoses. In all the tests (simple and complete architecture), we use the deadlock found with Algorithm 1[4]. The deadlock accepts two singleton minimal meta-diagnoses: the first contains the preference targetting $\texttt{low}_{\texttt{pow}}$ and the second the preference targetting $\texttt{t}_{\texttt{pow}}$.

Table 2 shows the time for computing all minimal meta-diagnoses. It takes a few seconds for the simple architecture when the deadlocking path length is 6 or less, then grows exponentially, because the number of scenario objects to test grows similarly.

In Table 3, we tested the computation time to test whether a set of preferences Ω is a meta-diagnosis. We show that preference subsets with different sizes have different average testing time. We tested singletons, subsets of half the preferences and subsets of all but one preference.

The results show that the computation is very fast for small subsets or short deadlocks, but grows exponentially with both deadlock length and subset size (in both cases the number of generated scenario objects grows exponentially). Some sets are particularly hard to check, and the hardest sets can take more than 10 times the average to check. They occur more frequently in sets of half the preferences, because

[4]For large_6, no deadlock was fould by Algo. 1 and we have manually generated a deadlock.

Instance	$\lvert\Omega\rvert = 1$	$\lvert\Omega\rvert = \lvert\Gamma\rvert/2$	$\lvert\Omega\rvert = \lvert\Gamma\rvert - 1$
small_3	0 [0, 0]	0 [0, 0]	0 [0, 0]
small_4	0 [0, 0]	0 [0, 0]	0 [0, 0]
small_5	0 [0, 0]	0 [0, 0]	0 [0, 1]
small_6	0 [0, 0]	0 [0, 2]	1 [0, 2]
small_7	0 [0, 0]	1 [0, 17]	5 [0, 14]
small_8	0 [0, 0]	8 [0, 134]	37 [0, 108]
small_9	0 [0, 0]	61 [0, 1129]	311 [0, 960]
large_3	0 [0, 0]	0 [0, 2]	0 [0, 1]
large_4	0 [0, 0]	0 [0, 2]	3 [1, 4]
large_5	0 [0, 0]	14 [0, 163]	87 [16, 119]
large_6	0 [0, 0]	48 [0, 444]	2691 [652, 4730]

Table 3: Results for computing whether a set of preferences Ω is a meta-diagnosis. Columns indicate the size of subsets of preferences Ω sets that were tested. In each cell, we tested at most 20 sets of the specified size, and selected sets at random when needed. Each cell contains the mean, minimum and maximum time required for testing a set in the format $mean[min, max]$.

when most perferences are relaxed, it becomes easier to find a diagnosis sequence.

This variability comes from the strategy we use to explore scenarios in Algorithm 2. In these experiments, we follow a depth-first strategy, by always expanding the scenario objects generated last. While this tends to use little memory, no intelligent heuristic is used to explore the search space.

We consider the performance satisfactory for two reasons. First, memory usage is not a limiting factor, and time is not a constraint during design. Second, long deadlocks are difficult to find, but also difficult to interpret and debug by the designer. When a diagnosis model becomes too large, architectural responses may help dividing it in smaller decentralized diagnosers.

7 Conclusion

In this paper, we present a novel approach for blaming a deadlock on a set of preferences at design time. It follows a consistency-based meta-diagnosis strategy based on relaxation of conditional preferences. We have defined and implemented algorithms with satisfactory performances. For larger benchmarks, several approaches can be explored to improve their associated computation time. For instance, we could parallelize the algorithms by dividing the space search beween several computation cores. We could also define an intelligent heuristic for finding relevant scenarios quicker.

During our experiments we noted that many deadlocking paths reproduce the same pattern. A perspective is to identify deadlock patterns to represent them more compactly. Another perspective is to find relaxations that eliminate several deadlocks at once.

References

Benveniste, A.; Fabre, E.; Haar, S.; and Jard, C. 2003. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE Transactions on Automatic Control* 48(5):714–727.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and D.Poole. 2004a. Preference-based constrained optimization with cp-nets. In *Computational Intelligence*. 137–157.

Boutilier, C.; Brafman, R. I.; Domshlak, C.; Hoos, H. H.; and Poole, D. 2004b. Cp-nets: A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.(JAIR)* 21:135–191.

Bouziat, V.; Pucel, X.; Roussel, S.; and Travé-Massuyès, L. 2018. Preferential discrete model-based diagnosis for intermittent and permanent faults. In *Submitted to 29th International Workshop on Principles of Diagnosis (DX'18), available at https://hal.archives-ouvertes.fr/hal-01796310*.

Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; and Tonetta, S. 2014. The nuxmv symbolic model checker. In *CAV*, 334–342.

Contant, O.; Lafortune, S.; and Teneketzis, D. 2004. Diagnosis of intermittent faults. *Discrete Event Dynamic Systems* 14(2):171–202.

De Kleer, J. 2009. Diagnosing multiple persistent and intermittent faults. In *IJCAI*, 733–738.

Grastien, A.; Anbulagan, J. R.; Rintanen, J.; Kelareva, E.; et al. 2007. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the National Conference on Artificial Intelligence*, volume 22, 305. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999.

Hamscher, W., et al. 1992. Readings in model-based diagnosis.

Havelund, K., and Rosu, G. 2002. Synthesizing monitors for safety properties. In *Proc. of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-02)*, 342–356.

Jéron, T.; Marchand, H.; Pinchinat, S.; and Cordier, M.-O. 2006. Supervision patterns in discrete event systems diagnosis. In *Discrete event systems, 2006 8th international workshop on*, 262–268. IEEE.

Kurien, J., and Nayak, P. P. 2000. Back to the future for consistency-based trajectory tracking. In *AAAI/IAAI*, 370–377.

Le Berre, D., and Parrain, A. 2010. The Sat4j library, release 2.2. *JSAT* 7(2-3):59–6.

Macedo, N.; Brunel, J.; Chemouil, D.; Cunha, A.; and Kuperberg, D. 2016. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, 373–383. New York, NY, USA: ACM.

Morgado, A., and Marques-Silva, J. 2005. Algorithms for propositional model enumeration and counting. In *Technical Report, Instituto de Engenharia de Sistemas e Computadores, Investigaçao e Desenvolvimento*.

Pencolé, Y., and Cordier, M.-O. 2005. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence* 164(1-2):121–170.

Pralet, C.; Pucel, X.; and Roussel, S. 2016. Diagnosis of intermittent faults with conditional preferences. In *Proceedings of the 27th International Workshop on Principles of Diagnosis (DX'16)*.

Pucel, X., and Roussel, S. 2017. Intermittent fault diagnosis as discrete signal estimation: Trackability analysis. In *28th International Workshop on Principles of Diagnosis (DX'17)*.

Qiu, W., and Kumar, R. 2006. Decentralized failure diagnosis of discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 36(2):384–395.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial intelligence* 32(1):57–95.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1995. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control* 40(9):1555–1575.

Thorsley, D., and Teneketzis, D. 2005. Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control* 50(4):476–492.

Wilson, N. 2011. Computational techniques for a simple theory of conditional preferences. *Artificial Intelligence* 175(7-8):1053–1091.

Annexe C

Single State Trackability of Dicrete Event Systems

Single State Trackability of Discrete Event Systems

Valentin Bouziat[1] and **Xavier Pucel**[1] and **Stéphanie Roussel**[1] and **Louise Travé-Massuyès**[2]
[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France
firstname.lastname@onera.fr
[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
louise@laas.fr

Abstract

Specific requirements must guide the design of autonomous systems as they are increasingly present in our everyday environment. Their properties must be carefully defined and checked to guarantee safety, security and dependability. In this paper, we adopt a discrete event system modelling framework and focus on properties that are related to diagnosis. A new property called *single state trackability* is introduced. While available observations may lead to an ambiguous estimate, i.e. several admissible state candidates, this property assesses the possibility of reducing the estimate to a single state without this leading to a dead-end in the continuation of the execution. A single state estimate advantageously facilitates decision making and allows the use of a deterministic planner in the autonomous architecture. We provide a necessary and sufficient condition for single state trackability of a discrete event system and we propose a recursive algorithm to check this property. The algorithm is validated with a set of benchmarks.

1 Introduction

Self-awareness is one of the essential properties of autonomous systems : it heavily impacts decision making and can be critical for the survival of the system. The ability of a system to react to unexpected events depends on its capacity to evaluate its current state. We focus on systems whose dynamics can be represented with a discrete event modelling formalism.

In this context, state tracking and diagnosis have been the target of many works [1; 2; 3; 4]. In general, observations are not enough to guaranty state observability [5; 6], which means that state estimation is ambiguous and returns several estimates at each time step. Consequently, the number of possible state candidates grows exponentially as time goes by. Most works tackle this problem by selecting a limited number of best candidates according to some preference criterion, for example probabilities like in [7; 8]. Yet, when the real state is not among the selected subset, this may lead to a dead-end in the continuation of the tracking. The solution proposed by [8] to this problem is to backtrack and recover the state trajectory that allows estimation to resume.

In this paper, we reduce to an extreme the number of estimates and we propose to keep only one, as in [9]. We call such an estimator a *single state estimator*. The reasons for this are several. First, autonomous architectures have a very limited amount of memory and it is not desirable to store the complete history of the system execution as it grows over time. Second, estimation must be incremental, only based on the previous estimate and the current observation at each time step. Last but not least, a single-state estimate advantageously facilitates decision making and allows the use of deterministic planners, task allocators, etc. in the autonomous architecture. This last point is also a step towards explainability of autonomous systems as autonomous decisions are based on only one estimate.

In [9], we analyse whether a single state estimator specified by a set of preferences is subject to dead-ends. When this is the case, modifying the preferences, i.e. the estimation strategy, may solve the dead-end issue [10]. However, there exists systems for which every possible single-state estimator are subject to dead-ends. This means that there is no way to estimate the state of such systems in real time without potentially facing a dead-end in the continuation of the execution. When backtracking is not an option for the considered system (for instance because of real time constraints), the occurrence of a dead-end might cause the loss of the system. In this paper, we aim at characterizing such systems and we define a new property called *single state trackability* that assesses the existence of a single-state estimator that does not lead to dead-end. Our main contribution is to provide a necessary and sufficient condition for single state trackability of a discrete event system and we propose a recursive algorithm to check this property at design time.

This paper is structured as follows. Related work is covered in Section 2. Section 3 introduces our modeling formalism for discrete event systems and incremental estimators. Then, the property of *single state trackability* is introduced in Section 4. Investigations are conducted in order to check this property in Section 5. An algorithm for checking this property is proposed in Section 6 and validated on experimental systems in Section 7.

2 Related work

While the problem of single state trackability is new, the related notions of DES observability and diagnosability have been the subject of several papers. [5] addresses observability under partial state and total event observation. The notions of (weak) indistinguishable states account for all future observations, and (weak) observability is achieved when all

pairs of states are (weakly) indistinguishable. The notion of coobservability is defined similarly with past observations. Strong coobservability implies that the current state can be uniquely determined from the observation history, which is much stronger than single state trackability.

In [6] only some events are observed, and a system is said to be observable when an observer that tracks all the possible states regularly visits states with only one candidate, i.e. with a bounded period. The notion of delayed observability is similar, but allows for the uniquely known state to be in the past. The definition of resilient observers captures the principle of robustness to perturbated observations. Observability requires the knowledge of the exact system state, which is not necessary for single state trackability. Conversely, single state trackability requires that among two undistinguishable states, one explains all the observations produced by the other, which is not necessary in observability definitions.

Diagnosability, as defined in [11], differs from our approach by several aspects. First, it targets permanent faults, while our approach can be applied to estimate the presence of intermittent faults as well. Second, it requires the complete construction of a diagnoser that poses a scalability problem, that is mitigated in [12]. Finally, it accounts for a bounded delay between the occurrence of a fault and its diagnosis. While this idea is interesting and makes for a realistic requirement, there is no universal way to extend it to intermittent phenomena. Examples include [13] and [14], which we found hardly relevant from our point of view about autonomous systems.

3 State estimation of Discrete Event Systems

In this paper we adopt a modeling approach similar to model-checking, where states are defined as assignments on a set of Boolean variables $\mathtt{V_S}$. Some variables are observed and form a set of Boolean variables $\mathtt{V_0}$ such that $\mathtt{V_0} \subseteq \mathtt{V_S}$. Variables that are not observed are estimated. We assume that the system follows discrete dynamics where each time step lasts the same duration. The set of possible system states is noted S, and each state $s \in S$ is represented by a Boolean assignment to all variables in $\mathtt{V_S}$. Consequently, $S \subseteq 2^{\mathtt{V_S}}$.

Definition 1 (Discrete event system)**.** *A discrete event system (DES) is represented by a tuple (S, Δ, s_0) where S is set of system states, $\Delta \subseteq S \times S$ is the transition relation and $s_0 \in S$ is the initial state.*

Definition 2 (Execution language)**.** *For a discrete event system (S, Δ, s_0), the* execution language $\mathcal{L}(\Delta) \subseteq S^*$ *is the set of state sequences starting with s_0 that satisfy Δ:*

$$\mathcal{L}(\Delta) = \{(s_0, s_1, \cdots, s_n) | \\ n \in \mathbb{N}^+, i \in [0, n-1], (s_i, s_{i+1}) \in \Delta\}$$

Notations : Let $seq \in \mathcal{L}(\Delta)$ be a state sequence. $seq[i]$ refers to the i^{th} state of the sequence beginning at initial state $seq[0] = s_0$. The sequence's size is denoted $|seq|$.

An assignment on $\mathtt{V_0}$ is called an *observation* and O denotes the set of all possible observations. We denote obs the function from S to O that returns the observation associated to a state and we naturally extend it from S^* to O^* as follows: $|obs(seq)| = |seq|$ and $obs(seq)[i] = obs(seq[i])$ for $i \in [0, |seq|[$.

At each time step the system sends an observation to the estimator. Formally, the estimator receives as input a sequence of observations, and produces a sequence of estimations. At each time step, the estimator selects a unique estimated state among the set of estimation candidates.

Definition 3 (Observation language)**.** *The observation language $\mathcal{L}_{obs}(\Delta) \subseteq O^*$ of a DES is the set of all consistent observation sequences.*

$$\mathcal{L}_{obs}(\Delta) = \{obs(seq) | seq \in \mathcal{L}(\Delta)\}$$

In our approach, an estimator only takes into account the previous estimated state and the current observation.

Definition 4 (Set of estimation candidates)**.** *Given a DES (S, Δ, s_0), a state $s \in S$ and an observation $o \in O$, we define the set of estimation candidates $cands(s, o)$ as the set of states in which the system could be, assuming it was in state s at the previous time step, and produces observation o at the current time step. Formally:*

$$cands(s, o) = \{\hat{s} \in S | (s, \hat{s}) \in \Delta \text{ and } obs(\hat{s}) = o\}$$

Definition 5 (Estimation function)**.** *Let (S, Δ, s_0) be a DES. An* estimation function *is a function $estim : (S \times O) \to S$ that selects a unique estimation candidate, i.e. for every $s \in S$, and every $o \in O$, if $cands(s, o) \neq \emptyset$ then $estim(s, o) \in cands(s, o)$.*

An estimator is completely defined by its estimation function. It receives observations as inputs, and the estimated state is reused at the next time step to compute the next estimation. We assume that the initial system state is known to the estimator.

Definition 6 (Estimation sequence)**.** *Let (S, Δ, s_0) be a DES, $estim : (S \times O) \to S$ an* estimation function*, and $sobs \in \mathcal{L}_{obs}(\Delta)$ an observation sequence. The* estimation sequence *for $sobs$ is the unique state sequence $sest$ such that:*

- *$sest[0] = s_0$ and*
- *for $i \in [1, |sobs|[$, if $cands(sest[i-1], sobs[i]) \neq \emptyset$ then $sest[i] = estim(sest[i-1], sobs[i])$ else $sest$ is undefined.*

4 Single state trackability

The simple fact that an estimator selects a single state estimate creates scenarios where the estimate can differ from the real system state, and later the system produces an observation that is inconsistent with the previously estimated state. In such scenarios, the set of estimation candidates is empty, the estimation function is then undefined and the estimator is unable to produce an estimation[1]. Formally, if we note s the system state and $\hat{s}$ the state estimate, if $s \neq \hat{s}$, the system may evolve in a state s' and produce an observation $obs(s') = o$ such that $cands(\hat{s}, o) = \emptyset$. We call such a situation a *dead-end*, and the observable path is called a *dead-end path*.

Definition 7 (Dead-end path)**.** *Let (S, Δ, s_0) be a DES, $estim : (S \times O) \to S$ an estimation function, $sobs \in \mathcal{L}_{obs}(\Delta)$ an observable sequence of length k, and $o \in O$ a continuation of $sobs$, i.e. $sobs.o \in \mathcal{L}_{obs}(\Delta)$; $sobs.o$ is a* dead-end path *if and only if:*

[1]Note that such scenarios happen not only in our single state approach, but for any approach that does not keep all the estimation candidates in memory.

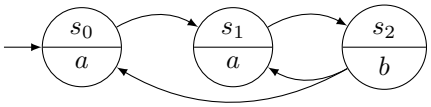


Figure 1: A non SST DES with states $S = \{s_0, s_1, s_2\}$, and observations $O = \{a, b\}$. Δ and obs are shown as in a Moore machine.

- *the estimation sequence for sobs is defined:* $sest = (s_0, \hat{s_1}, \cdots, \hat{s_k})$*;*
- *there exists no estimation candidate after observation o, i.e* $cands(\hat{s_k}, o) = \emptyset$*.*

Dead-end paths illustrate the situation where the estimator assumes something about the system's real state, and discovers later that this assumption was false. This can be a problem if some important decision was made as a consequence of this assumption. Most of the time, it is impossible (and not necessary) to know the full system state to operate it. This is why we introduce the concept of *single state trackability*, i.e. the ability to estimate the system state and never encounter a dead-end path.

Definition 8 (Single state trackability)**.** *A DES is* Single state trackable[2] *(SST) if and only if there exists an estimation function* $estim : (S \times O) \to S$ *such that no observable sequence* $sobs \in \mathcal{L}_{obs}(\Delta)$ *is a dead-end path.*

Example 1. *Let us consider the DES represented in Figure 1, described as a Moore automaton [15] with no input alphabet.*

Let the system produce the observation sequence (a, a, b)*. For the first 3 steps, estimation is trivial as the system can only go through the state sequence* (s_0, s_1, s_2)*. But for the observation sequence* (a, a, b, a)*, the set of candidates is then* $cands(s_2, a) = \{s_0, s_1\}$*, and a choice needs to be made. Let us consider the two possible estimation functions* $estim_0(s_2, a) = s_0$ *and* $estim_1(s_2, a) = s_1$*, and the two observation sequences* (a, a, b, a, a) *and* (a, a, b, a, b) *respectively produced by state sequences* $(s_0, s_1, s_2, s_0, s_1)$ *and* $(s_0, s_1, s_2, s_1, s_2)$*.*

With $estim_0$*, the observation sequence* (a, a, b, a) *is estimated as* (s_0, s_1, s_2, s_0)*, however since* $cands(s_0, b) = \emptyset$*, the observation sequence* (a, a, b, a, b) *is a dead-end.*

With $estim_1$*, the observation sequence* (a, a, b, a) *is estimated as* (s_0, s_1, s_2, s_1)*, and since* $cands(s_1, a) = \emptyset$*, the observation sequence* (a, a, b, a, a) *is a dead-end.*

Since all the possible estimation functions encounter dead-ends, the system is not SST.

5 Checking single state trackability

In this section we introduce some necessary conditions and one equivalent condition for checking single state trackability.

Definition 9 (Reachable states)**.** *A state* $\hat{s}$ *is reachable via the observation sequence* $sobs \in \mathcal{L}_{obs}(\Delta)$ *if and only if there exists a state sequence* $seq \in \mathcal{L}(\Delta)$ *such that* $obs(seq) = sobs$*, and seq ends with* $\hat{s}$*. The set of states reachable via sobs is noted* $reach(sobs)$*.*

Note that while the set of observation sequences is infinite, the set of all possible $reach(sobs)$, $sobs \in \mathcal{L}_{obs}(\Delta)$ is finite and a subset of 2^S. This set can be enumerated by constructing the so-called powerset automaton with respect to obs. This property is used in Algorithm 2.

[2]In this paper, "trackable" and "trackability" always refer to single state trackable and single state trackability respectively.

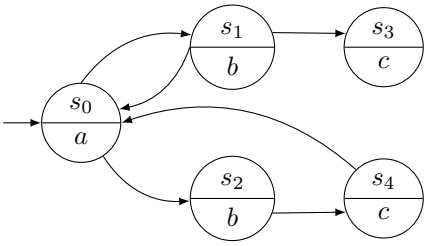


Figure 2: A DES in which all transitions are non-blocking, but that is still not SST.

Definition 10 (Non-blocking states (NBS))**.** *Let* (S, Δ, s_0) *be a DES, and* $sobs \in \mathcal{L}_{obs}(\Delta)$ *an observation sequence. A state* $\hat{s}$ *is* non-blocking *for sobs if and only if it is reachable via sobs and for every subsequent observation o, there is an estimation candidate from* $\hat{s}$*. Formally,* $\hat{s} \in reach(sobs)$ *is* non-blocking *if and only if:*

$$\forall o \in O, \text{ if } sobs.o \in \mathcal{L}_{obs}(\Delta) \text{ then } cands(\hat{s}, o) \neq \emptyset$$

A state reachable via $sobs$ but which is not non-blocking for $sobs$ is called a *blocking state*. Note that a state $\hat{s}$ may be non-blocking for some observation sequence $sobs_1$ and blocking for another sequence $sobs_2$. Non-blocking states are important because estimators must always select them, or they will encounter a dead-end path, as stated in the following propositions.

Proposition 1 (Non blocking state condition)**.** *If there exists an observation sequence* $sobs \in \mathcal{L}_{obs}(\Delta)$ *such that* $reach(sobs)$ *contains only blocking states, then the system is not trackable.*

Intuitively, Proposition 1 means that for any estimation function, the estimated sequence for $sobs$ ends with a state in $reach(sobs)$. If it contains only blocking states, then whatever state $\hat{s} \in reach(sobs)$ is chosen, there exists a continuation $sobs.o \in \mathcal{L}_{obs}(\Delta)$ that is a dead-end path.

Proposition 2 (Non blocking transition condition)**.** *Let* $sobs \in \mathcal{L}_{obs}(\Delta)$ *be an observation sequence and* $o \in O$ *an observation such that* $sobs.o \in \mathcal{L}_{obs}(\Delta)$*. If the system is single state trackable, then there exists a pair of states* $(s_1, s_2) \in \Delta$ *such that* s_1 *is non-blocking for* $sobs$ *and* s_2 *is non-blocking for* $sobs.o$*.*

Proposition 2 extends Proposition 1 to transitions, and could be extended further to paths of arbitrary length. However, when constructing the set of all $reach(sobs)$ to check Proposition 1, and particularly when constructing a transition between $reach(sobs)$ and $reach(sobs.o)$, it is straightforward to verify Proposition 2 on-the-fly. Checking it for longer paths can be done efficiently in a dynamic programming style algorithm.

Extending the NBS property to paths of any length does not provide a sufficient condition for trackability. In the DES presented in Figure 2, it is always possible for a given observation sequence to associate a state sequence respecting the condition of proposition 2. If we take the observation sequence (a, b, a, b, c) produced by the non-blocking state sequence $(s_0, s_1, s_0, s_2, s_4)$, however, it is impossible to construct a function $estim : (S \times O) \to S$ allowing

to borrow this sequence since starting from s_0 and receiving observation b, we would have to choose sometimes s_1, sometimes s_2. To provide a necessary and sufficient condition for trackability, we check that not only observable sequences, but the full observable language is supported by some estimator.

Definition 11 (Estimator accepted language)**.** *Let (S, Δ, s_0) be a DES, $sobs \in \mathcal{L}_{obs}(\Delta)$ an observation sequence and $estim : (S \times O) \to S$ an estimation function. The language accepted by this estimation function, denoted $\mathcal{L}_{obs}(estim)$, is the set of all possible observation sequences for which there exists an estimation sequence (i.e. that are not dead-ends). Formally :*

$$\begin{aligned}\mathcal{L}_{obs}(estim) = \{sobs \in \mathcal{L}_{obs}(\Delta) \mid (|sobs| > 1) \\ \text{and } \exists sest \in \mathcal{L}(\Delta) \\ \text{s.t. for } i \in [1, |sobs|], obs(sest[i]) = sobs[i] \\ \text{and } estim(sest[i-1], sobs[i]) = sest[i]\}\end{aligned}$$

It now becomes apparent that finding dead-ends for a given estimator can be done with an algorithm similar to the algorithm for checking equality of regular languages.

Proposition 3 (Trackability condition)**.** *A DES (S, Δ, s_0) is* single state trackable *if and only if there exists an estimation function whose accepted language equals the observation language of the system:*

$$\exists\, estim : (S \times O) \to S, \mathcal{L}_{obs}(estim) = \mathcal{L}_{obs}(\Delta)$$

The proof is straightforward: if proposition 3 is satisfied, then the estimation function provides an estimation sequence for all elements of $\mathcal{L}_{obs}(\Delta)$. Thus there are no dead-ends. The main difficulty lies in finding such an estimation function, or proving that it does not exist.

To efficiently check trackability, our approach consists in evaluating partially defined estimation functions. We define dead-ends for such partial estimation functions, and introduce a proposition that will be used in our algorithm.

Definition 12 (Extension of partial estimation functions)**.** *Let $estim : (S \times O) \to S$ be an estimation function, and let $pestim$ be a partial function from $(S \times O)$ to S. $estim$* extends *$pestim$ if and only if for every couple (s, o) from $S \times O$ such that $pestim(s, o)$ is defined, then $pestim(s, o)$ = $estim(s, o)$.*

A partial function that can be extended in an estimation function is called a partial estimation function*.*

Definition 13 (Dead-end for partial functions)**.** *Let $pestim$ be a partial estimation function from $(S \times O)$ to S, $sobs$ be an observation sequence and $sobs.o \in \mathcal{L}_{obs}(\Delta)$ a continuation. $sobs.o$ is a dead-end path for $pestim$ if and only if there exists a state sequence $sest = (s_0, \ldots, \hat{s_k})$ such that $pestim(sest[i-1], sobs[i])$ is defined and equals $sest[i]$ for $i \in [1, |sobs|]$, and $cands(\hat{s_k}, o) = \emptyset$.*

Proposition 4. *If $sobs \in \mathcal{L}_{obs}(\Delta)$ is a dead-end for a partial estimation function $pestim$, then $sobs$ is a dead-end for every estimation function $estim : (S \times O) \to S$ that extends $pestim$.*

6 Algorithm

We describe an algorithm for checking the single state trackability of a system, based on a search for dead-ends for partially defined estimation functions. Our algorithm is organized in two components: the first one produces partially defined estimation functions and the second one checks if a given partial estimation function satisfies Proposition 3. The algorithm is illustrated in Figure 3.

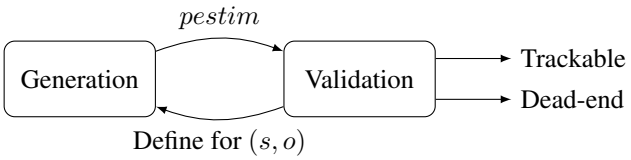


Figure 3: Algorithm structure.

The generation component recursively produces partial estimation functions $pestim$ and sends them for validation. The validation component has 3 outcomes: "Trackable" means the system is trackable with the current $pestim$; "Dead-end" means the current $pestim$ has a dead-end; "Define for (s, o)" means $pestim$ for the input pair (s, o). According to the validation result, the generation component may return or recursively produce other partial estimation functions.

6.1 Estimation function generation

The generation component provides the *GenEstim* function described in Algorithm 1 that starts the algorithm and calls the validation function. At the first iteration, the generation component produces the empty partial estimation function (i.e. defined on $\emptyset$), and sends it to the validation component.

If the validation component returns "Trackable", it means that the current partial estimation function $pestim$ defines an estimator that accepts $\mathcal{L}_{obs}(\Delta)$, and that the pairs for which $pestim$ is undefined are not used for estimation. Thus, every extension of $pestim$ satisfies Proposition 3, and the system is single state trackable. We can stop the algorithm and return true.

If the validation component returns "Dead-end", it means that there exists a dead-end path for $pestim$. By Proposition 4, no extension of this partial function can satisfy Proposition 3. We just stop the recursion.

If the validation component returns "Define for (s, o)", it means that there exists a pair (s, o) such that s is reachable, that $cands(s, o)$ contains several candidates, and that $pestim$ is undefined for (s, o). In this case, we need to check if there exists an estimation option for (s, o) that satisfies Proposition 3, so we recursively generate them.

Since the algorithm recursively explores all the estimation options, if at the end it has found no partial estimation function $pestim$ that satisfies Proposition 3, then we are certain that the system is not single state trackable.

6.2 Estimation function validation

The validation component contains a function CheckEstim that checks whether the language accepted by a given partial estimation function is equal to the observation language of the system. The algorithm is based on a slight modification of the classical algorithm for testing regular language equality [16] in order to account for cases where there are several estimation candidates, and the partial estimation function is undefined.

The approach is to simulate the execution of the estimator and the system, while ensuring that they are synchronized on the same observation sequences. Since for every observation sequence $sobs$, there exists (at most) one unique estimation sequence, we only need to keep track of a single

Algorithm 1 Generation component: GENESTIM function

```
1: Input
2:    Σ = (S, Δ, s_0): a DES
3: Output
4:    boolean: Σ is single state trackable
5: function GENESTIM(Σ, pestim)
6:    switch CHECKESTIM(Σ, pestim, s_0, {s_0}) :
7:       case Trackable : return true
8:       case Dead-end : return false
9:       case Define for (s,o) :
10:         for c in cands(s, o) do
11:            ext ← pestim ∪ ((s, o), c)
12:            if GENESTIM(Σ, ext) then return true
13:         end for
14:         return false
```

Algorithm 2 Validation component: CHECKESTIM function

```
1: Input
2:    Σ = (S, Δ, s_0) : a DES
3:    estim: a partial estimation function
4:    estSt ∈ S : the estimator state
5:    sysSts ⊆ S : the possible system states
6: Output
7:    "Trackable", "Dead-End" or "Define for (s, o)"
8: Global
9:    visited ∈ S × 2^S ← ∅
10: function CHECKESTIM(Σ, estim, estSt, sysSts)
11:    if (estSt, sysSts) ∈ visited then
12:       return Trackable
13:    visited ← visited ∪ (estSt, sysSts)
14:    nextObs ← {obs(s')|∃s ∈ sysSts, (s, s') ∈ Δ}
15:    for o in nextObs do
16:       estNxts ← NEXTESTSTATES(estSt, o)
17:       sysNxts ← {s' | obs(s') = o ∧
18:                                  ∃s ∈ sysSts, (s, s') ∈ Δ}
19:       if estNxts = ∅ then
20:          return Dead-end
21:       else if estNxts = {estNxt} then
22:          rec ←
23:                CHECKESTIM(Σ, estim, estNxt, sysNxts)
24:          switch rec :
25:             case Dead-End : return Dead-End
26:             case Define for (s, o) : return Define for
    (s, o)
27:             case Trackable : continue
28:       else
29:          return Define for (estSt, o)
30:    end for
31:    return Trackable
```

estimator state. Thus the estimator state reached via an observation sequence $sobs$ is associated with the set of system states $reach(sobs)$ (see Definition 9).

The algorithm recursively explores pairs $(estSt, sysSts)$ where $sysSts = reach(sobs)$ for some $sobs \in \mathcal{L}_{obs}(\Delta)$, and where $estSt \in sysSts$. To ensure termination, a global variable "visited" stores the pairs that have already been explored. The algorithm looks for sequences $sobs$ for which the estimator is not defined, so the termination condition is met only if no such sequence exists. In this case, the languages are equal, and we return "Trackable" (lines 11 to 13).

At each iteration, CheckEstim calculates the observations that can be produced by the system (line 14). Then for every such observation, it calls the `NextEstStates`$(estSt, o)$ (line 16) function defined as follows. If $pestim$ is defined for $(estSt, o)$, it returns $\{pestim(estSt, o)\}$ else it returns $cands(estSt, o)$. The algorithm then tests the number of possible estimator states: If $estNxts = \emptyset$ (line 19): this means that, if $sobs$ is the observation sequence that led to this recursive call, $sobs.o$ is a dead-end path (see Definition 7), we (recursively) return "Dead-end".

If $estNxts = \{estNxt\}$ (line 21): this means that for the current pair $(estSt, o)$, either there is a unique successor or that $pestim$ is defined. In this case we continue the search with a recursive call.

If $|estNxts| > 1<$ (line 28): there are several estimation candidates, and $pestim$ is undefined for $(estSt, o)$. We (recursively) return "Define for $(estSt, o)$" so that the generation component will generate estimation functions defined for this pair.

6.3 Performance

The performance of the algorithm described above can be significantly enhanced with a few mechanisms. A preliminary check is added to verify propositions 1 and 2 for every set of states that can be reached via some observable sequence.

In our experiments, the main source of complexity is the number of partial estimation functions to be tested. The only mechanism we have to prune partial estimation functions is to find dead-end paths as early as possible.

First, in Algorithm 1, upon detection of a Dead-End, one can try to remove from $pestim$ estimation triplets that are not used in the dead-end path, and memorize this trimmed partial estimation function. This way, by Proposition 4, we can test at line 10 whether some extensions are future functions that extend the one we just memorized, and spare some calls to Algorithm 2.

Second, there exist partial estimation functions for which there are both dead-end paths and undefined pairs. In these cases, in Algorithm 2, we may return either "Dead-end" or "Define for (s, o)" according to the order of traversal at line 15. Instead of immediately returning "Define for (s, o)", we store it in memory, and pursue the search for a dead-end. When we encounter a dead-end we immediately return it. When the recursive search finishes without finding any dead-end, if we have encountered an undefined pair (s, o) we return "Define for (s, o)" otherwise we return "Trackable". This favors early dead-end detection.

The properties of blocking states of Propositions 1 and 2 can also be used to reduce the search space: in Algorithm 1 at line 10, one can skip blocking states as they will definitely lead to a dead-end.

7 Experiments

We tested our approach on an example inspired from the autonomous robotics framework PLEXIL [17]. This framework is organised around the concept of actions, that have a complex hierarchical workflow. We use a simplified sequential action workflow described in Example 2.

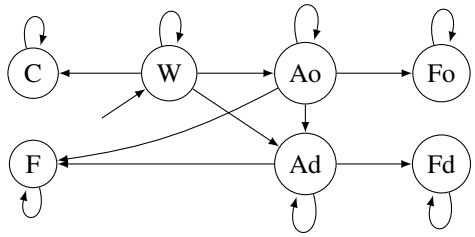


Figure 4: The workflow of an action, that can be (W)aiting, (C)ancelled, (A)ctive (o)k, (F)inished (o)k, (F)ailed, (A)ctive (d)elayed, or (F)inished (d)elayed.

Example 2. *We consider a robotics framework where robot plans consist in a sequence of actions. The workflow of an action is illustrated in Figure 4. We consider a robot with a plan composed of two sequential actions: move and inspect, whose states are represented by variables mv and ins. The robot's health status is described by three Boolean variables $hnav$, $hsens$ and $hpow$ representing respectively whether the navigation, sensor and power supply functions perform normally at each time step. Another Boolean variable $pert$ indicates whether the robot is subject to perturbations (slippery terrain, obstacle, wind) at each time step. We note $move = W$ to express that the move action is in state W, and denote $Y(v)$ the value at the previous time step for variable $v \in \{mv, ins, hnav, hsens, hpow, pert\}$. We use the function $start(v) = K$ that means $Y(v) \neq K \wedge v = K$. The system behaviour is described by the automata for each action, plus the following constraints:*

$$mv \in \{W, Ao, Ad\} \rightarrow ins = W \quad (1)$$
$$mv \in \{C, F\} \rightarrow ins = C \quad (2)$$
$$start(mv = Fo) \rightarrow ins = Ao \quad (3)$$
$$start(mv = Fd) \rightarrow ins = Ad \quad (4)$$
$$start(mv = Ad) \rightarrow (\neg hnav \vee \neg hpow \vee pert) \quad (5)$$
$$start(mv = F) \rightarrow \neg hpow \quad (6)$$
$$start(ins = Ad) \rightarrow (\neg hsens \vee \neg hpow \vee pert) \quad (7)$$
$$start(ins = F) \rightarrow \neg hpow \quad (8)$$
$$hnav \vee hsens \rightarrow hpow \quad (9)$$

Action ins must remain in W while action mv executes (1)*. If mv fails or is cancelled, ins is cancelled* (2)*. ins starts at the moment when mv finishes* (3), (4)*. A delay in mv (resp. ins) can be explained by a navigation (resp. sensor) or power supply problem, or a perturbation* (5) *(resp.* (7)*). A failure in mv or ins can only be explained by a problem in the power supply* (6)*,* (8)*. A problem in the power supply propagates to the sensor and navigation* (9)*.*

Variables mv and ins are observable, i.e. the state of each action is known. Variables $hnav$, $hsens$, $hpow$ and $pert$ are estimated. The set of states S is the set of valuations for all variables, the obs function restricts a valuation to variables mv and ins. For example, the initial state $(mv = W, ins = W, hnav, hsens, hpow, \overline{pert})$ yields the observation $(mv = W, ins = W)$.

Our algorithm is implemented in Scala, and executed on an Intel® Xeon(R) W-2123, 3.60GHz 8 core processor, with memory limited to 4 gigabytes. The system as described in Example 2 is trackable, so we introduced some modifications to make it non-trackable. We made actions show the same observation in their "Ao" and "Ad" states.

Model	States	Succ.	Result	Time (s)
Example 2	112	13.7	yes	66
Example 2-modified	112	13.7	no	0.4
Valve controller	209	15.5	no	0.4
Valve driver	51	22.3	yes	56

Table 1: Computation times for checking trackability. Column "States" indicates the number of states in the system, "Succ." the average number of outgoing transitions for each state, "Result" whether the system is trackable, and "Time" the computation time in seconds.

We also modelled the example systems from [11] (Valve controller) and [7] (Valve driver).

Results are presented in Table 1 and show that non-trackable systems are detected very quickly. This is due to our preliminary check described in section 6.3 that catches it early. While propositions 1 and 2 are not sufficient to ensure trackability, they catch many cases. For trackable systems, the computation time is related to the number of partial estimation functions tested. Note that it can be made faster by taking into account operational requirements to limit the space of estimation functions to explore.

8 Conclusion

This paper motivates and defines single state trackability for partially observed discrete event systems. This property states that it is possible to track the execution of a system by keeping a unique state in memory, without ever losing consistency with its dynamics. Some related conditions are provided along with an algorithm for checking this property. Experimental results exemplify this algorithm applied to autonomous robots. The algorithm is a proof of concept of the approach but could be improved in many ways, for instance by defining specific heuristics to make it more efficient. Larger benchmarks should also be tested.

Through this paper, single state trackability is achieved by finding one estimation function whose observation language matches that of the system. In general, as there might be many such functions, we could look for the best estimation function in regard to other properties, for example estimation correctness for some variables or at some instants.

Work about observability and diagnosability often account for a possible bounded delay between events and their observation or diagnosis. If we could account for delay in the estimation process, we could address a more general single state trackability definition. The theoretical study of the complexity of the single state trackability existence is also part of our prospects.

In addition, we are convinced that our work is strongly linked with controller synthesis [18], in particular when framed in the game theory framework [19; 20]. These links suggest future work for precise comparison and possible enhancement with ideas from these different areas.

Finally, automatic or semi-automatic synthesis of estimation functions is a direct application of this work, for example by representing estimation function with compact languages such as in [9].

References

[1] WM Wonham, Kai Cai, and Karen Rudie. Supervisory control of discrete-event systems: A brief history. *Annual Reviews in Control*, 2018.

[2] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[3] Alban Grastien, Marie-Odile Cordier, and Christine Largouët. Incremental diagnosis of discrete-event systems. In *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'05*, Pacific Grove, CA, USA, 2005.

[4] Alban Grastien, J Rintanen Anbulagan, Jussi Rintanen, Elena Kelareva, et al. Diagnosis of discrete-event systems using satisfiability algorithms. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, volume 22, page 305, Vancouver, British Columbia, 2007.

[5] Peter J Ramadge. Observability of discrete event systems. In *Proceedings of the 25th IEEE Conference on Decision and Control CDC'86*, pages 1108–1112, Athens, Greece, 1986. IEEE.

[6] C. M. Ozveren and A. S. Willsky. Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7):797–806, July 1990.

[7] C. Brian Williams and P Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, Portland, Oregon, 02 1996.

[8] James Kurien and P Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence*, pages 370–377, Austin, Texas, USA, 2000.

[9] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Preferential discrete model-based diagnosis for intermittent and permanent faults. In *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'18*, Warsow, Poland, August 2018. CEUR Workshops Proceedings.

[10] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Preference-based fault estimation in autonomous robots: Incompleteness and meta-diagnosis - extended abstract. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*, pages 1841–1843, Montreal, Canada, May 2019.

[11] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995.

[12] Anika Schumann, Yannick Pencolé, et al. Scalable diagnosability checking of event-driven systems. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI'00*, volume 7, pages 575–580, Hyderabad, India, 2007.

[13] Olivier Contant, Stéphane Lafortune, and Demosthenis Teneketzis. Diagnosis of intermittent faults. *Discrete Event Dynamic Systems*, 14(2):171–202, 2004.

[14] Johan De Kleer. Diagnosing multiple persistent and intermittent faults. In *Proceedings of the 21th International Joint Conference on Artificial Intelligence IJCAI'19*, Pasadena, CA, USA, June 2009.

[15] Edward F. Moore. Gedanken-experiments on sequential machines. In Claude Shannon and John McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ, 1956.

[16] Christos G Cassandras and Stephane Lafortune. *Introduction to discrete event systems*. Springer Science & Business Media, 2009.

[17] Vandi Verma, Tara Estlin, Ari Jónsson, Corina Pasareanu, Reid Simmons, and Kam Tso. Plan execution interchange language (plexil) for executable plans and command sequences. In *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space iSAIRAS'05*, Munich, Germany, 2005.

[18] Cédric Pralet, Gérard Verfaillie, Michel Lemaître, and Guillaume Infantes. Constraint-based controller synthesis in non-deterministic and partially observable domains. In *Proceedings of the 2010 Conference on ECAI 2010: 19th European Conference on Artificial Intelligence*, pages 681–686, Amsterdam, The Netherlands, 2010.

[19] Doyen Laurent and Jean-François Raskin. Games with imperfect information: Theory and algorithms. In *Lectures in Game Theory for Computer Scientists*, page 185–212, Cambridge, 2011.

[20] Dietmar Berwanger and Anup Basil Mathew. Infinite games with finite knowledge gaps. *Inf. Comput.*, 254(P2):217–237, June 2017.

Annexe D

Model-Based Synthesis of Incremental and Correct Estimators for Discrete Event Systems

Model-Based Synthesis of Incremental and Correct Estimators for Discrete Event Systems - Author version

Stéphanie Roussel[1] , **Xavier Pucel**[1] , **Valentin Bouziat**[1] and **Louise Travé-Massuyès**[2]

[1] ONERA / DTIS, Université de Toulouse, F-31055 Toulouse – France

[2] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France

{stephanie.roussel, xavier.pucel, valentin.bouziat}@onera.fr, louise@laas.fr

Abstract

State tracking, i.e. estimating the state over time, is always an important problem in autonomous dynamic systems. Run-time requirements advocate for incremental estimation and memory limitations lead us to consider an estimation strategy that retains only one state out of the set of candidate estimates at each time step. This avoids the ambiguity of a high number of candidate estimates and allows the decision system to be fed with a clear input. However, this strategy may lead to dead-ends in the continuation of the execution. In this paper, we show that single-state trackability can be expressed in terms of the simulation relation between automata. This allows us to provide a complexity bound and a way to build estimators endowed with this property and, moreover, customizable along some correctness criteria. Our implementation relies on the Sat Modulo Theory solver MONOSAT and experiments show that our encoding scales up and applies to real world scenarios.

1 Introduction

For autonomous systems, state estimation and tracking is a critical task because it strongly influences the decisions that are made and that can be essential to the life of the system. It provides the means to diagnose the faults and to react to the various hazards that can affect the system.

In this paper, we focus on discrete event systems [Zaytoon and Lafortune, 2013]. When the system state is partially observable, the number of candidate estimates can quickly become too large to be usable. In embedded or distributed systems, memory and communication limitations can also become a problem, even with symbolic representations techniques such as in [Torta and Torasso, 2007]. These limitations lead us to propose an estimation strategy that retains only one state out of the set of candidate estimates at each time step, as in [Bouziat *et al.*, 2018]. This can be seen as an extreme strategy but nevertheless efficient to feed the decision system with a clear input and consistent with several works that select a limited number of best candidates according to some preference criterion, for example probabilities like in [Williams and Nayak, 1996; Kurien and Nayak, 2000]. However, the more we limit the number of estimates, the more we may be confronted with the problem of dead-ends related to the fact that next observations may prove that previous estimates were wrong and there is no continuation in the estimated trajectory. Because backtracking [Kurien and Nayak, 2000] is not a viable solution with real time constraints, we propose to build single-state estimators that are guaranteed to avoid dead-ends. The ability of a system to support the construction of a single-state dead-end free estimator has been defined as single-state trackability in [Bouziat *et al.*, 2019].

Several properties have been investigated for discrete event systems. A system is (strongly) detectable if one can determine its state along some (all) trajectories of the system [Shu *et al.*, 2007]. Diagnosability [Lafortune *et al.*, 2018] and manifestability [Dague *et al.*, 2019] amount to strong detectability and detectability for faulty trajectories. Single-state trackability is somewhat orthogonal to these properties.

In this paper, we recall the necessary and sufficient condition for single-state trackability as it is derived in [Bouziat *et al.*, 2019] based on equality of regular languages. A first contribution is to show that this condition can be expressed in terms of the simulation relation between automata [Holík *et al.*, 2018]. This allows us to provide a complexity bound for the single-state trackability problem and a way to build estimators endowed with this property. A second contribution of the paper is to propose two customizable correctness criteria that lead the estimator to constrain the acceptable estimator states towards the real states of the system. A third contribution is an implementation that relies on the Sat Modulo Theory (SMT) solver MONOSAT for synthesising estimators. Intensive experiments show that our implementation is able to synthesize dead-end free estimators on real world scenarios.

The paper is organised as follows. Related work is discussed in Section 2. Section 3 introduces incremental single-state estimation and how it is performed in our framework. The property of single-state trackability is defined and characterized in Section 4 and the associated decision problem is proved to be in NP. Customisable correctness of single-state estimators is formalized in Section 5. Section 6 presents the SMT encoding for synthesizing single-state estimators given a system specification. Experimental results are presented in Section 7, and future work is discussed in Section 8.

2 Related Work

A related research domain is the well-known controller synthesis problem, as described in [Ramadge and Wonham, 1987]. Given a specification of the system, controller synthesis consists in finding a decision procedure that produces a command to apply to the system, given observations as input. Under partial observability, a trend of work proposes to synthesize finite-state controllers, which use a size-limited memory to record information related to the past. These were first introduced for POMDP [Meuleau *et al.*, 1999] and shown to be useful for non-deterministic planning [Bonet *et al.*, 2009; Pralet *et al.*, 2010]. Restricting the information recorded from the past is also one important feature of our approach. However, there are several differences compared to our incremental estimation approach. The above works design controllers guaranteed to satisfy some properties related to planning, in particular the evolution of the controlled system must terminate in a goal state. In our estimation framework, we are rather interested in correctness properties, i.e. guaranteeing that the estimated state is "as close as possible" to the real state of the system (see Section 5).

Controller synthesis has also been approached within the game theory framework. In particular, [Doyen and Raskin, 2011] consider observation-based strategies for two-player turn-based games. Similar to our approach, observation-based strategies rely on imperfect information on the past observation sequence, hence defining partially observable games. Such games occur in the synthesis of a controller that does not see the private state of the plant. The main difference with our approach is that game theory considers games as complete graphs. Such a hypothesis means that strategies cannot encounter dead-ends in the execution, *i.e.* there is always a feasible action from any state of the game.

3 Incremental Single-State Estimation

In this section, we present the process of state estimation for partially observable systems modelled by non-deterministic Moore machines with an empty input alphabet, and whose output alphabet represents the system observations.

Definition 1 (System)**.** *A system M is defined by a 5-tuple (S, s_0, O, Δ, obs) consisting of the following:*

- *a finite set of states S;*
- *an initial state $s_0 \in S$;*
- *a finite set of observations O (output alphabet in the Moore machine);*
- *a transition relation $\Delta \subseteq S^2$;*
- *an observation function $obs : S \to O$ mapping each state to its output.*

Without loss of generality, we consider that all the states of the system are reachable from the initial state s_0.

We define $cands$ as the function from $S \times O$ to 2^S such that for all state s in S, for all o in O, $cands(s, o)$ represents the set of successors of s that have observation o. Formally, $\forall s \in S, \forall o \in O, cands(s, o) = \{t | (s, t) \in \Delta \text{ and } obs(t) = o\}$.

We also define $nextObs$ the function $S \to 2^O$ such that $nextObs(s)$ is the set of observations that can be observed just after the system is in state s. Formally, $\forall s \in S, nextObs(s) = \{o | cands(s, o) \neq \emptyset\}$.

Notation. A state sequence seq is a list $(s_0, s_1, \ldots, s_{n-1})$ where each s_i is a state in S; $|seq| = n$ is the length of the sequence and $seq[i] = s_i$ is the ith state in the sequence; $last(seq)$ designates the last state of seq; if s is a state, $seq \cdot s$ is the sequence of length $|seq| + 1$ that begins with seq and ends with s. Similarly, we define an observation sequence. We also extend the function obs to a state sequence: if seq is a state sequence, $obs(seq)$ is the observation sequence seq_{obs} such that $|seq_{obs}| = |seq|$ and $seq_{obs}[i] = obs(seq[i])$ for all $i \in [0, n-1]$.

Definition 2 (Language, observation language)**.** *The language associated with a system $M = (S, s_0, O, \Delta, obs)$ is the set of state sequences accepted by the system and starting with s_0. Formally $\mathcal{L}(M) = \{seq \in S^+ | seq[0] = s_0$ and $\forall i \in [1, |seq| - 1], (seq[i-1], seq[i]) \in \Delta\}$.*
The observation language is the language accepted by the system projected on the observations. Formally, $\mathcal{L}_{obs}(M) = \{obs(seq) | seq \in \mathcal{L}(M)\}$.

We now focus on the estimation part for systems defined above. Since we consider non-deterministic, partially observable systems, there may be several state sequences that explain a given observation sequence. We adopt an incremental approach to select a unique explanation, called an *estimation strategy*. Then, we show that for a given system, any estimation strategy can be represented by a finite state machine called an estimator.

Definition 3 (Estimation strategy)**.** *An* incremental single-state estimation strategy *for a system $M = (S, s_0, O, \Delta, obs)$ is a function $estim : S \times O \to S$ such that for all s in S, for all o in $nextObs(s)$, $estim(s, o)$ represents the estimated state of the system at time step n if it was estimated in state s at time step $n-1$ and if o is observed at time step n.*
We impose the estimation strategy to be consistent both across time (i.e. $estim$ is a function) and with the system behaviour (i.e. $estim(s, o)$ belongs to $cands(s, o)$).

For conciseness purposes, we use "estimation strategy" to refer to "incremental single-state estimation strategy". Given a system and an estimation strategy, the estimation process takes place as follows.

Definition 4 (Estimated sequence)**.** *Let M be a system, seq_{obs} be an observation sequence in $\mathcal{L}_{obs}(M)$ and $estim$ an estimation strategy for M. The estimated sequence for seq_{obs} is the state sequence $\widehat{seq} \in \mathcal{L}(M)$ such that $\widehat{seq}[0] = s_0$ and for all i in $[1, |\widehat{seq}| - 1]$, $\widehat{seq}[i] = estim(\widehat{seq}[i-1], seq_{obs}[i])$.*

Example 1. *Let $M = (S, s_0, O, \Delta, obs)$ be the system represented in Fig. 1. We have $S = \{s_0, \ldots, s_5\}$, $O = \{a, b\}$, Δ is represented by the arrows in the figure, $obs(s_0) = obs(s_3) = obs(s_5) = a$ and $obs(s_1) = obs(s_2) = obs(s_4) = b$.*

In this system only two pairs in $S \times O$ have more than one estimation candidate: (s_0, b) and (s_1, a). Let $estim_1$ be the estimation strategy such that $estim_1(s_0, b) = s_1$ and $estim_1(s_1, a) = s_3$, and the unique candidate is selected

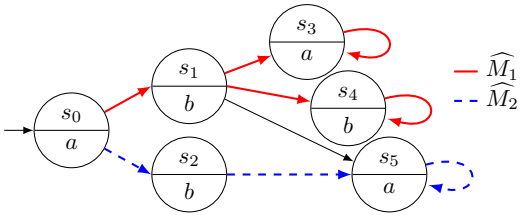


Figure 1: A simple system M, with one possible estimator $\widehat{M}_1$ depicted in solid red, and another $\widehat{M}_2$ in dashed blue.

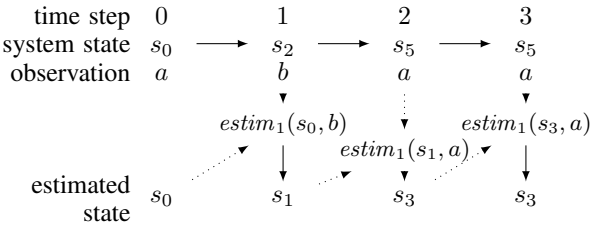


Figure 2: Estimation process in Example 1.

otherwise (for example, $cands(s_3, a) = \{s_3\}$ so necessarily $estim_1(s_3, a) = s_3$).

When the system produces the observation sequence (a, b, a, a), it can be explained by three state sequences in the system: (s_0, s_1, s_3, s_3), (s_0, s_1, s_5, s_5) and (s_0, s_2, s_5, s_5). With the strategy $estim_1$, we select the first one. The estimation process is illustrated in Figure 2 using this strategy.

In Example 1, one can observe that the state s_2 cannot be part of any estimated sequence, as the estimation strategy chooses a branch of the system states that does not contain s_2. The set of states that the estimation strategy actually uses is denoted $\widehat{S}$ and is formally defined by the set of states in S that are part of an estimation sequence associated with a sequence of observation in $\mathcal{L}_{obs}$.

An estimation strategy $estim$ for a system $M = (S, s_0, O, \Delta, obs)$ can also be represented by an *estimator*, *i.e.* a 5-tuple $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ composed of the reachable states and transitions in the system with the strategy.

Definition 5 (Estimator)**.** *Let $M = (S, s_0, O, \Delta, obs)$ be a system and $estim$ an estimation strategy. The* estimator *induced by $estim$ is a deterministic Moore machine $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ such that:*

- *$\widehat{S} \subseteq S$ is the set of states that belong to some estimated sequence for the given estimation strategy;*
- *$\widehat{\Delta}$ is the smallest subset of Δ such that $\forall \widehat{s} \in \widehat{S}, \forall o \in nextObs(\widehat{s})$, $(\widehat{s}, estim(\widehat{s}, o))$ belongs to $\widehat{\Delta}$.*

In Figure 1, $\widehat{M}_1$ is the estimator associated with the estimation strategy $estim_1$ introduced in Example 1.

Definition 5 indicates how for a given system, every estimation strategy can be represented by an estimator. Note that the reverse also holds: for a given system $M = (S, s_0, O, \Delta, obs)$, any deterministic Moore machine $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ such that $\widehat{S} \subseteq S$ and $\widehat{\Delta} \subseteq \Delta$, is in fact an estimator that implements an estimation strategy $estim$ such that $estim(\hat{s}, o)$ is the unique $\hat{t}$ such that $(\hat{s}, \hat{t}) \in \widehat{\Delta}$ and $obs(\hat{t}) = o$, for $\hat{s} \in \widehat{S}$ and $o \in O$. In the rest of the paper, we use estimators to represent estimation strategies.

4 Single-State Trackability

4.1 Dead-ends

In this section, we first recall the notion of dead-end as defined in [Bouziat *et al.*, 2019]. During estimation, the estimator may choose a state sequence different from the one taken by the system. This can be particularly problematic when the following conditions happen: a) the system is in state s, the estimator estimates that it is in state $\widehat{s}$ with $\widehat{s} \neq s$; b) the system moves in state t and produces observation o; c) there exists no candidate $\widehat{t}$ in $\widehat{S}$ such that $obs(\widehat{t}) = o$ and $(\widehat{s}, \widehat{t})$ belongs to $\widehat{\Delta}$.

Definition 6 (Dead-end)**.** *A* dead-end *for an estimator $\widehat{M}$ of a system M is an observation sequence that belongs to $\mathcal{L}_{obs}(M)$ but does not belong to $\mathcal{L}_{obs}(\widehat{M})$.*
An estimator $\widehat{M}$ is dead-end free *if there is no dead-end for it, i.e. $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$.*

Example 2. *We consider the example illustrated in Figure 1. Let $\widehat{M}_2$ be the estimator induced by the estimation strategy $estim_2$ defined by $estim_2(s_0, b) = s_2$ (no other choices necessary). The sequence of observations (a, b, b, b) belongs to $\mathcal{L}_{obs}(M)$ as it is produced by the state sequence (s_0, s_1, s_4, s_4) but it does not belong to $\mathcal{L}_{obs}(\widehat{M}_2)$. This observation sequence is therefore a dead-end for $\widehat{M}_2$. Note that $\widehat{M}_1$ is dead-end free.*

In [Kurien and Nayak, 2000], dead-ends are handled by backtracking in time in order to modify previous estimation choices. The main drawback of this technique is the lack of control over both the computational time and memory. Moreover, some actions may have been taken based on the previous estimated states that cannot be undone. Backtracking in time and revising estimations may turn past actions sub-optimal or even inconsistent.

Another way to tackle dead-ends is to keep a fixed number k of past observations and reason on this window [Su *et al.*, 2014]. However this approach is also subject to dead-ends as the estimator and system paths may diverge earlier beyond the memorized window.

Our goal is to design estimators able to follow the execution of a system while keeping just a unique state in memory and never encountering dead-ends. Such deterministic incremental estimators may be feasible depending on the system specifications. We express that possibility by defining single-state trackability for discrete event systems.

4.2 Trackability Problem

In [Bouziat *et al.*, 2019], a system is considered as single-state trackable if there exists a dead-end free single-state incremental estimator for this system. They show that this is the case if and only if there exists an estimator with an observation language equal to that of the system. In this paper, we

take the language equivalence as a definition of Single-State Trackability.

Definition 7 (Single-State Trackability)**.** *A system M is* single-state trackable *if there exists an estimator $\widehat{M}$ for M such that $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$.*

Determining whether a system is single-state trackable or not is called the TRACKABILITY problem.

The problem of deciding if two languages are equivalent (also called trace-equivalence) is, in the general case, a PSPACE-complete problem [Hüttel and Shukla, 1996]. This leads to the definition of a rather complex algorithm for the TRACKABILITY problem in [Bouziat *et al.*, 2019]. This approach can be improved because the system and the estimator, for which language equivalence is considered, are not independent from each other. In fact, the estimator is a submachine of the system's machine. A contribution of this paper is to prove that in this case, the language equivalence problem is reduced to the simulation preorder problem, which is decidable in polynomial time [Shukla *et al.*, 1996].

We first adapt the definition of simulation to the case of a system and an estimator.

Definition 8 (Simulation for estimators)**.** *Let $M = (S, s_0, O, \Delta, obs)$ be a system and $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ be an estimator for M. $\widehat{M}$ simulates M if there exists a relation $R \subseteq S \times \widehat{S}$ such that:*

$$(s_0, s_0) \in R \tag{1}$$

$$\forall (s, \widehat{s}) \in R, obs(s) = obs(\widehat{s}) \tag{2}$$

$$\forall (s, \widehat{s}) \in R, \forall t \in S \text{ s.t. } (s,t) \in \Delta, \\ \exists \widehat{t} \in \widehat{S} \text{ s.t. } (\widehat{s}, \widehat{t}) \in \widehat{\Delta} \text{ and } (t, \widehat{t}) \in R \tag{3}$$

Informally, if s is a state in S and $\widehat{s}$ a state in $\widehat{S}$, then $\widehat{s}$ simulates s (or $(s, \widehat{s})$ is in R) represents the fact that $\widehat{M}$ can estimate that the system is in state $\widehat{s}$ while it really is in state s without encountering a dead-end later in the execution.

Proposition 1. *Let M be a system and $\widehat{M}$ be an estimator for M. $\widehat{M}$ is dead-end free (or $\mathcal{L}_{obs}(M) = \mathcal{L}_{obs}(\widehat{M})$) if and only if $\widehat{M}$ simulates M.*

Proof. ($\Leftarrow$) $\mathcal{L}_{obs}(\widehat{M}) \subseteq \mathcal{L}_{obs}(M)$ as the states and transitions of $\widehat{M}$ are subsets of those of the system. Moreover, let $seq_{obs} \in \mathcal{L}_{obs}(M)$ be produced by $seq \in \mathcal{L}(M)$. We prove that for all $k < |seq|$, there exists a sequence $\widehat{seq} \in \mathcal{L}(\widehat{M})$ of length k such that $obs(seq) = obs(\widehat{seq})$ and $(seq[k], \widehat{seq}[k]) \in R$, which entails $\mathcal{L}_{obs}(\widehat{M}) \supseteq \mathcal{L}_{obs}(M)$.

($\Rightarrow$) We consider the following relation R: $\forall s \in S, \widehat{s} \in \widehat{S}$, $(s, \widehat{s}) \in R$ iff $\exists seq_{obs} \in \mathcal{L}_{obs}(M)$, $seq \in \mathcal{L}(M)$, $\widehat{seq} \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(seq) = obs(\widehat{seq}) = seq_{obs}$, $s = last(seq)$ and $\widehat{s} = last(\widehat{seq})$. We show that R is a simulation relation. (1,2) By construction of R, we have $(s_0, s_0) \in R$, and for all $(s, \widehat{s}) \in R$, $obs(s) = obs(\widehat{s})$. (3) Let $(s, \widehat{s}) \in R$ and $t \in S$ such that $(s,t) \in \Delta$. By construction of R, $\exists seq_{obs} \in \mathcal{L}_{obs}(M)$, $seq \in \mathcal{L}(M)$, $\widehat{seq} \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(seq) = obs(\widehat{seq})$, $s = last(seq)$ and $\widehat{s} = last(\widehat{seq})$. $(s,t) \in \Delta$ so $seq.t \in \mathcal{L}(M)$ and produces the sequence of observation $seq_{obs}.obs(t) \in \mathcal{L}_{obs}(M)$. By the equality of languages and definition of $\mathcal{L}_{obs}(\widehat{M})$, there exists $\widehat{seq}' \in \mathcal{L}(\widehat{M})$ *s.t.* $obs(\widehat{seq}') = seq_{obs}.obs(t)$. As $\widehat{M}$ is deterministic, seq_{obs} is generated by the same state sequence $\widehat{seq}$. So, $\widehat{seq}'$ is equal to $\widehat{seq}.\widehat{t}$ and $(\widehat{s}, \widehat{t}) \in \widehat{\Delta}$. By construction of R, $(t, \widehat{t}) \in R$. □

The following corollary presents a necessary condition for estimators. It indicates that any observation sequence that can follow a state s can also follow the state $\widehat{s}$ by which s is estimated. This corollary is used in Section 6 to increase the efficiency of estimator synthesis.

Corollary 1. *Let M be a system, $\widehat{M}$ a dead-end free estimator for M, s a state in S, $\widehat{s}$ a state in $\widehat{S}$, and R a simulation relation such that $(s, \widehat{s}) \in R$. For any observation sequence seq_{obs}, if seq_{obs} can follow s then seq_{obs} can follow $\widehat{s}$.*

Corollary 2. *The* TRACKABILITY *problem is in* NP.

Proof. Checking if $\widehat{M}$ is a dead-end free estimator for M is polynomial since it comes to checking whether $\widehat{M}$ simulates M, which can be encoded into HORN-SAT clauses as shown in [Shukla *et al.*, 1996]. □

5 Estimator Correctness

Correctness is the property of an estimator that estimates the actual system state. So far we provided a way to generate dead-end free estimators, which can be seen as a very weak form of correctness as a dead-end free estimator cannot be proven wrong by the system's observations. However, depending on the operational context, total correctness may be required or an incorrect pessimistic or optimistic estimation may be satisfying [Bouziat *et al.*, 2018; Couto *et al.*, 2020].

In this section, we model correctness as constraints and optimization criteria that allow some end-user to generate interesting estimators. Correctness constraints allow one to restrict the possible estimator states associated with some given system state. Correctness criteria let one associate a cost with each possible estimator, with the lowest cost being associated with the "most correct" estimators.

Definition 9 ((ϕ_1, ϕ_2)-correctness)**.** *Let M be a system, $\phi_1 \subseteq S$ and $\phi_2 \subseteq S$ be two sets of states of M, $\widehat{M}$ a dead-end free estimator for M, and R the associated simulation relation. $\widehat{M}$ is (ϕ_1, ϕ_2)-correct if whenever the system is in a state of ϕ_1, then the estimator is in a state of ϕ_2, i.e. $\forall (s, \widehat{s}) \in R$, if $s \in \phi_1$ then $\widehat{s} \in \phi_2$.*

In a symbolic model where states are represented by Boolean variables, correctness constraints can be represented by propositional formulae. They can also be deduced from a proof of reachability of an undesirable pair $(s, \hat{s})$ as produced in [Couto *et al.*, 2020]. With such constraints, one can require that whenever some critical fault is present in the system state, it is also present in the estimated state. This ensures the estimator is correct with respect to this critical fault.

Example 3. *Let us consider the system M presented in Example 1 and illustrated on Figure 1. Let f_1 be a fault only present in states $\phi^{f_1} = \{s_1, s_3, s_4\}$ and f_2 a fault only present in $\phi^{f_2} = \{s_5\}$. The estimator $\widehat{M_1}$ is (ϕ^{f_1}, ϕ^{f_1})-correct as all the states in ϕ^{f_1} are estimated by states in ϕ^{f_1}. However $\widehat{M_1}$ is not (ϕ^{f_2}, ϕ^{f_2})-correct as s_5 is estimated by s_3 which does not belong to ϕ^{f_2}.*

While it is very flexible, in some systems, (ϕ_1, ϕ_2)-correctness might be either too strong or too tedious to specify. For example, in Example 3, there exists no estimator that satisfies both (ϕ^{f_1}, ϕ^{f_1}) and (ϕ^{f_2}, ϕ^{f_2})-correctness. In order to relax this requirement, a first laborious approach is to enumerate the possible estimated states for each system state. Another approach consists in expressing a weaker form of correctness, for instance that (ϕ_1, ϕ_2)-correctness should be satisfied "as much as possible". In this regard we introduce a cost function that lets one model this kind of requirement.

Definition 10 (Correctness cost)**.** *Let M be a system, $\widehat{M}$ an estimator for M, and R their simulation relation. Let cost : $S \times \widehat{S} \to \mathbb{N}$ be a cost function such that $cost(s, \widehat{s})$ represents how much we want to avoid estimating that the system is in state $\widehat{s}$ when its real state is s. The correctness cost for $\widehat{M}$ is $cost(\widehat{M}) = \sum_{(s,\widehat{s})\in R} cost(s, \widehat{s})$*

Definition 11 (Correctness order)**.** *Let M be a system, $\widehat{M_1}$ and $\widehat{M_2}$ be two dead-end free estimators for this system, and cost a cost function. $\widehat{M_1}$ is strictly more correct than $\widehat{M_2}$ with respect to cost if $cost(\widehat{M_1}) < cost(\widehat{M_2})$.*

There are several ways to define a cost for pairs of states. A straightforward one is to check if the real state of the system is correctly estimated. Such a cost $c_=$ is formally defined by $\forall(s, \widehat{s}) \in S^2, c_=(s, \widehat{s}) = 0$ if $s = \widehat{s}$, 1 otherwise. One could also consider a cost based on (ϕ_1, ϕ_2)-correctness: $\forall(s, \widehat{s}) \in S^2, c_{\phi_1,\phi_2}(s, \widehat{s}) = 0$ if $s \in \phi_1$ and $\widehat{s} \in \phi_2$, 1 otherwise. Symbolic models make it easier to specify cost functions. The Hamming distance ([Hamming, 1950]) is an example of a cost function for Boolean variables. Weighted sums, or lexicographical aggregation of costs can also be used to emphasize some critical faults for example.

Example 4. *Let us consider the system M presented in Example 1 and illustrated on Figure 1. We define the dead-end free estimator $\widehat{M_3}$ similar to $\widehat{M_1}$ except that $estim_3(s_1, a) = s_5$. We consider a correctness cost c such that $\forall i, c(s_i, s_i) = 0$, $c(s_1, s_2) = c(s_2, s_1) = 1$, $c(s_5, s_3) = 1$ and $c(s_3, s_5) = 2$. This cost represents that it is preferable to estimate that the system is in state s_3 when it really is in state s_5 than the opposite. We have $c(\widehat{M_1}) = 2$ and $c(\widehat{M_3}) = 3$. This means that $\widehat{M_1}$ is more correct than $\widehat{M_3}$ with respect to cost c.*

6 Estimator Synthesis

In this section, we describe a procedure to solve the TRACKABILITY problem. To decide if a system M is single-state trackable, we try to synthesize a dead-end free estimator for it. More precisely, we consider that a system M can be seen as a graph and that a dead-end free estimator $\widehat{M}$ is a sub-graph that satisfies several constraints, that can be encoded into SMT provided the underlying theory provides graph predicates. We specifically target the solver MONOSAT ([Bayless *et al.*, 2015]), that allows to handle classical SAT constraints but also graph constraints.

Dead-end free estimator synthesis. Let $M = (S, s_0, O, \Delta, obs)$ be a system and $\widehat{M} = (\widehat{S}, s_0, O, \widehat{\Delta}, obs)$ the dead-end free estimator we try to synthesize for M. If $\widehat{M}$ exists (*i.e.* M is single-state trackable), then states and transitions of $\widehat{M}$ are subsets of those of M and following Prop. 1, there exists a simulation relation R between the states of M and $\widehat{M}$.

We associate the system M with a directed graph G whose vertices are the states S and whose edges are the transitions Δ. Similarly, $\widehat{G}$ is the graph associated with an estimator $\widehat{M}$.

First, we instantiate the following Boolean decision variables:

- for each $s \in S$, $\mathbf{v}_s$ is true iff s belongs to $\widehat{S}$;
- for each $(s,t) \in \Delta$, $\mathbf{e}_{s,t}$ is true iff (s,t) belongs to $\widehat{\Delta}$;
- for each $(s, \widehat{s})$ in S^2, $\mathbf{r}_{s,\widehat{s}}$ is true iff $(s, \widehat{s})$ belongs to R.

We next instantiate the following constraints.

$$\forall(s,t) \in \Delta, \mathbf{e}_{s,t} \to \mathbf{v}_s \wedge \mathbf{v}_t \tag{4}$$

$$\forall\widehat{s} \in S, \mathbf{v}_{\widehat{s}} \leftrightarrow reachable(\widehat{G}, s_0, \widehat{s}) \tag{5}$$

$$\forall s \in S, \forall o \in nextObs(s), \sum_{t\in S|obs(t)=o} \mathbf{e}_{s,t} \leq 1 \tag{6}$$

$$\forall s \in S, \forall o \in nextObs(s), \mathbf{v}_s \to \bigvee_{t\in S|obs(t)=o} \mathbf{e}_{s,t} \tag{7}$$

$$\mathbf{r}_{s_0,s_0} \tag{8}$$

$$\forall s \in S, \bigvee_{\widehat{s}\in S, obs(s)=obs(\widehat{s})} \mathbf{r}_{s,\widehat{s}} \tag{9}$$

$$\forall(s, \widehat{s}) \in S^2 \text{ s.t. } obs(s) = obs(\widehat{s}), \forall t \in S \text{ s.t. } (s,t) \in \Delta,$$
$$\mathbf{r}_{s,\widehat{s}} \to \bigvee_{(\widehat{s},\widehat{t})\in\Delta, obs(t)=obs(\widehat{t})} (\mathbf{r}_{t,\widehat{t}} \wedge \mathbf{e}_{\widehat{s},\widehat{t}}) \tag{10}$$

$$\forall\widehat{s} \in S, \mathbf{v}_{\widehat{s}} \leftrightarrow \bigvee_{s\in S|obs(s)=obs(\widehat{s})} \mathbf{r}_{s,\widehat{s}} \tag{11}$$

We first encode that the estimator is a well-built automaton. Constraint (4) states that if an edge is selected in $\widehat{G}$, its vertices are selected as well. Constraint (5) enforces the states of the estimator to be exactly the ones reachable from s_0 in the graph $\widehat{G}$. We use here the MONOSAT predicate *reachable*.

Constraints (6) and (7) force $\widehat{M}$ to be deterministic: for every state s and every successor observation o, exactly one outgoing transition leads to a state with observation o. We use two constraints, as in SAT/SMT the "at most one" cardinality constraint (6) cannot easily be combined with the implication of Constraint (7).

We next consider constraints related to the simulation relation. Constraint (8) states that the initial state simulates itself (see Definition 8). Constraint (9) encodes that every state

of the system is simulated by at least one state of the estimator. Constraint (10) encodes the simulation relation from Definition 8 and also states that the edge associated with the simulation relation must belong to $\widehat{M}$[1].

Constraint (11) makes the simulation relation minimal as the estimator states must all simulate at least one system state.

Using a SMT solver let us use the predicate *reachable* in constraint (5). This constraint can be expressed in SAT clauses using for instance the *Floyd-Warshall* algorithm [Cormen *et al.*, 2001] with a complexity of $\mathcal{O}(|V|^3)$. MONOSAT achieves a complexity of $\mathcal{O}(|V| \cdot |E|)$ [Bayless *et al.*, 2015].

From Definition 8 and Corollary 1, the number of variables $\mathbf{r}_{s,\widehat{s}}$ can be reduced by creating variables only for pairs $(s, \widehat{s})$ for which $\widehat{s}$ accepts all the observation sequences of length 2 (or more) that s accepts.

Correct estimator synthesis. The encoding presented above allows to synthesize dead-end free estimators. It is possible to enrich it to synthesize correct estimators, as defined in Section 5. (ϕ_1, ϕ_2)-correctness can be encoded as a hard constraint the following way: $\forall s \in \phi_1, \forall \widehat{s} \in S \setminus \phi_2, \neg \mathbf{r}_{s,\widehat{s}}$. The second correctness requires a cost function c as input, and is encoded as the following pseudo Boolean criterion: $\textbf{minimize} \sum_{(s,\widehat{s}) \in R} \mathbf{r}_{s,\widehat{s}} \cdot c(s, \widehat{s})$.

7 Experiments

Our implementation uses the SCALA programming language, and each experiment was performed on a computer with a Intel® Xeon® CPU E5-2699 v3 @ 2.30GHz processor and a limit of 16GB of memory.

Examples from the literature. We have encoded benchmarks from three system models from the literature:

- *valve controller* [Sampath *et al.*, 1995] that models a valve controller for heating, ventilation and air conditioning units with a valve that can be stuck closed or stuck open, a pump that can fail in modes on and off and a controller; each component has 4 possible states; this system is not trackable.
- *valve driver* [Williams and Nayak, 1996] that models a valve driver that has 4 modes, 6 command inputs and 3 command outputs; this system is trackable.
- *baggage transfer* [Pencolé *et al.*, 2018] that models a baggage transfer system composed of 2 conveyors, 1 piston, 1 controller and 4 sensors. No fault model is provided, so we enriched each component with a permanent fault model. By enabling or not the fault model for each of the 8 components, we managed to create 218 systems among 256, all of which are trackable.

Random examples. In order to test our prototype intensively, we randomly generated a set of systems as follows. First, we generated a set of small systems of 5 states, 3 observations, and 0.5 transition probability for any two states. Second, we aggregated them by Cartesian product (of states and observations), with a 0.1 probability that any two states are incompatible (in this case their pair is removed from the aggregated states). The benchmark contains systems obtained by aggregation of 4 and 5 small systems. Finally, we selected only "interesting" systems, such that:

1. the system is trackable;
2. the estimator has less than half the number of states of the system (so as to reject trivially trackable systems);
3. there exists a state s such that the system has no $(\{s\}, S \setminus \{s\})$-correct estimator.

This method generated systems from 10 to 1461 states, with more small systems than large ones.

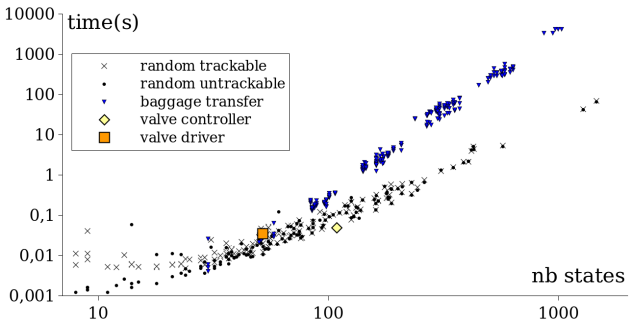


Figure 3: Computation time *w.r.t.* number of system states. Both scales are logarithmic.

Analysis. In all our experiments, most of the computation time is used for generating SMT clauses whereas the time required by MONOSAT to solve the SMT instance is negligible. This explains why the computation time is nearly the same for the trackable and untrackable versions of random examples. We have yet to properly assess the impact of a cost function on the solving time. Figure 3 shows that on random benchmarks, the solving time grows regularly in the number of states. The *valve driver* and *valve controller* systems require similar computation time as random systems. For the *baggage transfer* system, the computation time is higher, because of the large number of symmetries in the system.

8 Conclusion

This paper describes a new approach for checking single-state trackability as defined in [Bouziat *et al.*, 2019]. As a system is simulated by its estimator, we prove that the TRACKABILITY problem is in NP. We also define two customizable correctness types for modelling relevant estimators. We solve the TRACKABILITY problem by reducing it to a MONOSAT SMT query, along with correctness requirements. The approach is validated on a set of benchmarks both from the literature and from randomly generated problems.

Studying NP-completeness with and without correctness constraints is a natural next step in our work. Allowing for delayed estimation, or using a bounded number of memorized states at each time step could help apply this approach to a larger range of systems, including real-world systems, and will be considered in future work.

Symbolic representation of estimators with preferences as in [Bouziat *et al.*, 2018] is also a possible extension.

[1]Note that Constraint (9) is redundant with Constraints (8) and (10), but improves the solver's performance.

References

[Bayless *et al.*, 2015] Sam Bayless, Noah Bayless, Holger H. Hoos, and Alan J. Hu. SAT modulo monotonic theories. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*, pages 3702—3709, 2015.

[Bonet *et al.*, 2009] Blai Bonet, Hector Palacios, and Hector Geffner. Automatic derivation of memoryless policies and finite-state controllers using classical planners. In *Proceedings of the 19th International Conference on Automated Planning and Scheduling*, pages 34–41, 2009.

[Bouziat *et al.*, 2018] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Preferential discrete model-based diagnosis for intermittent and permanent faults. In *Proceedings of the 29th International Workshop on Principles of Diagnosis*, 2018.

[Bouziat *et al.*, 2019] Valentin Bouziat, Xavier Pucel, Stéphanie Roussel, and Louise Travé-Massuyès. Single state trackability of discrete event systems. In *Proceedings of the 30th International Workshop on Principles of Diagnosis*, 2019.

[Cormen *et al.*, 2001] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Transitive closure of a directed graph*, pages 993–1024. The MIT Press, 2001.

[Couto *et al.*, 2020] Diego Couto, Kévin Delmas, and Xavier Pucel. On the safety assessment of RPAS safety policy. *Proceedings of the 9th European Congress Embedded Real Time Software and Systems*, 2020.

[Dague *et al.*, 2019] Philippe Dague, Lulu He, and Lina Ye. How to be sure a faulty system does not always appear healthy?: Fault manifestability analysis for discrete event and timed systems. *Innovations in Systems and Software Engineering*, 2019.

[Doyen and Raskin, 2011] Laurent Doyen and Jean-François Raskin. Games with imperfect information: Theory and algorithms. In *Lectures in Game Theory for Computer Scientists*, pages 185—-212, 2011.

[Hamming, 1950] Richard W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[Holík *et al.*, 2018] Lukáš Holík, Ondřej Lengál, Juraj Síč, Margus Veanes, and Tomáš Vojnar. Simulation algorithms for symbolic automata. In *Automated Technology for Verification and Analysis*, pages 109–125, 2018.

[Hüttel and Shukla, 1996] Hans Hüttel and Sandeep Shukla. On the Complexity of Deciding Behavioural Equivalences and Preorders. A Survey. *BRICS Report Series*, 3(39), 1996.

[Kurien and Nayak, 2000] James Kurien and P. Pandurang Nayak. Back to the future for consistency-based trajectory tracking. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence*, pages 370–377, 2000.

[Lafortune *et al.*, 2018] Stéphane Lafortune, Feng Lin, and Christoforos N. Hadjicostis. On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45:257–266, 2018.

[Meuleau *et al.*, 1999] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence*, page 427–436, 1999.

[Pencolé *et al.*, 2018] Yannick Pencolé, Gerald Steinbauer, Clemens Mühlbacher, and Louise Travé-Massuyès. Diagnosing discrete event systems using nominal models only. In *28th International Workshop on Principles of Diagnosis*, pages 169–183, 2018.

[Pralet *et al.*, 2010] Cédric Pralet, Gérard Verfaillie, Michel Lemaître, and Guillaume Infantes. Constraint-based controller synthesis in non-deterministic and partially observable domains. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 681–686, 2010.

[Ramadge and Wonham, 1987] Peter J. G. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[Sampath *et al.*, 1995] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995.

[Shu *et al.*, 2007] Shaolong Shu, Feng Lin, and Hao Ying. Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12):2356–2359, 2007.

[Shukla *et al.*, 1996] Sandeep Shukla, Daniel Rosenkrantz, Harry Iii, and Richard Stearns. The polynomial time decidability of simulation relations for finite state processes: A HORNSAT based approach. *Satisfiability Problem: Theory and applications*, 1996.

[Su *et al.*, 2014] Xingyu Su, Yannick Pencolé, and Alban Grastien. Window-based diagnostic algorithms for discrete event systems: What information to remember. In *Proceedings of the 25th International Workshop on Principles of Diagnosis*, 2014.

[Torta and Torasso, 2007] Gianluca Torta and Pietro Torasso. An on-line approach to the computation and presentation of preferred diagnoses for dynamic systems. *AI Communications*, 20(2):93–116, 2007.

[Williams and Nayak, 1996] Brian C. Williams and P. Pandurang Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, pages 971–978, 1996.

[Zaytoon and Lafortune, 2013] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

Bibliographie

[Bayless et al., 2015] Bayless, S., Bayless, N., Hoos, H., and Hu, A. (2015). SAT Modulo Monotonic Theories. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

[Bayoudh and Travé-Massuyès, 2014] Bayoudh, M. and Travé-Massuyès, L. (2014). Diagnosability analysis of hybrid systems cast in a discrete-event framework. *Discrete Event Dynamic Systems*, 24.

[Belard et al., 2011] Belard, N., Pencolé, Y., and Combacau, M. (2011). A theory of meta-diagnosis : Reasoning about diagnostic systems. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence IJCAI'11*, pages 731–737, Barcelona, Spain.

[Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). Cp-nets : A tool for representing and reasoning with conditional ceteris paribus preference statements. *J. Artif. Intell. Res.(JAIR)*, 21 :135–191.

[Bouziat et al., 2018] Bouziat, V., Pucel, X., Roussel, S., and Travé-Massuyès, L. (2018). Preferential discrete model-based diagnosis for intermittent and permanent faults. In *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'18*, Warsow, Poland. CEUR Workshops Proceedings.

[Bouziat et al., 2019a] Bouziat, V., Pucel, X., Roussel, S., and Travé-Massuyès, L. (2019a). Preference-based fault estimation in autonomous robots : Incompleteness and meta-diagnosis - extended abstract. In *Proceedings of the 18th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2019)*, pages 1841–1843, Montreal, Canada.

[Bouziat et al., 2019b] Bouziat, V., Pucel, X., Roussel, S., and Travé-Massuyès, L. (2019b). Single state trackability of discrete event systems. In *Proceedings of the 30th International Workshop on Principles of Diagnosis DX'19*, Klagenfurt, Austria. CEUR Workshops Proceedings.

[Cassandras and Lafortune, 2009] Cassandras, C. G. and Lafortune, S. (2009). *Introduction to discrete event systems*. Springer Science & Business Media.

[Ceballos et al., 2005] Ceballos, R., Pozo, S., Del Valle, C., and Gasca, R. M. (2005). An integration of fdi and dx techniques for determining the minimal diagnosis in an automatic way. In Gelbukh, A., de Albornoz, Á., and Terashima-Marín, H., editors, *MICAI 2005 : Advances in Artificial Intelligence*, pages 1082–1092, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Cimatti et al., 2000] Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (2000). Nusmv : a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4) :410–425.

[Cimatti et al., 2003] Cimatti, A., Pecheur, C., and Cavada, R. (2003). Formal verification of diagnosability via symbolic model checking. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 363–369. Morgan Kaufmann Publishers Inc.

[Clarke and Emerson, 1982] Clarke, E. M. and Emerson, E. A. (1982). Design and synthesis of synchronization skeletons using branching time temporal logic. In Kozen, D., editor, *Logics of Programs*, pages 52–71, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Clarke et al., 1999] Clarke, Jr., E. M., Grumberg, O., and Peled, D. A. (1999). *Model Checking*. MIT Press, Cambridge, MA, USA.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA. Association for Computing Machinery.

[Coquand et al., 2020] Coquand, C., Pucel, X., Roussel, S., and Travé-Massuyès, L. (2020). Dead-end free single state multi-estimators for des – the 2-estimator case. In *Proceedings of the 31th International Workshop on Principles of Diagnosis DX-2020*. CEUR Workshops Proceedings.

[Cordier et al., 2014] Cordier, M.-O., Dague, P., Pencolé, Y., and Travé-Massuyès. (2014). Diagnostic et supervision : approches à base de modèles. In *Représentation des connaissances et formalisation des raisonnements*, volume 1 of *Panorama de l'Intelligence Artificielle, ses bases méthodologiques, ses développements*, chapter 18, pages 555–589. Cépaduès éditions.

[Couto et al., 2020] Couto, D., Delmas, K., and Pucel, X. (2020). On the safety assessment of rpas safety policy. *To appear in Proceedings of the 9th European Congress Embedded Real Time Software and Systems (ERTS2020)*.

[deKleer and Williams, 1987] deKleer, J. and Williams, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence*, 32(1) :97 – 130.

[Emerson, 1990] Emerson, E. A. (1990). Chapter 16 - temporal and modal logic. In VAN LEEUWEN, J., editor, *Formal Models and Semantics*, Handbook of Theoretical Computer Science, pages 995 – 1072. Elsevier, Amsterdam.

[Fabre and Jezequel, 2010] Fabre, E. and Jezequel, L. (2010). On the construction of probabilistic diagnosers. *IFAC Proceedings Volumes*, 43(12) :229 – 234. 10th IFAC Workshop on Discrete Event Systems.

[Felfernig and Schubert, 2010] Felfernig, A. and Schubert, M. (2010). Fastdiag : A diagnosis algorithm for inconsistent constraint sets. In *Proceedings of the 21st International Workshop on the Principles of Diagnosis (DX 2010), Portland, OR, USA*, pages 31–38.

[Forney, 1973] Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3) :268–278.

[Genesereth, 1984] Genesereth, M. R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24(1) :411 – 436.

[Grastien et al., 2005] Grastien, A., Cordier, M., and Largouët, C. (2005). Incremental diagnosis of discrete-event systems. In *Proceedings of the 29th International Workshop on Principles of Diagnosis DX'05*, Pacific Grove, CA, USA.

[Grastien and Zanella, 2019] Grastien, A. and Zanella, M. (2019). *Discrete-Event Systems Fault Diagnosis*, pages 197–234.

[Hamming, 1950] Hamming, R. (1950). Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2) :147–160.

[Huttel and Shukla, 1996] Huttel, H. and Shukla, S. (1996). On the complexity of deciding behavioural equivalences and preorders – a survey. *BRICS Report Series*, 3.

[Jackson, 2006] Jackson, D. (2006). *Software Abstractions : Logic, Language, and Analysis*. The MIT Press.

[Jiang et al., 2001] Jiang, S., Huang, Z., Chandra, V., and Kumar, R. (2001). A polynomial algorithm for testing diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 46(8) :1318–1321.

[Jéron et al., 2008] Jéron, T., Marchand, H., Genc, S., and Lafortune, S. (2008). Predictability of sequence patterns in discrete event systems. *IFAC Proceedings Volumes*, 41(2) :537 – 543. 17th IFAC World Congress.

[Jéron et al., 2006] Jéron, T., Marchand, H., Pinchinat, S., and Cordier, M.-O. (2006). Supervision patterns in discrete event systems diagnosis. In *Discrete event systems, 2006 8th international workshop on*, pages 262–268. IEEE.

[Kripke, 1965] Kripke, S. A. (1965). Semantical analysis of intuitionistic logic i. In Crossley, J. and Dummett, M., editors, *Formal Systems and Recursive Functions*, volume 40 of *Studies in Logic and the Foundations of Mathematics*, pages 92 – 130. Elsevier.

[Kurien and Nayak, 2000] Kurien, J. and Nayak, P. (2000). Back to the future for consistency-based trajectory tracking. In *Proceedings of the 17th AAAI Conference on Artificial Intelligence*, pages 370–377, Austin, Texas, USA.

[Le Berre and Parrain, 2010] Le Berre, D. and Parrain, A. (2010). The sat4j library, release 2.2, system description. *Journal on Satisfiability, Boolean Modeling and Computation*, 7 :59–64.

[Macedo et al., 2016] Macedo, N., Brunel, J., Chemouil, D., Cunha, A., and Kuperberg, D. (2016). Lightweight specification and analysis of dynamic systems with rich configurations. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 373–383, New York, NY, USA. ACM.

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

[Merz, 2003] Merz, S. (2003). On the Logic of TLA+. *Computers and Informatics*, 22(3-4) :351–379.

[Moore, 1956] Moore, E. (1956). Gedanken-experiments on sequential machines. In Shannon, C. and McCarthy, J., editors, *Automata Studies*, pages 129–153. Princeton University Press, Princeton, NJ.

[Morgado and Marques-Silva, 2005] Morgado, A. and Marques-Silva, J. (2005). Good learning and implicit model enumeration. In *Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence ICTAI'05*, pages 6–pp, Hong Kong, China. IEEE.

[Moskewicz et al., 2001] Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001). Chaff : Engineering an efficient sat solver. In *Proceedings of the 38th Annual Design Automation Conference*, DAC '01, page 530–535, New York, NY, USA. Association for Computing Machinery.

[Ozveren and Willsky, 1990] Ozveren, C. M. and Willsky, A. S. (1990). Observability of discrete event dynamic systems. *IEEE Transactions on Automatic Control*, 35(7) :797–806.

[Pencole, 2005] Pencole, Y. (2005). Assistance for the design of a diagnosable component-based system. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 8 pp.–556.

[Pencolé et al., 2018] Pencolé, Y., Steinbauer, G., Mühlbacher, C., and Travé-Massuyès, L. (2018). Diagnosing discrete event systems using nominal models only. In Zanella, M., Pill, I., and Cimatti, A., editors, *28th International Workshop on Principles of Diagnosis (DX'17)*, volume 4 of *Kalpa Publications in Computing*, pages 169–183. EasyChair.

[Petri, 1966] Petri, C. A. (1966). *Communication with automata*. PhD thesis, Universität Hamburg.

[Pnueli, 1977] Pnueli, A. (1977). The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57.

[Pralet et al., 2016] Pralet, C., Pucel, X., and Roussel, S. (2016). Diagnosis of intermittent faults with conditional preferences. In *Proceedings of the 27th International Workshop on Principles of Diagnosis (DX'16)*.

[Reiter, 1987] Reiter, R. (1987). A theory of diagnosis from first principles. *Artificial intelligence*, 32(1) :57–95.

[Roussel et al., 2020] Roussel, S., Pucel, X., Bouziat, V., and Travé-Massuyès, L. (2020). Model-based synthesis of incremental and correct estimators for discrete event systems. In Bessiere, C., editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1884–1890. International Joint Conferences on Artificial Intelligence Organization. Main track.

[Sampath et al., 1995] Sampath, M., Sengupta, R., Lafortune, S., Sinnamohideen, K., and D.Teneketzis (1995). Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9) :1555–1575.

[Schneider, 2003] Schneider, K. (2003). *Verification of Reactive Systems – Formal Methods and Algorithms*.

[Shu et al., 2007] Shu, S., Lin, F., and Ying, H. (2007). Detectability of discrete event systems. *IEEE Transactions on Automatic Control*, 52(12) :2356–2359.

[Su et al., 2014] Su, X., Pencolé, Y., and Grastien, A. (2014). Window-based Diagnostic Algorithms for Discrete Event Systems : What Information to Remember. In *25th International Workshop on Principles of Diagnosis (DX 14)*, Graz, Austria.

[Thorsley and Teneketzis, 2005] Thorsley, D. and Teneketzis, D. (2005). Diagnosability of stochastic discrete-event systems. *IEEE Transactions on Automatic Control*, 50(4) :476–492.

[V.Verma et al., 2005] V.Verma, T.Estlin, A.Jónsson, Pasareanu, C., Simmons, R., and Tso, K. (2005). Plan execution interchange language (plexil) for executable plans and command sequences. In *Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space iSAIRAS'05*, Munich, Germany.

[Williams and Nayak, 1996] Williams, B. and Nayak, P. (1996). A model-based approach to reactive self-configuring systems. In *Proceedings of the 13th AAAI Conference on Artificial Intelligence*, Portland, Oregon.

[Ye et al., 2016] Ye, L., Dague, P., Longuet, D., Brandán Briones, L., and Madalinski, A. (2016). Fault Manifestability Verification for Discrete Event Systems. In *22nd European Conference on Artificial Intelligence ECAI-16*, La Haye, Netherlands.

[Zaytoon and Lafortune, 2013] Zaytoon, J. and Lafortune, S. (2013). Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2) :308 – 320.