Calcul de trajectoires utilisant les propriétés d'interversibilité

Marie-Odile Cordier[1], Alban Grastien[1], Christine Largouët[1], Yannick Pencolé[1]

IRISA, Campus de Beaulieu, 35042 Rennes Cedex, France
{cordier,agrastie,clargoue}@irisa.fr
Yannick.Pencole@anu.edu.au

Résumé Le temps de calcul des trajectoires sur un modèle de comportement du système est un problème critique rencontré aussi bien en diagnostic qu'en planification Dans le but d'améliorer l'efficacité de cette tâche, un intérêt croissant est porté aux techniques de model-checking développées dans le domaine de la vérification automatique. Dans cet article, nous proposons de représenter le système par un automate, et nous définissons une nouvelle propriété appelée *interversibilité*. Cette propriété est utilisée pour améliorer l'efficacité de l'algorithme de recherche calculant les trajectoires. Nous présentons deux exemples dans les domaines du diagnostic et de la planification où cette approche donne des résultats satisfaisants.

Mots clef Diagnostic – Planification – Model-checking – Interversibilité

1 Introduction

Il est généralement reconnu que le diagnostic de systèmes dynamiques, représentés comme des systèmes à événements discrets (SED) [CL99] revient à trouver ce qui est arrivé au système à l'aide des observations existantes. On peut trouver différentes terminologies dans la littérature, tels *histories* [BLPZ99], *scenarios* [CT94], *narratives* [BMS00], *consistant paths* [CPR00]. Tous ces termes reposent sur l'idée que la tâche de diagnostic consiste à déterminer les trajectoires (une séquence d'états et d'événements) expliquant la séquence d'observations. De même, planifier consiste à trouver une séquence d'actions, appelée plan, qui conduit le système vers un certain but. Diagnostiquer et planifier peuvent donc tous les deux être vus comme le problème de trouver un chemin dans un modèle comportemental. La principale difficulté est la taille de ce modèle (nombre d'états et de transitions) et le nombre potentiellement très grand de trajectoires. Ceci explique l'intérêt que portent actuellement les communautés de diagnostic et de planification à l'utilisation de techniques de model-checking, créées initialement pour tester de manière efficace les systèmes temps-réels complexes. Une fois le système à diagnostiquer, ou le domaine à planifier représenté par un automate à états finis, le problème de trouver une trajectoire est exprimé comme une recherche d'atteignabilité sur le modèle [CL01]. Pour réduire le problème d'*explosion du nombre d'états*, des techniques ont été proposées tels les diagrammes de décision binaires (Binary Decision Diagrams, BDD) [BCM+92], la réduction d'ordre partiel (Partial Order Reduction, POR) [CGMP98]. Elles ont récemment été utilisées dans le domaine de la planification (utilisation des BDD dans [CR00]) et dans celui du diagnostic (utilisation de BDD dans [MR02] et de POR dans [Pen02]).

Dans cet article, nous proposons d'utiliser une propriété sur le modèle décrivant le SED pour réduire l'espace de recherche sans perdre d'information. Cette propriété est appelée interversibilité et est définie sur les événements (ou les actions) du système. Intuitivement, deux événements sont dits interversibles si deux séquences d'événements ne différant que dans l'ordre de ces deux événements (pas forcément successifs), conduisent toujours au même état du système.

L'article est structuré comme suit. Tout d'abord, nous présentons brièvement le formalisme des automates et définissons la notion de trajectoire. Ensuite, nous proposons deux exemples jouets en diagnostic et en planification qui serviront pour l'illustration et l'expérimentation dans la suite de l'article. La propriété d'interversibilité est ensuite définie et l'algorithme de recherche est présenté. Enfin, les résultats expérimentaux sont analysés et les perspectives sont présentées.

2 Formalisme du modèle

Dans cette section, nous commençons par rappeler le formalisme des automates que nous utilisons pour représenter les systèmes à événements discrets avant d'introduire les points théoriques nécessaires pour définir une trajectoire. Ce formalisme est illustré par deux exemples dans la section 3.

Définition 1 (Automate) *Un automate est un tuple* $\mathcal{A} = \langle Q, E, T, q_o \rangle$ *où :*
- Q *est un ensemble fini d'états,*
- E *est un ensemble fini d'étiquettes de transitions appelées événements*[1]*,*
- $T \subseteq Q \times E \times Q$ *est un ensemble fini de transitions sur le système. Une transition* t *est un triplet* (q_1, e, q_2) *tel que* t *relie* $q_1 \in Q$ *vers* $q_2 \in Q$ *et est étiquetée par* $e \in E$*,*
- q_o *est l'état initial de l'automate.*

L'automate représente l'ensemble des états du système et décrit l'évolution de l'état courant par rapport à l'occurence des événements sur le système. Dans l'article, nous considèrons uniquement le cas des automates déterministes.

Un système étant un ensemble de composants interconnectés, il est souvent plus simple de décrire le comportement de chacun des composants élémentaires. Le modèle global est obtenu par l'opération de composition avec synchronisation sur les événements.

Définition 2 (Produit synchronisé) *Le produit synchronisé de* n *automates* $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{io} \rangle$*, noté* $\otimes_{i=1,n} \mathcal{A}_i$*, est un automate* $\mathcal{A} = \langle Q, E, T, q_o \rangle$ *tel que :*

[1] Dans les domaines applicatifs, les étiquettes de transitions sont associées aux événements en diagnostic et aux actions en planification. Dans la suite de cet article, nous utiliserons le terme générique *événement*.

- $Q = Q_1 \times \cdots \times Q_n$,
- $E = \bigcup_{1 \leq i \leq n} E_i$,
- $T = \left\{ \begin{array}{c} \{(q_1, \cdots, q_n), e, (q'_1, \cdots, q'_n)\} \mid \\ \forall i((e \in E_i \Rightarrow (q_i, e, q'_i) \in T_i) \wedge \\ (e \notin E_i \Rightarrow q_i = q'_i)) \end{array} \right\}$,
- $q_o = (q_{1o}, \ldots, q_{no})$.

Les propriétés associées aux événements et aux séquences d'événements sont les suivantes.

Définition 3 (Événement applicable) *Un événement e est applicable dans un état $q \in Q$ d'un automate $\mathcal{A} = \langle Q, E, T, q_o \rangle$ s'il existe une transition depuis q étiquetée par e, i.e :* $en_e^{\mathcal{A}}(q) = \begin{cases} vrai \quad si \ \ (\exists q' \mid (q, e, q') \in T) \vee (e \notin E)) \\ faux \ \ sinon \end{cases}$

Le théorème suivant peut être facilement prouvé.

Théorème 1 *Dans un automate $\mathcal{A} = \langle Q, E, T, q_o \rangle = \otimes_{i=1,n} \mathcal{A}_i$, avec $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{io} \rangle$, un événement e est applicable dans un état $q = (q_1, \cdots, q_n)$ s'il est applicable dans l'état q_i de l'automate $\mathcal{A}_i$ pour tout i, i.e :* $en_e^{\mathcal{A}}(q) = en_e^{\mathcal{A}_1}(q_1) \wedge \cdots \wedge en_e^{\mathcal{A}_n}(q_n)$

On note $e(q)$, avec $e \in E$ et $q \in Q$, l'état $q' \in Q$ atteint depuis l'état q par la transition étiquetée par e tel que $(q, e, q') \in T$.

Une *séquence* sur un ensemble d'événements E est une séquence γ_1 ; γ_2 ; ...; γ_n où chaque γ_i est soit un événement, soit une séquence d'événements. Dans la suite, les séquences d'événements sont représentées par des lettres grecques α, $\beta \ldots$ et ε représente la séquence vide.

Définition 4 (Séquence applicable) *Une séquence d'événements $\alpha = e_1; e_2; \cdots; e_k$ est applicable dans un état q de l'automate $\mathcal{A}$, noté $en_\alpha^{\mathcal{A}}(q)$, ssi $en_{e_1}^{\mathcal{A}}(q) \wedge en_\beta^{\mathcal{A}}(e_1(q))$ avec $\alpha = e_1; \beta$ et $\beta = e_2; \cdots; e_k$.*

Une séquence vide est applicable quelque soit l'état du système, et nous avons donc : $\forall q \in Q, \ \ en_\varepsilon^{\mathcal{A}}(q) = vrai$.

Définition 5 (Chemin) *Une séquence applicable en un état q est appelée* chemin partant de *q.*

Définition 6 (Trajectoire) *Un chemin est appelé* trajectoire *de $\mathcal{A}$ ssi il s'agit d'un chemin partant de l'état initial q_o.*

3 Exemples

Nous présentons maintenant deux exemples, dans les domaines du diagnostic et de la planification, qui ont été utilisés pour les expérimentations de la section 6. Ces exemples sont illustrés par des figures. Les états sont représentés par des cercles, et les transitions par des flèches. L'état initial est indiqué par une flèche ne provenant d'aucun état.

Système de diagnostic

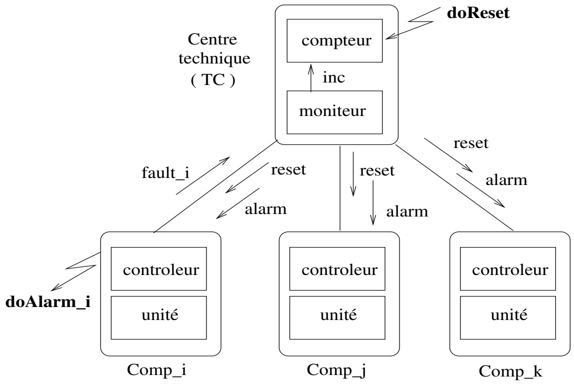

Fig.1. Un système de télécommunication simplifié

Considérons un système de télécommunication simplifié composé d'un ensemble de composants *Comp_i* et d'un centre technique *TC* qui reçoit des messages des composants (voir figure 1). Un composant peut être dans un comportement normal ou anormal dû à une panne. Un composant est constitué de deux parties : le composant en lui-même appelé l'unité et un contrôleur qui détecte les comportements anormaux de l'unité. Quand une panne a lieu, le contrôleur envoie un message *fault_i* au *TC* et place l'unité en état anormal. Le *TC* est également constitué de deux parties : un moniteur et un compteur. Quand le moniteur reçoit un message de panne depuis un contrôleur, il incrémente le compteur. Une horloge externe contrôle la remise à zéro de la valeur du compteur. Quand cet événement exogène (*doReset*) est reçu, le centre technique *TC* demande, en envoyant le message *reset*, à tous les composants de retourner à l'état normal. Entre deux remises à zéro, si le compteur atteint une certaine valeur prédéfinie, considérée comme le nombre maximum de composants en panne accepté par le système, le *TC* envoie le message *alarm* à tous les composants. Quand un contrôleur, dont l'unité est en panne, reçoit un message *alarm*, il envoie le message exogène *doAlarm_i* à un superviseur avant de remettre l'unité en fonctinonement normal. Le diagnostic consiste à retrouver les trajectoires du système qui expliquent les alarmes observées par le superviseur (*doAlarm_i*, *doReset*).

Pour modéliser le système, nous modélisons chaque composant. Le moniteur du *TC* (voir figure 2) est défini par deux états. Le moniteur passe de l'état 1 à l'état 2 lorsqu'il reçoit un message de panne d'un des composants. Après avoir envoyé un événement *inc* au compteur, il retourne à l'état initial. Pour représenter le comportement du compteur (voir figure 3), le nombre maximum de composants défecteux a été fixé à 3. Les états 0, 1, 2 et 3 correspondent à la valeur du compteur. Quand le compteur reçoit le message exogène *doReset*, il passe

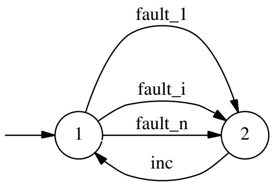

Fig.2. Modèle du moniteur du *TC*

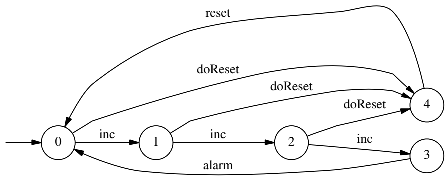

Fig.3. Modèle du compteur du *TC*

dans l'état 4, envoie l'événement *reset* à tous les composants et retourne à l'état initial.

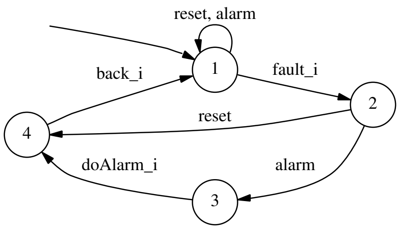

Fig.4. Modèle d'un contrôleur de composant

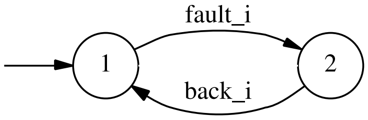

Fig.5. Modèle d'une unité de composant

Le contrôleur d'un composant (voir figure 4) reçoit l'événement $fault\_i$ de l'unité et se place dans l'état 2. Dans cet état, deux événements peuvent être reçus : *reset* ou *alarm*. L'événement *reset* conduit le contrôleur à l'état 4. Dans le cas où l'événement *alarm* est reçu, une alarme *doAlarm_i* est envoyée et le contrôleur revient à l'état 4. L'unité du composant (figure 5) peut être dans un état normal ou anormal (si une panne a eu lieu). L'événement *back_i*, envoyé par le contrôleur, fait revenir l'unité dans un état normal.

Système de planification

L'exemple de planification est une variation du domaine *Bomb in the toilet* de Moore [McD87] repris dans l'article de Cimatti et Roveri [CR00]. Nous supposons initialement qu'il y a n valises dans une salle d'eau, que certaines contiennent une bombe armée, et que nous ne savons pas si les toilettes sont bouchées ou non. Le nombre de valises contenant une bombe est inconnu. Le but est de désarmer toutes les bombes et que les toilettes soient finalement non bouchées. Le seul moyen de désarmer une bombe est de la jeter dans les toilettes (action *dunk_i*), si les toilettes ne sont pas bouchées. Jeter une valise a pour effet incertain de boucher les toilettes. L'action *flush* permet de déboucher les toilettes.

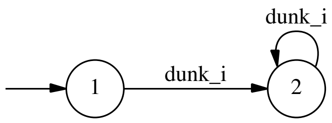

Fig.6. Modèle d'une valise

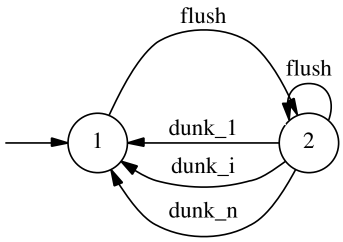

Fig.7. Modèle des toilettes

Le comportement du système est représenté par deux automates. Le premier (voir figure 6) est associé à une valise. Le deuxième (voir figure 7) décrit les toilettes. Dans l'automate de la valise, l'état 1 indique que la valise contient une bombe potentiellement armée alors qu'elle est désarmée dans l'état 2. L'action *dunk_i* désarme la bombe si la valise en contient une. Dans la figure 7, dans l'état 1, nous ne savons pas si les toilettes sont bouchées ou non, mais dans l'état 2, nous savons qu'elles sont débouchées. Jeter une valise peut boucher les toilettes, ce qui est représenté par les transitions de l'état 2 à l'état 1. La transition étiquetée par $flush$ indique que l'action $flush$ permet de déboucher les toilettes.

4 Interversibilité

Dans cette section, nous définissons la propriété d'interversibilité entre événements. Dans la section suivante, nous présentons un algorithme utilisant cette propriété pour améliorer l'efficacité du calcul de trajectoires.

Intuitivement, cette propriété indique que deux événements, sous certaines conditions, peuvent être intervertis dans une séquence, sans aucune conséquence sur l'état final atteint par le système.

Soit $\mathcal{L}$, un langage composé de séquences d'événements.

Définition 7 *Deux événements a et b ($a \neq b$) satisfont la propriété d'interversibilité sur le langage $\mathcal{L}$ dans un automate $\mathcal{A} = \langle Q, E, T, q_o \rangle$ (noté $\overset{a \leftrightarrow b}{\mathcal{L}}$) ssi $\forall q \in Q, \forall \beta \in \mathcal{L}$,*

- $en^{\mathcal{A}}_{a;\beta;b}(q) \Leftrightarrow en^{\mathcal{A}}_{b;\beta;a}(q)$, *et*
- $en^{\mathcal{A}}_{a;\beta;b}(q) \Rightarrow a;\beta;b(q) = b;\beta;a(q)$.

Cette définition signifie que si $\overset{a \leftrightarrow b}{\mathcal{L}}$, alors pour toutes séquences $\beta \in \mathcal{L}$, la séquence d'événements $a;\beta;b$ a le même effet sur le système que la séquence $b;\beta;a$.

Cette propriété est compositionnelle, comme le montre le théorème suivant :

Théorème 2 *Si deux événements a et b peuvent être intervertis sur les langages $\mathcal{L}_i$ dans les automates $\mathcal{A}_i$ ($\overset{a\leftrightarrow b}{\mathcal{L}_i}$ dans $\mathcal{A}_i$, $\forall i \in \{1, \ldots, n\}$), alors ces deux événements peuvent être intervertis sur le langage $\mathcal{L} = \bigcap_{1\leq i\leq n} \mathcal{L}_i$ dans l'automate $\mathcal{A} = \otimes \mathcal{A}_i$.*

Preuve : Soient a et b deux événements (avec $a \neq b$) satisfiant $\overset{a\leftrightarrow b}{\mathcal{L}_i}$ dans les automates $\mathcal{A}_i, \forall i \in 1, \ldots, n$. Soit $\mathcal{A} = \otimes \mathcal{A}_i$. Soit $\mathcal{L} = \bigcap_{1\leq i\leq n} \mathcal{L}_i$. Nous allons prouver que $\overset{a\leftrightarrow b}{\mathcal{L}}$ dans l'automate $\mathcal{A}$.

Soit $\beta \in \mathcal{L}$ ($\Rightarrow \beta \in \mathcal{L}_i$), $q = (q_1, \ldots, q_n) \in Q$. Alors :

– première condition :

$$\begin{array}{ll} \mathrm{en}^{\mathcal{A}}_{a;\beta;b}(q) & \Leftrightarrow \\ \mathrm{en}^{\mathcal{A}_i}_{a;\beta;b}(q_i), \forall i \text{ (Théorème 1)} & \Leftrightarrow \\ \mathrm{en}^{\mathcal{A}_i}_{b;\beta;a}(q_i), \forall i & \Leftrightarrow \\ \mathrm{en}^{\mathcal{A}}_{b;\beta;a}(q) & \end{array}$$

– deuxième condition :

$$\begin{array}{ll} \mathrm{en}^{\mathcal{A}}_{a;\beta;b}(q) & \Leftrightarrow \\ \mathrm{en}^{\mathcal{A}_i}_{a;\beta;b}(q_i), \forall i \text{ (Théorème 1)} & \Rightarrow \\ a;\beta;b(q_i) = b;\beta;a(q_i) = q'_i, \forall i & \Leftrightarrow \\ a;\beta;b(q) = b;\beta;a(q) = (q'_1, \cdots, q'_n) & \end{array}$$

Définition 8 *Deux séquences s et s' sont dites* inv-équivalentes *vis-à-vis d'un ensemble de relations d'interversibilité si on peut trouver une suite de séquences s_1, …, s_k telle que :*

– $\forall i \in \{1, \ldots, k-1\}$, $s_i = \alpha_i; a_i; \beta_i; b_i; \gamma_i$, $s_{i+1} = \alpha_i; b_i; \beta_i; a_i; \gamma_i$, *avec* $\overset{a_i\leftrightarrow b_i}{\{\beta_i\}}$,

– $s = s_1$,

– $s' = s_k$.

Deux séquences sont donc *inv-équivalentes* si on peut obtenir la deuxième à partir de la première par interversions successives d'événements respectant la relation d'interversibilité.

Corollaire 1 *Deux séquences inv-équivalentes sont de même longueur.*

5 Algorithme

Dans la première sous-section, nous voyons comment les propriétés d'interversibilité sont utilisées pour réduire le calcul dans la recherche de trajectoires. L'algorithme proposé est la base des algorithmes de diagnostic et de planification même s'il doit être légèrement adapté pour prendre en compte les observations dans un contexte de diagnostic ou prendre en compte les buts dans un contexte de planification. Cet algorithme suppose que les propriétés d'interversibilité ont déjà été calculées et peuvent être facilement vérifiées. La deuxième sous-section donne une illustration de cet algorithme. Dans la troisième sous-section, nous présentons un deuxième algorithme permettant d'établir ces propriétés, à savoir quels sont les événements interversibles et sur quels langages.

5.1 Algorithme de recherche

Algorithme 1 Dépliage de l'automate utilisant l'interversibilité

input 1: $etatInitial \in Q$
input 2: $etatsFinaux \in 2^Q$
$noeudSolution \leftarrow null$
$noeudRacine \leftarrow creerNoeudRacine(etatInitial)$
$noeuds\_non\_developpes \leftarrow \{noeudRacine\}$
tant que $noeuds\_non\_developpes \neq \varnothing \wedge noeudSolution = null$ **faire**
  $n' \leftarrow oterNoeud(noeuds\_non\_developpes)$
  pour tout $b \in E \mid (\mathrm{en}_b^{\mathcal{A}}(etat(n')) \ \wedge b \notin evites(n'))$ **faire**
    $developpes(n') \leftarrow \{b\} \cup developpes(n')$
    $n \leftarrow creerNoeud(n', b)$
    si $\neg cyclique(chemin(noeudRacine, n))$ **alors**
      si $etat(n) \notin etatsFinaux$ **alors**
        pour tout $a \in E$ **faire**
          si $\epsilon \in \mathcal{L}_n(a)$ **alors**
            $evites(n) \leftarrow evites(n) \cup \{a\}$
          fin si
        fin pour
        $noeuds\_non\_developpes \leftarrow noeuds\_non\_developpes \cup \{n\}$
      sinon
        $noeudSolution \leftarrow n'$
      fin si
    fin si
  fin pour
fin tant que

L'idée est d'utiliser la propriété d'interversibilité pour améliorer la recherche de trajectoires sur un espace d'états défini par un automate.

La propriété d'interversibilité permet d'obtenir une stratégie d'élagage : un chemin, qui est *inv-équivalent* à un chemin déjà développé à partir d'un état dans l'arbre de recherche, n'a pas besoin d'être développé à partir du même état. L'algorithme 1 étend l'algorithme classique de recherche en largeur.

Pour chaque nœud n créé (fonction $creerNoeud$), les structures de données suivantes lui sont associées :

1. $etat(n)$ est l'état du système représenté par n ;
2. $parent(n)$ est le nœud père de n ; soit n' ce nœud, ceci implique qu'il existe un événement e applicable en $etat(n')$ et reliant $etat(n')$ à $etat(n)$ (i.e $\mathrm{en}_e^{\mathcal{A}}(etat(n'))$ et $e(etat(n')) = etat(n)$) ; remarque : $parent(noeudRacine) = null$;
3. $developpes(n)$ est l'ensemble des événements applicables en $etat(n)$ et déjà développés dans l'arbre de recherche ;

4. $evites(n)$ est l'ensemble des événements applicables en $etat(n)$ et qui ne seront pas développés ;
5. la fonction $\mathcal{L}_n : E \rightarrow 2^{E^\star}$ associe chaque événement à un langage d'événements.

La fonction $\mathcal{L}_n$ est définie comme suit. Étant donné n un nœud de l'arbre de recherche, et a un événement, deux cas peuvent être distingués :

1. *n est le nœud racine de l'arbre* :
   $\mathcal{L}_n(a) = \emptyset$;
2. *n a un nœud père n'* : soit $b \in E$ l'événement qui mène de n' à n ($b(etat(n')) = etat(n)$), la séquence $\beta \in E^\star$ appartient à $\mathcal{L}_n(a)$ ssi une des conditions suivantes est respectée :
   – $(b;\beta) \in \mathcal{L}_{n'}(a)$
   – $(a \in developpes(n') \cup evites(n')) \wedge \overset{a \leftrightarrow b}{\{\beta\}}$

Un chemin β appartient à $\mathcal{L}_n(a)$ si le chemin $\beta; a$ n'a pas besoin d'être développé depuis le nœud n, parce qu'il est suffixe d'une trajectoire $\alpha;\beta;a$ dont une trajectoire *inv-équivalente* a déjà été développée.

5.2 Illustration

Prenons l'automate de la figure 8. Dans cet automate, il apparaît clairement qu'on a la propriété suivante : $\overset{a \leftrightarrow b}{\{c\}}$ qui signifie que les séquences suivantes sont équivalentes vis-à-vis de la propriété d'interversibilité : $a; c; b$ et $b; c; a$.

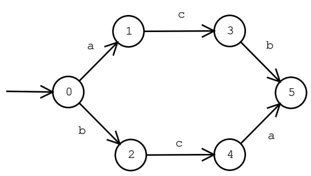

Fig.8. Exemple d'automate

L'état initial est l'état 0, et on cherche à atteindre l'état 5. Un développement en largeur conduit à l'arbre de recherche de la figure 9. La recherche par l'algorithme utilisant l'interversibilité (cf. 5.1) conduit à l'arbre présenté figure 10. Pour chaque nœud de l'arbre, sont indiqués l'état courant et les langages associés aux événements e ($\mathcal{L}_n(e)$) lorsqu'ils sont non vides.

Partant de l'état 0, deux événements peuvent se produire : l'événement a et l'événement b. L'événement a conduit à l'état 1. Les langages associés à cet état sont tous vides puisque :

– les langages associés au nœud père sont vides,
– aucun autre événement n'a été développé pour l'instant à partir du nœud père.

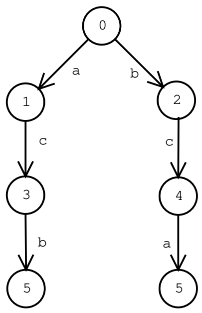

Fig. 9. Développement classique de l'automate

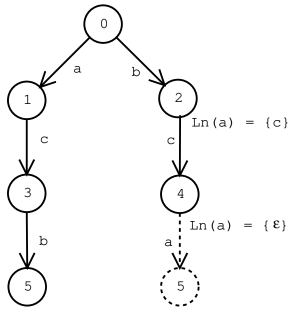

Fig. 10. Développement de l'automate selon l'algorithme 1

Les nœuds développés à partir de celui-ci réagissent de la même manière, nous les passons donc sous silence.

Depuis l'état 0, il est également possible de développement une trajectoire commençant par b. Cet événement permet d'atteindre le nœud associé à l'état 2. Le langage $\mathcal{L}_n(a)$ est alors égal à $\{c\}$ puisque a a déjà été développé depuis le nœud père, et que $\overset{a\leftrightarrow b}{\{c\}}$. Le langage $\mathcal{L}_n(a)$ a la signification suivante : si on développe l'événement a depuis un nœud n' atteint par un chemin β en partant de n (avec $\beta \in \mathcal{L}_n(a)$), alors la trajectoire permettant d'atteindre n' est *inv-équivalente* à une autre trajectoire de l'arbre qui va être développée. Ici, partant du nœud étiqueté par l'état 2, il est inutile de développer l'événement a après c puisqu'il nous permet d'atteindre l'état 5 par la trajectoire $b; c; a$, alors que cet état est atteint par la trajectoire *inv-équivalente* $a; c; b$.

À partir de ce nœud n', on peut développer l'événement c. Cela conduit dans l'état 4. Le langage $\mathcal{L}_n(a)$ pour ce nœud vaut alors uniquement l'ensemble des suffixes des mots de $\mathcal{L}_{n'}(a)$ commençant par c. Ici, cela correspond au mot ϵ (puisque $c; \epsilon \in \mathcal{L}_{n'}(a)$). Ainsi, depuis ce nœud, il ne faut pas développer l'événement a après ϵ, ce qui signifie qu'il ne faut pas développer a depuis ce nœud. Aussi, le développement de l'arbre s'arrête là.

5.3 Établir les propriétés d'interversibilité

Dans l'algorithme ci-dessus, il est supposé que les propriétés d'interversibilité sont déjà connues. Elles sont nécessaires pour calculer les fonctions $\mathcal{L}_n(e)$. Un problème est alors de calculer ces propriétés, c'est-à-dire pour toute paire a et b d'événements, le langage $\mathcal{L}^{a,b}$ tel que $\overset{a\leftrightarrow b}{\mathcal{L}^{a,b}}$. Dans la suite, un algorithme (algorithme 2) est proposé dans le cas où on se restreint à des langages de la forme $S^\star$, $S \subseteq E$ (on inclut le cas $\mathcal{L}^{a,b} = \varnothing$)[2].

L'idée de l'algorithme est la suivante. Étant donné un ensemble d'automates $\mathcal{A}$, pour chaque couple (a, b) d'événements, l'algorithme partitionne $\mathcal{A}$ en quatre

[2] $S^\star$ est l'ensemble des séquences construites à partir de l'ensemble d'événements S. Quand $S = \varnothing$, alors $S^\star = \{\varepsilon\}$.

Algorithme 2 Calcul des langages $\mathcal{L}^{a,b}$

pour tout $(a,b) \in E \times E, a \neq b$ **faire**
  let $I^{a,b} \subseteq \{1,\ldots,n\}$ tel que $i \in I^{a,b} \Leftrightarrow$
    $(a \in E_i) \wedge (b \in E_i)$
  si $\forall i \in I^{a,b}, \forall (q,q') \in (Q_i \times Q_i)$,
  $(q,a,q') \in T_i \Leftrightarrow (q,b,q') \in T_i$ **alors**
    soit $I^a \subseteq \{1,\ldots,n\}$ tel que $i \in I^a \Leftrightarrow$
      $(a \in E_i) \wedge (b \notin E_i)$
    soit $I^b \subseteq \{1,\ldots,n\}$ tel que $i \in I^b \Leftrightarrow$
      $(a \notin E_i) \wedge (b \in E_i)$
    soit $S = \{c \in E; \forall i \in I^a, c \notin E_i,$
      $\forall j \in I^b, c \notin E_j\}$
    nous avons : $\overset{a\leftrightarrow b}{S^\star}$
  sinon
    $\overset{a\leftrightarrow b}{\varnothing}$
  fin si
fin pour

sous-ensembles : $\mathcal{A}^{a,b}$, $\mathcal{A}^a$, $\mathcal{A}^b$, $\mathcal{A}^\varnothing$, où $\mathcal{A}^{a,b}$ est le sous-ensemble des automates de $\mathcal{A}$ dans lesquels a et b apparaissent tous les deux comme des événements, $\mathcal{A}^a$ est le sous-ensemble des automates de $\mathcal{A}$ dans lesquels a apparaît comme événement mais pas b, etc. La première étape est de vérifier si les événements a et b ont exactement le même rôle dans les automates appartenant à $\mathcal{A}^{a,b}$. Si ce n'est pas le cas, le langage vide est la seule solution et non avons $\mathcal{L}^{a,b} = \varnothing$. Sinon, soit S l'ensemble des événements qui n'apparaissent ni dans les automates de $\mathcal{A}^a$ ni dans ceux de $\mathcal{A}^b$. Il peut être montré, à l'aide du théorème 2 que $\overset{a\leftrightarrow b}{S^\star}$.

6 Résultats expérimentaux

Dans cette section, nous comparons l'algorithme 1 proposé dans la section 5 qui utilise la propriété d'interversibilité, et un algorithme traditionnel de recherche en largeur (qui n'explore pas les trajectoires bouclant) sur les deux exemples de la section 2. Les expérimentations ont été effectuées sur un processeur Intel 2.40GHz Pentium-4, 1GB RAM, sous le système d'exploitation Linux. Ces tests ont été faits en C++.

Système de diagnostic

Le test a été effectué sur un système comportant un TC et six composants. Étant donnée une séquence d'observations, i.e événements observables (*doAlarm_i*, *doReset*), le problème est de calculer la trajectoire de longueur minimale d'événements expliquant les observations. De plus, on considère que les composants fonctionnent de manière normale au début et à la fin des observations.

Par exemple, si nous considérons que le nombre maximum de composants en panne est 1 (le seuil du compteur a une valeur de 1), et les observations se limitent à $doAlarm\_1$, alors le diagnostic est le suivant : $(fault\_1; inc; alarm; doAlarm\_1; back\_1)$.

Les propriétés d'interversibilité sont calculées par l'algorithme 2. Par exemple, nous avons la propriété suivante sur deux événements de panne : $\underset{S^*}{fault\_i \leftrightarrow fault\_j}$, où $S = \{fault\_k, back\_k, doAlarm\_k, inc, doReset\}$ avec $k \neq i, k \neq j, i \neq j$.

		Algorithme en largeur		Algorithme 1		Algorithme 2
cpt	obs	temps	nœuds	temps	nœuds	temps
1	1	231ms	122	191ms	102	26ms
1	3	33mn 53s	6.472	3s 681ms	492	24ms
1	6	–	–	23s 822ms	1.027	24ms
2	2	1mn 40s	2.454	3s 244ms	531	26ms
2	4	–	–	38s 892ms	1.418	26ms
2	6	–	–	2mn 12s	2.361	24ms

Tab.1. Résultats pour le problème de diagnostic

La table 1 présente les résultats en terme de temps et de nombre de nœuds développés. Les cases marquées d'un – représente des expérimentations trop longues (supérieures à plusieurs heures). La première partie de la table correspond au cas où le maximum de composants en panne (le seuil du compteur, *cnt*) est mis à 1 ; le deuxième au cas où il est mis à 2. *obs* donne le nombre d'observations utilisées pour le diagnostic.

Système à planifier

Dans le problème *Bomb in the toilet*, il peut être montré que $Dunk\_i$ et $Dunk\_j$ sont interversibles sur n'importe quelle séquence d'événements. Nous avons : $\underset{S^*}{Dunk\_i \leftrightarrow Dunk\_j}$, avec $S = E - Dunk\_i - Dunk\_j$.

Le problème auquel nous nous intéressons est de trouver un plan optimal, c'est-à-dire le plan de longueur minimale. Cette longueur minimale est $2n + 1$ où n est le nombre de valises (*vls*). La complexité de ce problème est directement relative au nombre de valises. Alors que le nombre de nœuds développés par l'algorithme en largeur est en $o(n!)$, le nombre de nœuds développés par l'algorithme 1, utilisant la propriété d'interversibilité, est en $o(2^n)$, ce qui explique les résultats donnés dans la table 2.

	Algorithme en largeur		Algorithme 1		Algorithme 2
vls	temps	nœuds	temps	nœuds	temps
3	4ms	33	4ms	17	2ms
4	43ms	131	20ms	33	2ms
5	1s 37ms	653	73ms	65	2ms
6	2mn 24s	3.915	297ms	129	2ms
7	3h 17mn	27.401	1s 381ms	257	4ms
8	–	–	13s 133ms	513	4ms
9	–	–	1ms 50s	1.029	6ms

Tab.2. Résultats pour le problème de planification

7 Conclusion

Dans cet article, nous définissons une propriété, appelée propriété d'interversibilité, sur les événements d'un automate. Deux événements sont dits interversibles si deux séquences d'événements ne différents que dans l'ordre de ces deux événements (pas forcément successifs), conduisent toujours au même état du système. Cette propriété offre un moyen efficace pour représenter un ensemble de trajectoires (une trajectoire représente alors l'ensemble des trajectoires *inv-equivalentes*) et est exploitée pour restreindre l'ensemble des comportements à considérer en diagnostic et en planification. Deux algorithmes sont proposés : le premier permet de calculer de manière efficace les trajectoires. Le deuxième calcule automatiquement les propriétés d'interversibilité à partir de l'automate.

La propriété d'interversibilité peut être comparée avec la propriété d'indépendance entre événements utilisée dans la Réduction d'Ordre Partial [CGMP98,Pel93]. Il peut ainsi être prouvé que deux événements indépendants a et b sont interversibles sur le langage $\{\epsilon\}$.

Dans ce papier, nous nous intéressons uniquement aux automates déterministes. L'extension aux automates non déterministes n'est pas problématique puisqu'elle requiert de considérer des états de connaissance (ensemble des états courants possibles) au lieu d'états, et par conséquent de modifier la définition d'applicabilité des événements.

Références

[BCM+92] J.R. Burch, E.M. Clarke, K.L. Mc Millan, D.L. Dill, and L.J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BLPZ99] P. Baroni, G. Lamperti, P. Pogliano, and M. Zanella. Diagnosis of large active systems. *Artificial Intelligence*, 110:135–183, 1999.

[BMS00] C. Barral, S. McIlraith, and T.C. Son. Formulating diagnostic problem solving using an action language with narratives and sensing. In *KR'2000*, pages 311–322, 2000.

[CGMP98] E. M. Clarke, O. Grumberg, M. Minea, and D. Peled. State space reduction using partial order techniques. *International Journal on Software Tools for Technology Transfer*, 2:279–287, 1998.

[CL99] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic Publishers, 1999.

[CL01] M.-O. Cordier and C. Largouët. Using model-checking techniques for diagnosing discrete-event systems. In *DX'2001*, pages 39–46, 2001.

[CPR00] L. Console, C. Picardi, and M. Ribaudo. Diagnosis and diagnosability using PEPA. In *ECAI'2000*, pages 131–135, 2000.

[CR00] A. Cimatti and M. Roveri. Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research*, 13:305–338, July-December 2000.

[CT94] M.-O. Cordier and S. Thiébaux. Event-based diagnosis for evolutive systems. In *DX'1994*, pages 64–69, 1994.

[McD87] D. McDermott. A critique of pure reason. *Computational Intelligence*, 3(3):151–237, 1987.

[MR02] H. Marchand and L. Rozé. Diagnostic de pannes sur des systèmes à événements discrets : une approche à base de modèles symboliques. In *RFIA'2002*, pages 191–200, 2002.

[Pel93] D. Peled. All from one, one for all: on model checking using representatives. In *CAV'93*, pages 409–423, 1993.

[Pen02] Y. Pencolé. *Diagnostic décentralisé de systèmes à événements discrets : application aux réseaux de télécommunications.* PhD thesis, Université de Rennes 1, 2002.