EFFICIENT TRAJECTORIES COMPUTING EXPLOITING INVERSIBILITY PROPERTIES

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

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

Abstract: A time-consuming problem encountered both in system diagnosis and planning is that of computing trajectories over a behavioral model. In order to improve the efficiency of this task, there is currently a great interest in using model-checking techniques developed within the area of computer aided verification. In this paper, we propose to represent the system as automata and we define a property called inversibility. This property is used to improve the efficiency of the search algorithm computing trajectories. We present two study cases in diagnosis and planning domains where this approach gives satisfactory results.

1. INTRODUCTION

It is generally recognized that diagnosing dynamical systems, represented as discrete-event systems (DES) (Cassandras and Lafortune, 1999) amounts to finding what happened to the system from existing observations. Different terminologies can be found in the literature as histories (Baroni *et al.*, 1999), scenarios (Cordier and Thiébaux, 1994), narratives (Barral *et al.*, 2000), consistent paths (Console *et al.*, 2000). They all rely on the idea that the diagnostic task consists in determining the trajectories (a sequence of states and events) explaining the sequence of observations. In a similar way, planning consists in finding a sequence of actions, called a plan, that is guaranteed to achieve a goal. Both diagnostic and planning can thus be viewed as the problem of finding a path over a behavioral model. The main difficulty is the size of the model (number of states and transitions) and the explosive number of trajectories. It explains the current interest shown by the diagnosis and planning community in using model-checking techniques, originally designed for efficiently testing complex real-time systems (Clarke *et al.*, 1999). Having represented the planning domain or the system to be diagnosed as a finite state automaton, the problem of finding a trajectory is expressed as a reachability analysis on the model (Cordier and Largouët, 2001). To cope with the so-called state-explosion problem, techniques as Binary Decision Diagrams (BDD) (Burch *et al.*, 1992), Partial Order Reduction (POR) (Clarke *et al.*, 1998; Peled, 1993) have been proposed. They have been recently used in planning (use of BDD in (Cimatti and Roveri, 2000)) and in diagnosis (use of BDD in (Marchand and Rozé, 2002) and of POR in (Pencolé, 2002)).

In this paper, we propose to use a property of the model describing the DES to prune the search space without losing any information. This property is called inversibility and is defined on the events (or actions) of the system. Intuitively, two events are said to be inversible when, after transiting through any sequence of events including these two events, the state reached by the system is the same whenever the one or the other of the two events has been executed first.

The paper is structured as follows. We first briefly present the formalism of automata and define the notion of trajectory. We then propose two toy examples in diagnosis and planning that will be used as running examples throughout all the paper. The property of inversibility is then defined and the search algorithm is presented. Lastly, experimental results are analyzed and perspectives are discussed.

2. MODEL FORMALISM

In this section we first recall the formalism of automata we use to represent discrete-event systems before introducing the theoretical framework needed to define a trajectory. The formalism is illustrated by two examples in section 3.

Definition 1. (Automaton). An automaton is an tuple $\mathcal{A} = \langle Q, E, T, q_o \rangle$ where:

- Q is a finite set of state labels,
- E is a finite set of transition labels called events [1] ,
- $T \subseteq Q \times E \times Q$ is a finite set of transitions over the system. A transition t is a 3-uplet (q_1, e, q_2) such that t links $q_1 \in Q$ to $q_2 \in Q$ on an edge labeled by $e \in E$,
- q_o is the initial state.

The automaton represents the set of states of the system and describes the evolution of the current state w.r.t the events occurring on the system. In this article, we only consider the case of deterministic automata.

A system being a set of interconnected components, it is usually easier to describe the behavior of each elementary component. The global model is obtained by a composition operation with synchronization over the events.

Definition 2. (Synchronized product). The synchronized product of n automata $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{o,i} \rangle$, noted $\otimes_{i=1,n}\mathcal{A}_i$, is an automaton $\mathcal{A} = \langle Q, E, T, q_o \rangle$ such that:

- $Q = Q_1 \times \cdots \times Q_n$,
- $E = \bigcup_{1 \leq i \leq n} E_i$,
- $T = \left\{ \begin{array}{c} \{(q_1, \cdots, q_n), e, (q'_1, \cdots, q'_n)\} \mid \\ \forall i((e \in E_i \Rightarrow (q_i, e, q'_i) \in T_i) \wedge \\ (e \notin E_i \Rightarrow q_i = q'_i)) \end{array} \right\}$,
- $q_o = (q_{o,1}, \cdots, q_{o,n})$.

The properties associated to events and sequences of events are the following.

Definition 3. (Enabled event). An event e is enabled in a state $q \in Q$ of an automaton $\mathcal{A} = \langle Q, E, T, q_o \rangle$ if there exists a transition from q labeled with e, i.e :

$$\mathrm{en}_e^{\mathcal{A}}(q) = \begin{cases} true & \text{if } (\exists q' \mid (q, e, q') \in T) \vee (e \notin E)) \\ false & \text{elsewhere} \end{cases}$$

The following theorem can be easily proved.

Theorem 1. In an automaton $\mathcal{A} = \langle Q, E, T, q_o \rangle = \otimes_{i=1,n}\mathcal{A}_i$, with $\mathcal{A}_i = \langle Q_i, E_i, T_i, q_{o,i} \rangle$, an event e is enabled in a state $q = (q_1, \cdots, q_n)$ iff e is enabled in the state q_i of $\mathcal{A}_i$ for all i, i.e: $\mathrm{en}_e^{\mathcal{A}}(q) = \mathrm{en}_e^{\mathcal{A}_1}(q_1) \wedge \cdots \wedge \mathrm{en}_e^{\mathcal{A}_n}(q_n)$

We denote $e(q)$ with $e \in E$ and $q \in Q$ the state q' reached from the state q by the transition labeled by e such that $(q, e, q') \in T$.

A *sequence* over a set of events E is a sequence $\gamma_1;\gamma_2;\ldots;\gamma_n$ where each γ_i is either an event in E or a sequence of events. In the following, the sequences of events are denoted by Greek letters α, β, ... and ε denotes the empty sequence.

Definition 4. (Enabled sequence of events). A sequence of events $\alpha = e_1; e_2; \cdots; e_k = e_1; \beta$ is enabled in a state q of the automaton $\mathcal{A}$, noted $\mathrm{en}_\alpha^{\mathcal{A}}(q)$, iff $\mathrm{en}_{e_1}^{\mathcal{A}}(q) \wedge \mathrm{en}_\beta^{\mathcal{A}}(e_1(q))$.

An empty sequence is enabled in any state and we have thus $\forall q \in Q\ \mathrm{en}_\epsilon^{\mathcal{A}}(q) = \text{true}$.

Definition 5. (Trajectory). A sequence of events is called a *trajectory* [2] of $\mathcal{A}$ iff the sequence is enabled in the initial state q_o of the automaton.

3. EXAMPLES

We present now two toy examples, in diagnosis and planning domains respectively, which have been used for the experimentations presented in section 6.

Diagnosis system

Consider a simplified telecommunication system composed of a set of components $Comp_i$ and a technical center TC which receives messages from the components (see figure 1). A component can be in normal behavior or in abnormal behavior due to a failure. A component is composed of two parts: the component itself called the unit and a controller which detects abnormal behaviors of the unit. When a fault occurs, the controller sends a $fault\_i$ message to the TC and turns the unit to the abnormal state. The TC is also composed of two parts: a monitor and a counter.When the

[1] In applicative domains, transitions labels are associated to events in diagnosis and to actions in planning. In the following of this paper, we call a transition label by the generic term *event*.

[2] The term trajectory commonly used in diagnosis corresponds to a *plan* in planning

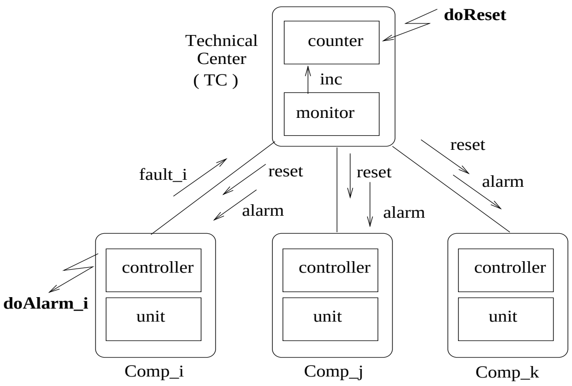

Fig. 1. A simplified telecommunication system

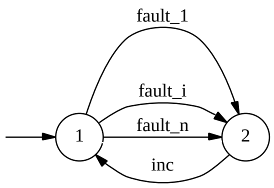

Fig. 2. Model of the TC monitor

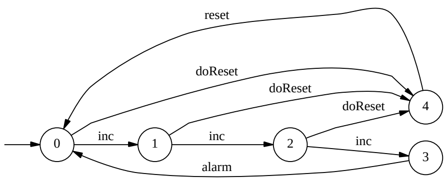

Fig. 3. Model of the TC counter

monitor receives a $fault$ event from a component controller, it asks the counter to increment its value. An external clock governs the resetting of the counter value. When this exogenous $doReset$ event is received, the technical center TC asks, by sending a $reset$ message, all the components to evolve from the abnormal state to the normal state. Between two resets, if the counter value reaches a predefined threshold, considered as the maximum number of faulty components supported by the system, the TC sends the $alarm$ event to all components. When a component controller, the unit of which is faulty, receives an $alarm$ event, it emits an exogenous $doAlarm\_i$ event to a supervisor. The diagnosis task consists in providing the sequences of events that occurred on the system and explain the alarms observed by the supervisor ($doAlarm\_i$, $doReset$).

Let us give now the model of this system described by the model of each of its components. The TC monitor (see figure 2) is defined by two states. The system moves from state 1 to state 2 when it receives a fault event from one of the component. After having sent a inc event to the counter, it returns to its initial state. To represent the behavior of the counter (see figure 3), the maximum number of faulty components has been set to 3. The states 1, 2 and 3 correspond to the value of the counter. When the counter receives the exogenous $doReset$ alarm, it moves to state 4, sends to all the controllers a $reset$ event and returns to the initial state.

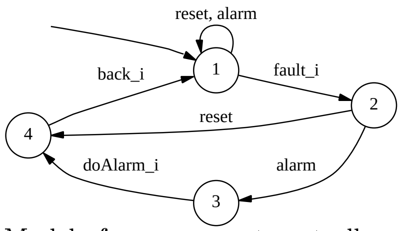

Fig. 4. Model of a component controller

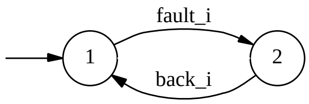

Fig. 5. Model of a component unit

The component controller (see figure 4) receives the $fault\_i$ event from the unit and moves to state 2. In this state, two kinds of events can be received: $reset$ or $alarm$. The $reset$ event lead the controller to state 4. In case the $alarm$ event is received, a $doAlarm\_i$ event is sent and the controller comes back to state 4. The component unit (figure 5) can be in a normal or abnormal state (if a fault has occurred). The $back\_i$ event, received from the controller, makes the unit coming back to a normal state.

Planning system

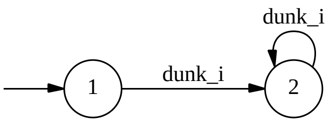

Fig. 6. Model of a package

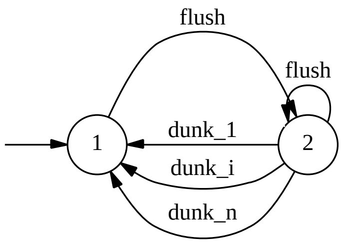

Fig. 7. Model of the toilet

The planning example is a variation of Moore's bomb in the toilet domain (McDermott, 1987). We initially suppose that there are n packages in a bathroom, that all contain an armed bomb and that the toilet is clogged. The goal is to get all the bombs disarmed and the toilet unclogged. The only way to disarm a bomb is to dunk a package containing a bomb in the toilet (*dunk_i* action), provided that the toilet is not clogged. Dunking the package has the effect of clogging the toilet. The toilet can be clogged by the $flush$ action.

The behavior of the system is represented by two automata. The first one (see figure 6) is associated to the package i. The second one (see figure 7) describes the toilet. In the package automata, state 1 syas that the package contains the armed bomb and state 2 that it is disarmed. The *dunk_i* action defuses the bomb if any. In figure 7, in state 1, the toilet is clogged, while it is unclogged in state 2. Dunking a package has the effect of clogging the toilet which is represented by the *dunk_i* actions. The $flush$ transition indicates that performing the $flush$ action unclogs the toilet.

4. INVERSIBILITY

In this section, an inversibility property is defined on events. In the next section, an algorithm exploiting this property for improving trajectories computing is presented. Intuitively, this property indicates that two events, under some conditions, can be inverted in a sequence without any consequence on the final state reached by the system.

Let $\mathcal{L}$ be a language composed of sequences of events.

Definition 6. Two events a and b ($a \neq b$) satisfy the inversibility property with respect to a language $\mathcal{L}$ in an automaton $\mathcal{A} = \langle Q, E, T, q_o \rangle$ (noted $\overset{a\clubsuit b}{\mathcal{L}}$) iff $\forall q \in Q, \forall \beta \in \mathcal{L}$,

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

This definition means that if $\overset{a\clubsuit b}{\mathcal{L}}$, then for all sequences $\beta \in \mathcal{L}$, the sequence of events $a;\beta;b$ has the same effect on the system that the sequence $b;\beta;a$.

This property is compositional, as shown by the following theorem.

Theorem 2. If two events a and b can be inverted w.r.t the languages $\mathcal{L}_i$ in the automata $\mathcal{A}_i$ ($\overset{a\clubsuit b}{\mathcal{L}_i}$ in $\mathcal{A}_i, \forall i \in \{1, \ldots, n\}$), then these two events can be inverted over the language $\mathcal{L} = \bigcap_{1 \leq i \leq n} \mathcal{L}_i$ in the automaton $\mathcal{A} = \otimes \mathcal{A}_i$.

5. ALGORITHM

In the first subsection, it is shown how the inversibility properties are used to prune the search when looking for trajectories. This algorithm is the heart of a diagnosis or planning algorithm as soon as it is viewed as a path search algorithm. It has to be slightly adapted to take into account observations in a diagnosis context or the goals in a planning context. Domain dependant heuristics can clearly be used to improve the search efficiency. This algorithm supposes that the inversibility properties have already been collected and can then be easily checked. In the second subsection, we consider the problem of establishing this collection of properties, i.e which are the events inversible and with respect to which language.

5.1 Search algorithm

Algorithm 1 Unfolding of the automaton using inversibility

input: $finalStates \in 2^Q$
$solutionNode \leftarrow null$
$rootNode \leftarrow makeRootNode(initialState)$
$node\_not\_developed \leftarrow \{rootNode\}$
while $node\_not\_developed \neq \varnothing \wedge solutionNode = null$ **do**
  $n' \leftarrow removeNode(node\_not\_developed)$
  for all $b \in E \mid (en^{\mathcal{A}}_b(state(n')) \wedge b \notin events\_pruned(n'))$ **do**
    $\mathrm{events\_developed}(n') \leftarrow b \cup \mathrm{events\_developed}(n')$
    $n \leftarrow makeNode(n', b)$
    if $\neg cyclic(path(rootNode, n))$ **then**
      if $state(n) \notin finalStates$ **then**
        for all $a \in E$ **do**
          if $\epsilon \in \mathcal{L}_n(a)$ **then**
            $events\_pruned(n) \leftarrow events\_pruned(n) \cup \{a\}$
          end if
        end for
        $node\_not\_developed \leftarrow node\_not\_developed \cup \{n\}$
      else
        $solutionNode \leftarrow n'$
      end if
    end if
  end for
end while
if $solutionNode \neq null$ **then**
  return $path(rootNode, solutionNode)$
else
  return null
end if

The idea is to use the inversibility property to improve the search of a path (sequence of events) over the state space defined by the model automata. Two sequences $s1$ and $s2$ are said to be *inv-equivalent* if $s2$ can be obtained from $s1$ by inverting the events according to the inversibility properties.

The inversibility property induces a pruning strategy: a path which is *inv-equivalent* to a path already developed in the search tree does not need to be developed. The algorithm (see 1) extends the classic breadth-first search algorithm.

Each time a node n is created (by $makeNode$), the following data structures are associated to it:

(1) $state(n)$ is the state represented by n;
(2) $parent(n)$ is the parent node of n; let n' be this node; it implies that there exists an event e enabled in $state(n')$ and linking $state(n')$ to $state(n)$ (i.e $\mathrm{en}_e^{\mathcal{A}}(state(n'))$ and $e(state(n')) = state(n)$).
(3) $events\_developed(n)$ is the set of events, enabled in $state(n)$, and already developed in the search tree;
(4) $events\_pruned(n)$ is the set of events, enabled in $state(n)$, which do not need to be developed;
(5) a function $\mathcal{L}_n : E \to 2^{E^\star}$ which maps each event to a language of events.

The function $\mathcal{L}_n$ is defined as follows. Given n a node of the search tree and a an event, two cases are distinguished:

(1) *n is the root node of the search tree*: $\mathcal{L}_n(a) = \emptyset$;
(2) *n has a parent node n'*: let $b \in E$ be the event such that $b(state(n')) = state(n)$, the sequence $\beta \in E^\star$ belongs to $\mathcal{L}_n(a)$ iff one of the following assertions is true:
   - $(b;\beta) \in \mathcal{L}_{n'}(a)$
   - $a \in events\_developed(n') \cup events\_pruned(n') \wedge \overset{a\clubsuit b}{\{\beta\}}$

A sequence β belongs to $\mathcal{L}_n(a)$ if the path $\beta; a$ does not need to be developed, because an *inv-equivalent* path has already been developed.

5.2 Establishing the inversibility properties

In the above algorithm, it is supposed that the inversibility properties are already known. They are for instance needed to compute the functions $\mathcal{L}_n(e)$. A problem is thus to compute, for any pair of events a and b, the language $\mathcal{L}^{a,b}$ such that $\overset{a\clubsuit b}{\mathcal{L}^{a,b}}$. In the following, an algorithm (algorithm 2) is proposed in the restricted case where we impose that the language $\mathcal{L}^{a,b}$ is in the form $S^{\star}$ [3], $S \subseteq E$ (it includes the case $\mathcal{L}^{a,b} = \varnothing$).

The idea of the algorithm is the following. Given a set of automata $\mathcal{A}$, for each couple (a, b) of events, the algorithm computes a partition of $\mathcal{A}$ into 4 sets: $\mathcal{A}^{a,b}$, $\mathcal{A}^a$, $\mathcal{A}^b$, $\mathcal{A}^\varnothing$, where $\mathcal{A}^{a,b}$ is the subset of $\mathcal{A}$ in which a and b both appear as events, $\mathcal{A}^a$ is the subset of $\mathcal{A}$ in which a appears as event but not b, and so on. The first step checks whether the events a and b have exactly the same role in the automata belonging to $\mathcal{A}^{a,b}$. If it is not the case, the empty language is the only solution and we have $\mathcal{L}^{a,b} = \varnothing$. Else, let S be the set of events that do not appear in $\mathcal{A}^a$ nor in $\mathcal{A}^b$. It can be shown, using theorem 2, that $\overset{a\clubsuit b}{S^\star}$. For each automaton, the complexity is $o(|E|^3 \times |Q|)$.

Algorithm 2 Computation of the languages $\mathcal{L}^{a,b}$

for all $(a, b) \in E \times E, a \neq b$ **do**
  let $I^{a,b} \subseteq \{1, \ldots, n\}$ so that $i \in I^{a,b} \Leftrightarrow (a \in E_i) \wedge (b \in E_i)$
  if $\forall i \in I^{a,b}, \forall (q, q') \in (Q_i \times Q_i), (q, a, q') \in T_i \Leftrightarrow (q, b, q') \in T_i$ **then**
    let $I^a \subseteq \{1, \ldots, n\}$ so that $i \in I^a \Leftrightarrow (a \in E_i) \wedge (b \notin E_i)$
    let $I^b \subseteq \{1, \ldots, n\}$ so that $i \in I^b \Leftrightarrow (a \notin E_i) \wedge (b \in E_i)$
    let $S = \{c \in E; \forall i \in I^a, c \notin E_i, \forall j \in I^b, c \notin E_j\}$
    we have: $\overset{a\clubsuit b}{S^\star}$
  else
    $\overset{a\clubsuit b}{\varnothing}$
  end if
end for

6. EXPERIMENTAL RESULTS

In this section, we compare the algorithm 1 proposed in section 5 which uses the inversibility property and a traditional breadth-first algorithm (that do not explore looping trajectories) on the two examples of section 3. The experimentation was performed on an Intel 2.40GHz Pentium-4, 1GB RAM, running Linux.

Diagnosis system

The test has been performed on a system composed of one TC and six components. Given a sequence of observations, i.e observable events ($doAlarm\_i$, $doreset$), the problem is to compute the minimal sequences of events explaining the observations. Moreover, we suppose that all the components are normally running at the beginning and at the end of the observations.

For example, if we consider that the maximum number of acceptable faults is 1 (the counter threshold equals 1), and the observation is $doAlarm\_1$, then the diagnosis is the following : $(fault\_1; inc; alarm; doAlarm\_1; back\_1)$.

The inversibility properties are computed by algorithm 2. For instance, we get the following property on two fault events : $\overset{fault\_i\clubsuit fault\_j}{S^*}$, where

$S = \{fault\_k, back\_k, doAlarm\_k, inc, doReset\}$ with $k \neq i, k \neq j$.

Table 1 presents the results in terms of time and number of developed nodes. The first part of the table corresponds to the case where the maximum number of acceptable faulty components, i.e the counter threshold (cnt) is set to one; the second one to the case where it is set to 2. obs gives the number of observations considered for the diagnosis.

[3] $S^\star$ is the set of all the sequences built from elements belonging to S. When $S = \varnothing$ then $S^\star = \varepsilon$.

		Breadth-first algorithm		Algorithm 1	
cnt	obs	time	nodes	time	nodes
1	1	553 ms	82	429 ms	67
1	3	2 h 8 mn	5077	22 s 121 ms	427
1	6	–	–	2 mn 52 ms	967
2	2	2 mn 49 s	1207	5 s 732 ms	372
2	4	–	–	1 mn 29 s	1265
2	6	–	–	5 mn 22 s	2190

Table 1. Results for the diagnosis problem

Planning system

In the *Bomb in the Toilet* problem, it can be shown that any *dunk_i* and *dunk_j* are inversible w.r.t any sequence of events. We have : $Dunk\_i \underset{S^*}{\clubsuit} Dunk\_j$, with $S = E - Dunk\_i - Dunk\_j$.

	Breadth-first algorithm		Algorithm 1	
pkg	time	nodes	time	nodes
3	6 ms	28	6 ms	17
4	82 ms	108	24 ms	33
5	2 s 787 ms	534	102 ms	65
6	6 mn 41 s	3196	440 ms	129
7	6 h 21 mn	22362	2 s 157 ms	257
8	–	–	28 s 131 ms	513

Table 2. Results for the planning problem

The problem we considered is to find the optimal plan, i.e the plan with minimal length. This minimal length is $2n + 1$ where n is the number of packages. The complexity of the problem is directly related to the number of packages. While the number of nodes developed by the breadth-first algorithm is $o(n!)$, the number of nodes developed by the algorithm 1, using the inversibility property, is $o(2^n)$, which explains the results given by table 2.

7. CONCLUSION

In this paper, we define a property, called inversibility property, on events in automata. Two events are said to be inversible when, after transiting through any sequence of events including these two events, the state reached by the system is the same whenever the one or the other of the two events has been executed first. This property provides an efficient way of representing sets of trajectories (a trajectory represents the set of all its inv-equivalent trajectories) and is exploited to restrict the set of behaviors to consider in diagnosis or planning systems. Two algorithms are proposed : the first one to efficiently computing trajectories. The second one to automatically computing the inversibility properties from the automata.

The inversibility property is related to the independence property between events used in Partial Ordered Reduction (Clarke *et al.*, 1998; Peled, 1993). It can be shown that two independent events a and b are two events which are inversible w.r.t to ϵ. Using the inversibility property is clearly relevant to the kind of applications where Producer/Consumer relations exist between the components.

In this paper, we restrict ourselves to deterministic automata. The extension to nondeterministic automata is unproblematic but requires to consider belief states instead of states and to consequently modify the definition of *enabled events*.

REFERENCES

Baroni, P., G. Lamperti, P. Pogliano and M. Zanella (1999). Diagnosis of large active systems. *Artificial Intelligence* **110**, 135–183.

Barral, C., S. McIlraith and T.C. Son (2000). Formulating diagnostic problem solving using an action language with narratives and sensing. In: *KR'2000*. pp. 311–322.

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

Cassandras, C. G. and S. Lafortune (1999). *Introduction to Discrete Event Systems*. Kluwer Academic Publishers.

Cimatti, A. and M. Roveri (2000). Conformant planning via symbolic model checking. *Journal of Artificial Intelligence Research* **13**, 305–338.

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

Clarke, E., O. Grumberg and D. Peled (1999). *Model checking*. MIT Press.

Console, L., C. Picardi and M. Ribaudo (2000). Diagnosis and diagnosability using PEPA. In: *ECAI'2000*. pp. 131–135.

Cordier, M.-O. and C. Largouët (2001). Using model-checking techniques for diagnosing discrete-event systems. In: *DX'2001*. pp. 39–46.

Cordier, M.-O. and S. Thiébaux (1994). Event-based diagnosis for evolutive systems. In: *DX'1994*. pp. 64–69.

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

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

Peled, D. (1993). All from one, one for all: on model checking using representatives. In: *CAV'93*. pp. 409–423.

Pencolé, Y. (2002). Diagnostic décentralisé de systèmes à événements discrets : application aux réseaux de télécommunications. PhD thesis. Université de Rennes 1.