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.Pencol@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 sys-
tems (DES) (Cassandras and Lafortune, 1999) amounts to finding what happened to the sys-
tem from existing observations. Different terminologies can be found in the literature as histo-
ries (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 con-
sists 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 de-
signed for efficiently testing complex real-time sys-
tems (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 plan-
ing (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 prop-
erty is called inversibility and is defined on the
events (or actions) of the system. Intuitively, two
events are said to be irreversible when, after tran-
siting 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 irreversibility 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 $A = (Q, E, T, q_0)$ where:

- $Q$ is a finite set of state labels,
- $E$ is a finite set of transition labels called events \footnote{In applicative domains, transition 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.},
- $T \subseteq Q \times E \times Q$ 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 $e \in E$,
- $q_0$ 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 $A_i = (Q_i, E_i, T_i, q_{0,i})$, noted $\otimes_{i=1}^n A_i$, is an automaton $A = (Q, E, T, q_0)$ such that:

- $Q = Q_1 \times \cdots \times Q_n$,
- $E = \bigcup_{1 \leq i \leq n} E_i$,
- $T = \left\{ (q, e, (q_1, \cdots, q_n)) \mid \forall i \in \{1, \cdots, n\}, (q_i, e, q_i) \in T_i \wedge (e \notin E_i \Rightarrow q_i = q_i) \right\}$,
- $q_0 = (q_{0,1}, \cdots, q_{0,n})$.

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 $A = (Q, E, T, q_0)$ if there exists a transition from $q$ labeled with $e$, i.e.:

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

The following theorem can be easily proved.

Theorem 1. In an automaton $A = (Q, E, T, q_0)$, an event $e$ is enabled in a state $q = (q_1, \cdots, q_n)$ iff $e$ is enabled in the state $q_i$ of $A_i$ for all $i$, i.e.:

$$e^A(q) = e^A_1(q_1) \wedge \cdots \wedge e^A_n(q_n)$$

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 $\gamma_1; \gamma_2; \cdots ; \gamma_n$ where each $\gamma_i$ is either an event in $E$ or a sequence of events. In the following, the sequences of events are denoted by Greek letters $\alpha, \beta, \ldots$ and $e$ denotes the empty sequence.

Definition 4. (Enabled sequence of events). A sequence of events $\alpha = e_1; e_2; \cdots ; e_k$ is enabled in a state $q$ of the automaton $A$, noted $e^A_\alpha(q)$, iff $e^A_1(q) \land e^A_\beta(q)$.

An empty sequence is enabled in any state and we have thus $\forall q \in Q \ e^A_\emptyset(q) = \text{true}$.

Definition 5. (Trajectory). A sequence of events is called a trajectory \footnote{The term trajectory commonly used in diagnosis corresponds to a plan in planning} of $A$ iff the sequence is enabled in the initial state $q_0$ of the automaton.

3. EXAMPLES

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

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](image)

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 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 says that the package contains the armed bomb and state 2 that it is disarmed. The dunk 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 actions. The flush transition indicates that performing the flush action unclogs the toilet.

4. INVERSIBILITY

In this section, an invariance 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 invariance property with respect to a language \( \mathcal{L} \) in an automaton \( \mathcal{A} = (Q, E, T, q_0) \) (noted \( a \in \mathcal{L} \)) iff \( \forall q \in Q, \forall \beta \in \mathcal{L}, \)
- \( \epsilon_n^A;\beta;\alpha(q) \Leftrightarrow \epsilon_n^A;\beta;\alpha(q) \), and
- \( \epsilon_n^A;\beta;\alpha(q) \Rightarrow a; \beta; b(q) = b; \beta; a(q) \).

This definition means that if \( a \in \mathcal{L} \), 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 \( \mathcal{A}_i \) (\( \mathcal{L} \) 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 invariance 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 invariance 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 irreversible and with respect to which language.

5.1 Search algorithm

**Algorithm 1 Unfolding of the automaton using invariance**

| input: finalStates \( \in 2^Q \) | solutionNode \( \leftarrow \) null |
| rootNode \( \leftarrow \) makeRootNode(initialState) |
| node_not_developed \( \leftarrow \) {rootNode} |
| while node_not_developed \( \neq \emptyset \) \& solutionNode \( = \) null do |
| \( n' \leftarrow \text{removeNode}(\text{node_not_developed}) \) |
| for all \( b \in E \) | \( (\epsilon_n^A (\text{state}(n')) \land b \notin \text{events_pruned}(n')) \) do |
| \( \text{events_developed}(n') \leftarrow b \cup \text{events_developed}(n') \) |
| \( n \leftarrow \text{makeNode}(n', b) \) |
| if \( \neg \text{cyclic(path(rootNode, n))} \) then |
| if \( \text{state}(n) \notin \text{finalStates} \) then |
| for all \( a \in E \) do |
| if \( \epsilon_n^A \in \mathcal{L}_a(a) \) then |
| \( \text{events_pruned}(n) \leftarrow \text{events_pruned}(n) \cup \{a\} \) |
| end if |
| end for |
| node_not_developed \( \leftarrow \text{node_not_developed} \cup \{n\} \) |
| else |
| solutionNode \( \leftarrow n' \) |
| end if |
| end if |
| end while |
| if solutionNode \( \neq \) null then |
| return path(rootNode, solutionNode) |
| else |
| return null |
| end if |

The idea is to use the invariance 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 in-equivalent if \( s2 \) can be obtained from \( s1 \) by inverting the events according to the invariance properties.

The invariance property induces a pruning strategy: a path which is in-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 \( \text{makeNode} \)), the following data structures are associated to it:

1. \( \text{state}(n) \) is the state represented by \( n \);
2. \( \text{parent}(n) \) is the parent node of \( n \); let \( n' \) be this node; it implies that there exists an event \( e \) enabled in \( \text{state}(n') \) and linking \( \text{state}(n') \) to \( \text{state}(n) \) (i.e., \( \text{en}_e(\text{state}(n')) \) and \( e(\text{state}(n')) = \text{state}(n) \).
3. \( \text{events\_developed}(n) \) is the set of events, enabled in \( \text{state}(n) \), and already developed in the search tree;
4. \( \text{events\_pruned}(n) \) is the set of events, enabled in \( \text{state}(n) \), which do not need to be developed;
5. a function \( \mathcal{L}_n : E \to 2^{\mathcal{E}} \) 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 \( b(\text{state}(n')) = \text{state}(n) \), the sequence \( \beta \in E^* \) belongs to \( \mathcal{L}_n(a) \) iff one of the following assertions is true:
   - \( \langle a, \beta \rangle \in \mathcal{L}_n(a) \)
   - \( a \in \text{events\_developed}(n') \cup \text{events\_pruned}(n') \land \langle a, \beta \rangle \)

A sequence \( \beta \) belongs to \( \mathcal{L}_n(a) \) if the path \( \beta \) \( a \) does not need to be developed, because an inequivalent path has already been developed.

5.2 Establishing the invariance properties

In the above algorithm, it is supposed that the invariance 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 \( \mathcal{L}^{a,b} \). 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 \( S^* \), \( S \subseteq E \) (it includes the case \( \mathcal{L}^{a,b} = \emptyset \)).

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: \( \mathcal{A}^{a,b} \), \( \mathcal{A}^a \), \( \mathcal{A}^b \), \( \mathcal{A}^{a,b} \), 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 \( \mathcal{L}^{a,b} = \emptyset \). 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 \( \mathcal{S}^* \). For each automaton, the complexity is \( o(|E|^3 \times |Q|) \).

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

\begin{algorithm}
\begin{algorithmic}
\ForAll{\( (a, b) \in E \times E, a \neq b \)}
\State let \( t^{a,b} \subseteq \{1, \ldots, n\} \) such that \( i \in t^{a,b} \iff\)
\State \( \langle a \in E_i \rangle \land (b \notin E_i) \)
\State \langle \forall i \in t^{a,b}, \forall (q, q') \in (Q_i \times Q_i), (q, a, q') \in T_i \Rightarrow \langle q, q' \rangle \in T_i \rangle \)
\State let \( t^a \subseteq \{1, \ldots, n\} \) such that \( i \in t^a \iff\)
\State \( \langle a \notin E_i \rangle \land (b \in E_i) \)
\State let \( t^b \subseteq \{1, \ldots, n\} \) such that \( i \in t^b \iff\)
\State \( \langle a \notin E_i \rangle \land (b \in E_i) \)
\State \langle \forall j \in t^b, c \notin E_i \rangle \)
\State \langle \forall j \in t^b, c \notin E_i \rangle \)
\State we have: \( \mathcal{S}^{a,b} \)
\EndIf
\EndFor
\end{algorithmic}
\end{algorithm}

6. EXPERIMENTAL RESULTS

In this section, we compare the algorithm 1 proposed in section 5 which uses the invariance 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 2.4GHz 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,j}, doneRset), 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_{i,1}, then the diagnosis is the following: \( \text{fault}_{i,j}; inc; alarm; doAlarm_{j,1}; back_{j,1} \).

The invariance properties are computed by algorithm 2. For instance, we get the following property on two fault events: \( \text{fault}_{i,j}; \text{fault}_{i,j}, \text{doAlarm}_{j,1}, \) where

\[ S = \{ \text{fault}_{k,1}, \text{back}_{k,1}, \text{doAlarm}_{j,1}, \text{inc}, \text{doReset} \} \]

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.

\footnote{\( S^* \) is the set of all the sequences built from elements belonging to \( S \). When \( S = \emptyset \) then \( S^* = e. \)}
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: \[ S' = D_{i} \cap D_{j} \]

<table>
<thead>
<tr>
<th>pkg</th>
<th>time</th>
<th>nodes</th>
<th>time</th>
<th>nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5 ms</td>
<td>20</td>
<td>0.5 ms</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1.5 ms</td>
<td>40</td>
<td>1.5 ms</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 1. 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 inequivalent 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 \( \epsilon \). 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


