Using model-checking techniques for diagnosing discrete-event systems

Marie-Odile Cordier
IRISA, 35042 Rennes, FRANCE
cordier@irisa.fr
tel: (33)299847100
fax: (33)299847171

Christine Largouët
ENSAR+IRISA, 35042 Rennes, FRANCE
clargoue@irisa.fr
tel: (33)299847100
fax: (33)299847171

Abstract

This paper deals with diagnosing dynamical systems represented by a discrete-event model and more precisely represented in an automata formalism. It shows how model-checking techniques which have been designed for efficiently testing complex real-time systems can be exploited for diagnostic task. This work originates from an application in the monitoring of agricultural plots which has been implemented by using these model-checking techniques.

Introduction

The model-based approaches addressed first the limited range of static systems. They are now being extended to dynamical systems and an increasing amount of work, in both AI and Control engineering, is devoted to the class of discrete-event systems (Cassandras & Lafortune 1999). It concerns continuous-variable systems which, after quantization, are represented as discrete systems (Sampath *et al.* 1994; 1996; Lunze 1999) as well as "discrete by nature" systems such as communicating processes which exchange messages and alarms (Rozé & Cordier 1998; Baroni *et al.* 1999; Pencolé 2000). Even if, in some restricted cases, state-based approaches have been proved to be possible, it is generally recognized that diagnosing dynamical systems amounts to finding what has happened to the system from existing observations and models. Different terminologies can be found in the literature as history reconstruction (Baroni *et al.* 1999), scenarios (Cordier & Thiébaux 1994), situation histories or narratives (Barral, McIlraith, & Son 2000), consistent paths (Console, Picardi, & Ribaudo 2000); they all rely on the idea that the diagnostic task consists in tracking the system and determining the most probable trajectories (a sequence of states and transitions) explaining the sequence of observations. The diagnoses are then extracted from these trajectories according to what is precisely looked for.

This paper deals with diagnosing dynamical systems represented by a discrete-event model and more precisely represented in an automata formalism. It shows how model-checking techniques which have been designed for efficiently testing complex real-time systems can be exploited for this diagnostic task. It presents an application in the monitoring of agricultural plots which has been implemented by using these model-checking techniques (Largouët & Cordier 2000a) and discusses the benefits of this method.

We begin by presenting an illustrative example inspired by (Console, Picardi, & Ribaudo 2000). A theoretical framework is then provided in order to precisely define what we call trajectories and diagnoses according to this discrete-event approach. After an introduction to model-checking techniques, we show how the diagnostic task can be solved by reachability analysis and we illustrate it in the Kronos formalism. Finally, we briefly relate the experimentation that has motivated the use of this kind of methods and which concerns the monitoring of agricultural plots from a sequence of images. We conclude by comparing our approach to related work.

An illustrative example

In this section, we present an example that we shall use to illustrate the definitions and methods presented throughout the paper. This example was introduced by (Console, Picardi, & Ribaudo 2000). The device (see figure 1) is composed of a pump P which delivers water to a tank TA via a pipe PI; a container CO is used to collect the water which may leak from the pipe.

The pump can exhibit three modes of behavior, the *OK* one and two faulty ones: *leaking-P* and *blocked-P*. The pipe PI exhibits two modes of behavior: the *OK* one (all the received water is delivered to the tank) and a faulty one where it is leaking (*leaking-PI*): in this case, part of the input flow goes to the container and we assume that it delivers to the tank a low output when receiving a normal or low input and no output when receiving no input.

Three sensors are available: the first one measures the flow from the pump which can be normal (*nrmp*), low (*lowp*) or zero (*zrop*); the second one records the presence of water in the container, either present (*presco*) or absent (*absco*) and the third one measures the level of the water in the tank, which can be normal (*nrmta*), low (*lowta*) or zero (*zrota*) .

The theoretical framework

In this section, we give the theoretical framework which allows us to precisely define the model description language and the observation language. We then define a *trajectory*, a

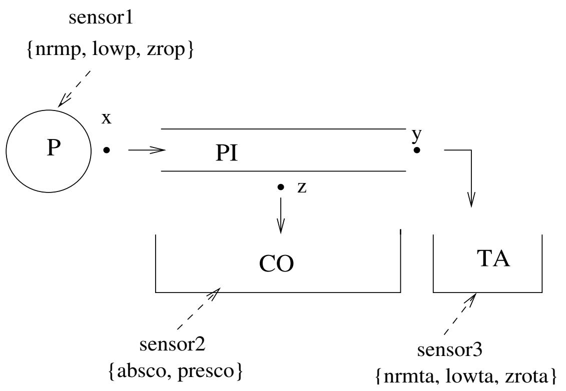

Figure 1: Scheme of the pump

trajectory compatible with observations and a *diagnosis*. To make things easier, first we give the definitions without taking into account any explicit temporal constraints. We then explain how we extend these definitions with time.

The model description language

There are many formalisms which can be used for representing discrete-event systems, the most common ones being Petri nets (Peterson 1981) and automata (Alur & Dill 1990). High-level specification languages also exist based on process algebras (Nicollin & Sifakis 1991; Gilmore & Hillston 1994), dynamic and temporal logics (Alur & Henzinger 1991). We focus throughout this paper on an abstract representation of the dynamics of the system by a finite-state machine (automaton) formalism. The system model is described as a deterministic automaton $\mathcal{A}$ that defines then a transition function from states to states by transitions. The formal definition of an automaton is the following:

Definition: An automaton $\mathcal{A}$ is defined by an triplet $< \mathcal{S}, \Sigma, \mathcal{E}, \mathcal{F}, \mathcal{L} >$ where

- $\mathcal{S}$ is a finite set of state labels;
- Σ is a finite set of transition labels;
- $\mathcal{E} \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a finite set of edges; an edge e is an 3-plet (s, σ, s') such that e links $s \in \mathcal{S}$, to $s' \in \mathcal{S}$, on an edge labelled by $\sigma \in \Sigma$; we use the notation $s' = \sigma(s)$ when $(s, \sigma, s') \in \mathcal{E}$ and suppose that σ is a function;
- $\mathcal{F}$ is a finite set of fluents (or properties);
- $\mathcal{L}$ is a function from $\mathcal{S}$ to $2^{\mathcal{F}}$; which associates to each state s the maximal set of fluents it satisfies.

Let us illustrate the just defined formalism on the example. The behavior of each component of the pump system is given in the figure 2. The states correspond to the component's states and are associated (by $\mathcal{L}$) to the values of the sensors and to the modes of behavior of components[1]. The evolution between states depends on the water flow through the system and more particularly on the connections between each component referred as points x, y and z. For example, a low flow on the point x is denoted as *xlow*. On the point z, the constant flow is *zflow*.

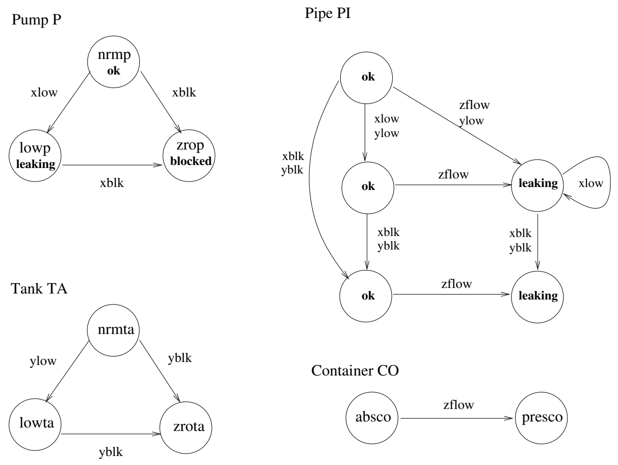

Figure 2: Component's behaviors

A system being a set of interconnected components (a pump, a pipe, and two tanks in our example), it is usually easier to describe its global behavior by describing its elementary component models. The global model can then be built by a composition operation by specifying the synchronization constraints.The synchronization is realized through the labels of the edges (i.e. transitions). The composition of automata, noted $||$, is a well-known operation defined for example in (Hopcroft & Ullman 1979; Alur & Dill 1994) and proposed in most of the tools based on this automata formalism. The product automaton $\mathcal{P}$, defined as the composition of the automata such that $\mathcal{P} = P \ || \ PI \ || \ CO \ || \ TA$, is shown in figure 3. In the following, we will denote by σ_{ij} the label of the transition joining the state i to the state j.

The observation language

A diagnostic problem has usually two components: the system description given by the model and the observations. A first important point to deal with is then to define the information which can be obtained from the set of sensors, i.e the set of observables. Two main possibilities exist: either the observables are a subset of the set of fluents which means that the states are (partially) observable, or the observables are a subset of the set of transitions which means that some of the transitions are observable[2]. These two cases are well-known and correspond to state-based transition systems (based on Kripke structures) and action-based transition systems (also called Labelled Transition Systems). In the above example, we hypothesized that the states were described by the values of the sensors and then we chose the

[1]The container and the tank having only one OK behavior mode, we do not specify it in the figures

[2]A last choice is to have both states and transitions observable as it is the case in (Barral, McIlraith, & Son 2000).

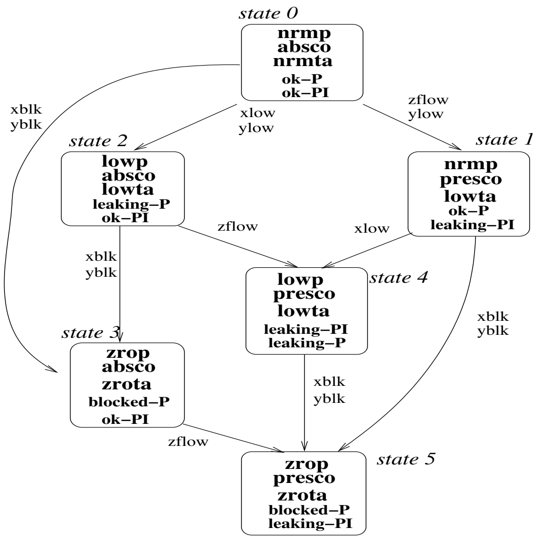

Figure 3: Product automaton of the pump system

state-based case. In the following, we focus on this case but definitions given below can be easily adapted to cover the action-based case.

In the state-based case, observations are expressed by means of observable fluents OF, with $OF \subseteq F$. In the example, we have $OF = \{nmrp, lowp, zrop, absco, presco, nrmta, lowta, zrota\}$.

A precise observation O is then a set of observable fluents. Due to the sensors, observations are often imprecise; an observation Obs is then described by a set of precise observations $\{O_1, \ldots, O_k\}$ with $O_j \subseteq OF \subseteq F$. A sequence of observations is a sequence $OBS = [Obs_1, \ldots, Obs_p]$ with $Obs_i = \{O_{i1}, \ldots, O_{ik_i}\}$.

In the following sequence of two observations, the second one is imprecise (the flow of the pump is either normal or low). We have $OBS = [\{\{nrmp, absco, nrmta\}\}, \{\{nrmp, absco, lowta\}, \{lowp, absco, lowta\}\}]$.

Trajectory

Definition: A *trajectory* $Traj(s_b, s_e)$ between two states s_b and s_e is a pair $\{[s_1, \ldots, s_n], [\sigma_1, \ldots, \sigma_{n-1}]\}$ such that $s_b = s_1$, $s_e = s_n$ and $\sigma_i(s_i) = s_{i+1}$.

Definition: The *projection on the states of a trajectory* $Traj(s_b, s_e)$, denoted $Proj_s$, is the sequence $[s_1, \ldots, s_n]$ or equivalently $[\phi_1, \ldots, \phi_n]$ where $\phi_i = \mathcal{L}(s_i)$ is the maximal set of fluents satisfied by the state s_i.

Definition: The *observable projection on the states of a trajectory* is the sequence $[O\phi_1, \ldots, O\phi_n]$ where $O\phi_i$ is such that $O\phi_i = \phi_i \cap OF$, i.e the maximal set of observable fluents satisfied by s_i.

Definition: The *projection on the transitions of a trajectory* $Traj(s_b, s_e)$, denoted $Proj_\sigma$, is the sequence $[\sigma_1, \ldots, \sigma_{n-1}]$. It is called a *path* between s_b and s_e.

The diagnostic task

The diagnostic task consists in inferring information on the faulty state of the system components and/or on the occurrence of faulty events from the sequence of observations. Even if examining the successive states is, in some cases, sufficient (Malik & Struss 1997; Panati & Theseider-Dupré 2000), we focus on the general case and consider as many authors (Baroni *et al.* 1999; Cordier & Thiébaux 1994; Barral, McIlraith, & Son 2000; Williams & Nayak 1996) that the diagnostic task relies, for dynamical systems, on finding the trajectories explaining the observations. From these trajectories, it is in general easy to extract the diagnosis itself. This extracting task is briefly discussed below.

Trajectory explaining observation sequences

Two main points have to be considered here. The first one is to decide if we consider that all the observable states or transitions are supposed to have been observed or if we accept that some observed states or transitions have not been seen by the sensors. It is clearly dependent on the application: in the case where the sensors are continuously running, as in our illustrating example, the first case can be considered. In the case where measurements are obtained by irregular snapshots, as it is the case in the experiment related in the last section, the second case is the good one. In the following, the definitions are given for the second case but can be easily adapted to cover the first one.

The second point depends on the sense which is given to "explaining". As usual, it can be interpreted as "consistent with" or "implying" the observations. Having restricted a state to be a set of fluents, this point is not discussed here, but it has to be considered as soon as the domain description language is extended and allows formulas and static laws on the states (see for instance (Barral, McIlraith, & Son 2000)).

Definition: A mapping function *map* from a sequence $[X_1, \ldots, X_p]$ to a sequence $[Y_1, \ldots, Y_n]$ is a function such that $\forall X_i \exists Y_j$ with $map(X_i) = Y_j$ and $\forall i_1 < i_2$, let $Y_{j1} = map(X_{i1}), Y_{j2} = map(X_{i2})$ we have $j_1 < j_2$.

Definition: A trajectory $Traj(s_b, s_e)$ *explains* a sequence of p state observations $[Obs_1, \ldots, Obs_p]$ iff the observable projection on the states of the trajectory $Proj_s(Traj(s_b, s_e)) = [O\phi_1, \ldots, O\phi_n]$ is such that there exists a mapping function *map* from $[1, \ldots, p]$ to $[1, \ldots, n]$ and $\forall i \in [1 \ldots p]$ $Explain(O\phi_{map(i)}, Obs_i)$ is satisfied where $Explain(O\phi, Obs)$ means that $\exists O \in Obs$ such that $O \subseteq O\phi$.

Diagnosis

As said before, we consider that the diagnostic task relies, for dynamical systems, on finding the trajectories explaining the observations. The diagnosis itself can then be obtained by extracting the information from these trajectories. This extracting task is highly related to the diagnostic language for the problem: a diagnosis can be defined as being a set of faulty events (Cordier & Thiébaux 1994), a set of behavioral modes (Console, Picardi, & Ribaudo 2000), a set of faults (Lunze 2000) or a set of faulty components (Barral, McIlraith, & Son 2000). The diagnostic task consists in abstracting the set of possible trajectories according to what is looked for. In the case of (Cordier & Thiébaux 1994), only the faulty transitions are kept. (Console, Picardi, & Ribaudo 2000) collect the behavioral modes associated to the states of the trajectories. In (Barral, McIlraith, & Son 2000), the states of each narrative are examined to detect the abnormal components. We focus on the computation of the trajectories explaining the observations and suppose that there exists an extraction function which provides the diagnoses from these trajectories.

Extending definitions by taking into account temporal constraints

In order to simplify, the definitions given above were atemporal definitions in which time only appears as the order relation in the sequence of states and observations. In this section, we extend the definitions by adding explicit temporal constraints.

Timed model and timed observations

Having chosen to describe the dynamical behavior of a system by an automaton, a direct extension is that of timed automata. Timed automata extend the automata formalism by adding clocks. Clocks are real-valued variables increasing uniformly with time. Several independent clocks may be defined for the same timed automaton. They define timing constraints associated to the states and to the transitions. A clock constraint associated to a transition governs the triggering of the transition. It is called its *guard*. The transitions are instantaneous and allow the resetting of clocks. A clock constraint associated to a state is called its *invariant*. It is possible to stay in a state as long as its invariant is true. Let us note X_i the clocks of the automata. A $date$ is an n-uplet $< x_1, .., x_n >$ where x_i is the value of X_i. In the following, we associate a date interval variable I_{s_i} to each state s_i, where the left (resp. right) bound of the interval corresponds to the input (resp. output) date. The temporal constraints can be expressed as constraints on these variables I_s and are denoted $\mathcal{K}_\mathcal{A}(I_s)$. The timed model is denoted by the pair $(\mathcal{A}, \mathcal{K}_\mathcal{A}(I_s))$.

Figure 4 illustrates the model of the container augmented with temporal constraints. We suppose that the sensor measuring the level in the container is not sensitive enough and detects water only after 5 to 8 seconds. The new global automaton is given figure 5.

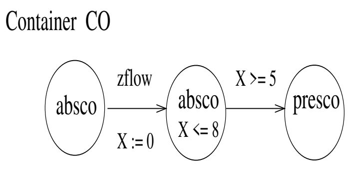

Figure 4: Introducing time in the container CO

Observations can also be constrained by temporal constraints. We consider instantaneous observations and associate a date t_{Obs} to each observation Obs. A timed observation is a pair $(Obs, \mathcal{K}_{Obs}(t_{Obs}))$.

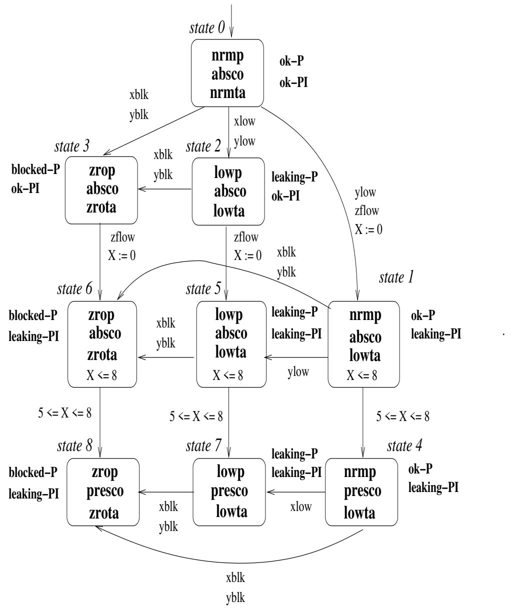

Figure 5: Timed automata of the pump system

A sequence of timed observations OBS is a pair $([Obs_1, \ldots, Obs_p], \mathcal{K}_{OBS}(t_{Obs_1}, .., t_{Obs_p}))$ where each Obs_i is an observation and where $\mathcal{K}_{OBS} = \bigcup_{i \in [1 \ldots p]} \mathcal{K}_{Obs_i}(t_{Obs_i})$ collects the constraints associated to each Obs_i. If we know, for instance, that $t_{Obs_1} = 0$ and $t_{Obs_2} = 10$, we have
$OBS = [\{\{nrmp, absco, nrmta\}\}, \{\{nrmp, absco, lowta\}, \{lowp, absco, lowta\}\}], \{t_{Obs_1} = 0, t_{Obs_2} = 10\}$.

Timed trajectories

An elementary timed trajectory is a trajectory of the model where each state and each transition has been precisely dated (a state is associated to exactly one date interval and a transition to exactly one date) and where the temporal constraints are satisfied.

The useful notion is that of timed trajectory which describes a set of elementary timed trajectories.

Definition: A *timed trajectory* of the model $\mathcal{A}$ is a pair $(Traj(s_b, s_e), \mathcal{K}_{Traj}(I_s))$ where $\mathcal{K}_{Traj}(I_s) \models$ [3] $\mathcal{K}_\mathcal{A}(I_s)$ which means that the states and transitions of the trajectory are temporally constrained and satisfy the temporal constraints of the model. If $Traj(s_b, s_e)$ is an atemporal trajectory of the model $\mathcal{A}$, the pair $(Traj(s_b, s_e), \mathcal{K}_\mathcal{A}(I_s))$ is clearly a timed trajectory of the timed model $(\mathcal{A}, \mathcal{K}_\mathcal{A}(I_s))$.

Definition: A *timed trajectory* $(Traj(s_b, s_e), \mathcal{K}_{Traj}(I_s))$ *explains* a sequence of p timed observations $([Obs_1, \ldots, Obs_p], \mathcal{K}_{OBS}(t_{Obs_1}, .., t_{Obs_p}))$ iff the observable projection on the states of the trajectory $Proj_s(Traj(s_b, s_e)) = [O\phi_1, \ldots, O\phi_n]$ is such that:
i) there exists a mapping function *map* from $[1, \ldots, p]$ towards $[1, \ldots, n]$; ii) $\forall i \in [1 \ldots p]$ $Explain(O\phi_{map(i)}, Obs_i)$ is satisfied; iii) the constraints

[3] Clearly $\models$ depends on the temporal constraint language used.

on the sequence of observations satisfy the constraints of the trajectory, i.e: $\{\mathcal{K}_{OBS}(t_{Obs_1},..,t_{Obs_p}) \bigcup \forall i \in [1\dots p], t_{Obs_i} \in I_{s_{map(i)}}\} \models \mathcal{K}_{Traj}(I_s)$. According to the mapping, each observation is mapped to a state and then the observation date has to belong to the corresponding state interval. Let us recall that $Explain(O\phi, Obs)$ means that $\exists O \in Obs$ such that $O \subseteq O\phi$.

Introduction to model-checking techniques

Generalities

Model-checking is one of the most successful techniques for automatic verification of complex systems (Henzinger *et al.* 1994; Yovine 1998). It consists in a system specification language, a property specification language and efficient algorithms called model-checkers.

The high-level specification language allows us to build a model of the system. This model is generally an automaton or more exactly a set of synchronized elementary automata. A synchronization operation is always provided allowing to define an automaton as being the synchronized product of two automata. Model-checking tools, first dedicated for classical automata, have been extended to timed automata, (Yovine 1997; Larsen, Pettersson, & Yi 1997) being the most well-known ones.

A property specification language, based on temporal logics, allows us to formally express the properties to be satisfied by the states of the automata. PCTL and CTL (Pnueli 1981; Clarke, Emerson, & Sistla 1986) are the most used temporal logics by model-checking tools. TCTL (Alur, Courcoubetis, & Dill 1993) extends CTL by allowing quantitative constraints on the delays and is used for checking timed automata. The most studied properties are reachability, safety and vivacity properties. In this paper, we focus on reachability analysis which consists in determining whether or not there exists an execution between two states described by the properties they satisfy.

The important point is the efficiency of the model-checking algorithms which do never explicitly represent all the states of the automaton but rely on symbolic methods allowing states to be concisely represented and manipulated. Binary Decision Diagrams (BDD) or Difference Bound Matrices (DBM) are some of the efficient data structures used by model-checking techniques. In the case of timed automata, a partition, named region graph, is used as an efficient abstraction of the system.

Diagnostic task and model-checking

Model-based diagnostic amounts to confronting a system model to observations, and more exactly to finding the trajectories explaining the observations. This problem has already been studied in the DX community and the efficiency of this computation is discussed for instance in (Williams & Nayak 1996; Kurien & Nayak 2000). Here we focus on the contribution of model-checking techniques to this problem. As seen before, two cases can be distinguished: in the first one, the states are (partially) observable. Diagnostic trajectories can be obtained, as will be seen below, by checking reachability properties on the states. In the second case, the transitions are (partially) observable. Diagnostic trajectories can then be obtained by synchronizing the system automata and an observer (an automaton describing the observations). It amounts to keeping from the system model the only paths accepted by the observer, which is exactly what is looked for (see for instance (Console, Picardi, & Ribaudo 2000; Pencolé 2000)). These two kind of representation are well-known and correspond to the state-based transition systems (based on Kripke structures) and the action-based transition systems (also called Labelled Transition Systems) (see (Nicola & Vaandrager 1990) for a comparison). It seems then that both approaches can be implemented by using model-checking techniques and selecting the suited tools. In this paper, we have decided to explore the first case. This choice is justified by the application which motivated the study and is discussed later. The trajectories are then computed by checking reachability properties.

Kronos

Our implementation has been realized using the tool Kronos (Yovine 1997). Kronos is a model-checking engine, dedicated to real-time systems. It has been applied on real-time communication protocols, on timed asynchronous circuits and on hybrid systems. The system is defined as a timed automaton or as several synchronized sub-automata since a parallel composition operation is provided. One of its main advantage is the property specification language which is more powerful than for example UPPAAL's one (Larsen, Pettersson, & Yi 1997). Kronos implements efficient model-checking algorithms to check whether the system satisfies properties expressed as a TCTL formula.

TCTL is a temporal logic extending CTL by allowing to express quantitative temporal constraints. The TCTL formulae are defined by the following grammar:

$$f ::= p \mid X \in I \mid \neg f \mid f_1 \vee f_2 \mid \exists \Diamond_I f \mid \forall \Diamond_I f$$

where $p \in \mathcal{F}$ is a fluent, $X \in \mathcal{X}$ is a clock and I is a time interval. Intuitively, $\exists \Diamond_I f$ means that there is an execution leading to a state where f holds at time $t \in I$ and $\forall \Diamond_I f$ means that every execution goes through a state where f holds at time $t \in I$. $f_1 \Rightarrow \exists \Diamond_I f_2$ checks whether there exists a state where f_1 holds and an execution starting from it and reaching a state in which f_2 holds at time $t \in I$. Usually a model-checker answers to the verification by *yes* or exhibits a counter-example. The model-checker Kronos has been extended to return, when a reachability property is satisfied, the sequences of transitions that lead from the initial state to the final state.

Computing trajectories using model-checking techniques

In this section, we present how to compute trajectories explaining observations by using model-checking techniques on a system described as an automaton. We illustrate the method on the observations taken as examples by (Console, Picardi, & Ribaudo 2000). We first present how to compute the trajectories of the system given two observations. We then generalize it to the case of n observations.

Two observations

Consider a timed sequence of two observations $([Obs_1, Obs_2], \mathcal{K}_{OBS}(t_{Obs_1}, t_{Obs_2}))$ where, as seen before, each observation Obs_i is described as a set of precise observations $\{O_{i1}, \cdots, O_{ik}\}$.

To compute the trajectories explaining these two observations, we ask the model-checker to check a reachability property on the (global) automaton. This reachability property expresses that there exists a trajectory joining a state of the system satisfying $Obs1$ to a state satisfying $Obs2$; moreover the trajectory has to satisfy the temporal constraints. In case of success, the model-checker returns all the paths satisfying the query. Let $TRAJ$ be this set of paths. We have $TRAJ = Proj_\sigma(\{Traj(s_1, s_2)\})$ where s_1 is a state satisfying Obs_1 and s_2 a state satisfying Obs_2.

The reachability property is expressed in the TCTL logical formalism: we consider a state and a precise observation as the conjunction of their fluents. Analogously, we consider an observation Obs as a disjunction of each of its precise observations, and then as a disjunction of conjunctions of fluents.

To obey the syntax of Kronos, the temporal constraints are in the form $X \in I = [integer_1, integer_2]$ where X denotes a clock of the timed automata. We note I_{Obs_i} the interval constraining the values of the clock X due to t_{Obs_i}[4]. Let φ be $Obs_1 \wedge X \in I_{Obs_1} \Rightarrow \exists\Diamond_{I_{Obs_2}} Obs_2$. The paths satisfying the query φ are exactly the trajectories starting from the states satisfying Obs_1 at time t_{Obs_1} and reaching the states satisfying Obs_2 at time t_{Obs_2}. The algorithm used to compute the trajectories between two observations is the following:

Trajectory Algorithm: Two observations

$TRAJ := Kronos[\mathcal{A}, Obs_1 \wedge X \in I_{Obs_1} \Rightarrow \exists\Diamond_{I_{Obs_2}} Obs_2]$

Consider first, in the pump example, the case where the two observations are precise and precisely dated. The sequence of observations is $[Obs_1, Obs_2]$ with $Obs_1 = \{\{nrmp, absco, nrmta\}\}$, and $Obs2 = \{\{nrmp, presco, lowta\}\}$. The temporal constraints on t_{Obs_1} and t_{Obs_2} are $I_{Obs_1} = [0, 0]$ and $I_{Obs_2} = [10, 10]$. Applying the previous algorithm on the pump model $\mathcal{P}$, the query to Kronos is: $Kronos[\mathcal{P}, nrmp \wedge absco \wedge nrmta \wedge X \in [0, 0] \Rightarrow \exists\Diamond_{[10,10]} nrmp \wedge presco \wedge lowta]$ where $\mathcal{P}$ is the pump model. The model satisfies the formula and the model-checker returns a unique path $TRAJ = \{[\sigma_{01}, \sigma_{14}]\}$ (see figure 5). From this, we can extract a diagnosis: at the date of the first observation, the system is ok (pump ok and pipe ok) and at the date of the second observation, the pump is ok and the pipe is $leaking$.

Let us take another example where at time t_{Obs2}, there is no water in the container and, due to a noisy sensor, the pump flow can be either normal or low. We have $Obs_1 = \{\{nrmp, absco, nrmta\}\}$ and $Obs_2 = \{O_{21}, O_{22}\}$ with $O_{21} = \{nrmp, absco, lowta\}$ and $O_{22} = \{lowp, absco, lowta\}$. The query is: $Kronos[\mathcal{P}, nrmp \wedge absco \wedge nrmra \wedge X \in [0, 0] \Rightarrow \exists\Diamond_{[10,10]}((nrmp \wedge absco \wedge lowta) \vee (lowp \wedge absco \wedge lowta))]$. The unique path provided by the model-checker is $[\sigma_{02}]$, indicating that at the date of the second observation the pump is certainly leaking. The other possible paths are discarded due to the temporal constraints.

[4]To simplify, we suppose in the following that there is only one clock denoted X.

N observations

We want now to generalize the process to the case where more than two observations are available.

A first idea consists in extending the previous principle on the n observations $[Obs_1, \cdots, Obs_n]$ and in using a TCTL formula taking into account all the observations. We get the following algorithm:

Trajectory Algorithm: N observations (1)

$TRAJ := Kronos[\mathcal{A}, Obs_1 \wedge X \in I_{Obs_1}$
$\Rightarrow \exists\Diamond_{I_{Obs_2}} (Obs_2 \Rightarrow (\exists\Diamond_{I_{Obs_3}} \cdots \Rightarrow \exists\Diamond_{I_{Obs_n}} Obs_n))]$

Let us consider the sequence of three observations such that $Obs1 = \{\{nrmp, presco\}\}$, $Obs2 = \{\{lowp\}\{absco\}\}$ and $Obs3 = \{\{zrota\}\}$ with no information on the dates of the observations. The query is: $TRAJ := Kronos[\mathcal{A}, nrmp \wedge presco \Rightarrow \exists\Diamond\,(lowp \vee absco) \wedge (\exists\Diamond\, zrota))]$. Given such a query, the model-checker answers correctly whether or not the model satisfies the property (in the above case, it answers "yes"). However, with the current version of Kronos, it is not possible to retrieve the paths explaining the observations. Moreover, this algorithm is not incremental: adding a new observation means having Kronos checking again the entire formula from the first observation to the last added one and through all the intermediate ones.

In order to answer these problems, we propose a second algorithm which is incremental and for which Kronos outputs the paths explaining the sequence of observations. It should first be clear that the only iteration of the two-observations algorithm given above (for i from 1 to $n-1$ between Obs_i and Obs_{i+1}) is not a solution to this problem. Let us consider the 3-observations example. There exists a path explaining $[Obs_1, Obs_2]$: $([\sigma_{47}])$ (see figure 5). There exists a lot of paths explaining $[Obs_2, Obs_3]$. It could then be wrongly deduced that we have a lot of solutions. But, looking at the states, we can see that there is in fact only one solution $([\sigma_{47}, \sigma_{78}])$.

It is the reason of this second algorithm in which the reachability property, instead of being expressed directly in terms of fluents, is expressed in terms of states. It requires an $observation\ model$ which maps observations into states.

Definition: An *observation model* is the function $\mathcal{L}^{-1}$ that converts a set of fluents into a set of states: $\mathcal{L}^{-1} : 2^{\mathcal{F}} \rightarrow 2^{\mathcal{S}}$.

The idea of this algorithm is to store in a set K_i all the possible states at the date of the observation Obs_i.

The general algorithm applied on a sequence of n observations is then:

Trajectory Algorithm: N observations (2)

Initialization:

$K_1 := \emptyset$

For all $O_{1j} \in Obs_1$ $K_1 := K_1 \cup \mathcal{L}^{-1}(O_{1j})$

Iteration:

For all pairs (Obs_i, Obs_{i+1}) with $i \in [1, n-1]$

$S_{(i+1)} := \emptyset$

For all $O_{(i+1)j} \in Obs_{i+1}$

$S_{(i+1)} := S_{(i+1)} \cup \mathcal{L}^{-1}(O_{(i+1)j})$

For all $s \in K_i$ and all $s' \in S_{(i+1)}$

$TRAJ' := Kronos[\mathcal{A}, s \wedge X \in I_{Obs_i} \Rightarrow \exists\Diamond_{I_{Obs_{i+1}}} s']$

If $TRAJ' \neq \emptyset$

$K_{i+1} := K_{i+1} \cup s'$

$TRAJ := Build(TRAJ', s, s')$

The function $Build$ builds the global paths from 1 to $i+1$ from the paths from 1 to i and $TRAJ'$. The function $\mathcal{L}^{-1}$ can be implemented as a Kronos query with an adequate option.

Let us illustrate the algorithm on the sequence of three observations given above. The initialization gives $K_1 = \{state4\}$. For the first pair (Obs_1, Obs_2), we get from the observation model $\mathcal{L}^{-1}(O_{21} = \{lowp\}) = S_{21} = \{state2, state5, state7\}$ and $\mathcal{L}^{-1}(O_{22} = \{absco\}) = S_{22} = \{state0, state1, state2, state3, state5, state6\}$. Thus $S_2 = \{state0, state1, state2, state3, state5, state6, state7\}$. Seven verifications are executed looking for a path between $state4$ and the states of S_2. Since the property is only satisfied between $state4$ and $state7$, $K_2 = \{state7\}$ and $TRAJ = \{[\sigma_{47}]\}$. The pair (Obs_2, Obs_3) is then examined. We get $\mathcal{L}^{-1}(O_{31} = \{zrota\}) = S_{31} = \{state3, state6, state8\}$. Finally, the unique path is $\{[\sigma_{47}, \sigma_{78}]\}$. This path allows us to explain what happened to the system: the pipe was initially leaking; the pump starts leaking and then blocked.

For sake of simplification, we have illustrated the algorithm without any temporal constraints but recall that Kronos clearly takes in charge the checking of temporal constraints along the paths.

An experimentation in monitoring agricultural plots

This work originates from an application in monitoring agricultural plots given a sequence of images. Images are related to an agricultural site and the aim is to label, at each date of snapshot, each plot of the area with a landcover type: corn, barley, forest, water, etc. The history of the plots is useful for analyzing environmental changes and is a fundamental feature more particularly regarding water pollution problems. The idea consists in modeling the evolution of the plot as a dynamic system and in comparing the observations extracted from the images against this model.

The plot behavior model encodes the knowledge about crop cycles, evolution of main crop states and rotation practices. Due to meteorological and some human factors, the transition-based model needs to deal with uncertainty on temporal constraints over transitions. Timed automata brings all the properties required to model the evolution of the plot. We consider landcover classes, as the states of the timed automaton. Date of crop calendars are expressed in a number of days between 0 and 365 and two clocks are used: one referring to the days and the other referring to the years.

The first step of the method is to translate images into observations. A preliminary classification based on the plot is applied on each image and provides a set of all plausible classes for each plot. The second step of the method is to check the observations against the plot evolution model which amounts to finding the plot trajectories explaining the observations. The method has been specified as a model-checking problem and implemented with Kronos using TCTL formulae. Kronos has been selected for the expressiveness of its query language and the efficiency of its algorithms. The method is based on a prediction and a postdiction phase detailed in (Largouët & Cordier 2000c). For instance, in the prediction phase, a reachability property verifies that each new state, proposed by the observation, is consistent with the previously known states of the plot. The method has been applied on five images of an environmental project taken at different dates (between April 1997 and August 1998) on a watershed near Rennes in France. Each image contains 2124 plots we want to classify, at each date, into ten classes known *a priori*. The results are quite satisfactory and globally better than with traditional classification methods as detailed in (Largouët & Cordier 2000b).

Conclusion

In this paper, we present model-checking techniques and demonstrate how they can be exploited for diagnosing dynamical systems represented by a discrete-event model. The main advantage of these techniques is the efficiency of the model-checker algorithms which were designed for testing complex real-time systems. Moreover, most of the existing tools provide high-level languages for building the system model and expressive query languages based on temporal logics. These techniques have been recently extended to deal with timed automata allowing the use of quantitative temporal constraints.

In (Buccafurri *et al.* 1999) it is shown how AI techniques can enhance model-checking in verification by locating an error and proposing a repair. Our contribution shows that model-checking techniques can be advantageously used by the AI community in the on-line diagnosis of systems. An incremental algorithm, based on reachability analysis, is proposed to compute the trajectories explaining a sequence of observations.

This work can be compared to (Console, Picardi, & Ribaudo 2000)'one which discusses the use of process algebras and PEPA. The main difference is that they consider Labelled Transition Systems, in which transitions are observable when we chose to consider state-based transition systems in which states are observable. In their case, a diagnosis corresponds to the synchronized product of the system model and an observer, when, in our case, the diagnosis is computed by checking reachability properties. The paper (Console, Picardi, & Ribaudo 2000) focuses on diagnosability but it would be interesting to compare more deeply these two approaches with respect to their efficiency for on-line diagnosis. At first view, one advantage is the incrementality of our algorithm which makes it adapted to an on-line analysis of a sequence of observations. Another point is that we take into account quantitative temporal constraints.

This work was motivated by an application in the monitoring of agricultural plots from a sequence of images. It has been implemented and experimented with the model-checking Kronos tool with satisfactory results.

References

Alur, R., and Dill, D. 1990. Automata for modeling real-time systems. In *Proceedings of ICALP'90, LNCS 443*. Springer-Verlag. 321–335.

Alur, R., and Dill, D. 1994. A theory of timed automata. *Theoretical Computer Science* 126(183):235.

Alur, R., and Henzinger, T. 1991. Logics and models of real time: A survey. In *Real-Time: Theory in Practice, REX Worshop*, volume 600, 74–106. Springer-Verlag.

Alur, R.; Courcoubetis, C.; and Dill, D. 1993. Model-checking in dense real time. *Information and Computation* 1:2–34.

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

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

Buccafurri, F.; Eiter, T.; Gottlob, G.; and Leone, N. 1999. Enhancing model checking in verification by AI techniques. *Artificial Intelligence* 112:57–104.

Cassandras, C. G., and Lafortune, S. 1999. *Introduction to discrete event systems*. Kluwer Academic Publishers.

Clarke, E. M.; Emerson, E. A.; and Sistla, A. P. 1986. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems* 8(2):244–263.

Console, L.; Picardi, C.; and Ribaudo, M. 2000. Diagnosis and diagnosability analysis using PEPA. In *Proceedings of ECAI'2000*, 131–135.

Cordier, M.-O., and Thiébaux, S. 1994. Event-based diagnosis for evolutive systems. In *Proceedings of DX'94* , 64–69.

Gilmore, S., and Hillston, J. 1994. A tool to support a process algebra-based approach to performance. *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* 353–368.

Henzinger, T.; Nicollin, X.; Sifakis, J.; and Yovine, S. 1994. Symbolic model checking for real-time systems. *Information and Computation* 111(2):193–244.

Hopcroft, J., and Ullman, J. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley.

Kurien, J., and Nayak, P. 2000. Back to the future for consistency-based trajectory tracking. In *Proceedings of DX'00* , 92–100.

Largouët, C., and Cordier, M.-O. 2000a. Combining observations and expectations: Application to the refinement of an image sequence classification. In *Workshop on Fusion of Domain Knowledge with Data for Decision Support, UAI'2000.*

Largouët, C., and Cordier, M.-O. 2000b. Improving the landcover classification using domain knowledge. In *ISPRS'2000 (XIXth Congress of the International Society for Photogrammetry and Remote Sensing)*, volume 4B, 538–545.

Largouët, C., and Cordier, M.-O. 2000c. Timed Automata Model to Improve the Classification of a Sequence of Images. In *ECAI'2000 (European Conference on Artificial Intelligence)*, 156–160.

Larsen, K. G.; Pettersson, P.; and Yi, W. 1997. Uppaal in a nutshell. *Journal of Software Tools for Technology Transfer* 1(1-2):134–152.

Lunze, J. 1999. Discrete-event modelling and diagnosis of quantized dynamical systems. In *10th International Workshop on Principles of Diagnosis*, 147–154.

Lunze, J. 2000. Diagnosis of quantized systems. In *Proceedings of Safeprocess'2000*, 28–39.

Malik, A., and Struss, P. 1997. Diagnosis of dynamic systems does not necessarily require simulation. In *Proceedings of DX'97* , 147–156.

Nicola, R. D., and Vaandrager, F. 1990. Action versus state based logics for transition systems. *LNCS* 469:407–419.

Nicollin, X., and Sifakis, J. 1991. An overview and synthesis of timed process algebras. In *Proceedings of the 3rd Workshop on Computer-aided Verification*.

Panati, A., and Theseider-Dupré, D. 2000. State-based vs simulation-based diagnosis of dynamic systems. In *Proceedings of the ECAI Conference (ECAI'2000)*, 176–180.

Pencolé, Y. 2000. Decentralized diagnoser approach: application to telecommunication networks. In *Proceedings of DX'2000)*, 185–192.

Peterson, J. L. 1981. *Petri net theory and the modeling of systems*. Prentice-Hall.

Pnueli, A. 1981. The temporal semantics of concurrent programs. *Theoretical Computer Science* 13:45–60.

Rozé, L., and Cordier, L. 1998. Diagnosing discrete-event systems : an experiment in telecommunication networks. In *4th International Workshop on Discrete Event Systems*, 130–137.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1994. A discrete event systems approach to failure diagnosis. In *Proceedings of DX'94* , 269–277.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Teneketzis, D. 1996. Failure diagnosis using discrete event models. *Proceedings of the IEEE Transactions on Automatic Control* 4(2):105–124.

Williams, B. C., and Nayak, P. 1996. A model-based approach to reactive self-configuring systems. In *Proceedings of DX'96* , 267–274.

Yovine, S. 1997. Kronos : A verification tool for real-time systems (kronos user's manual release 2.2). *International Journal of Software Tools for Technology Transfer* 1.

Yovine, S. 1998. Model Checking timed automata. In *Embedded Systems*. LNCS Springer-Verlag.