Formal Verification of Diagnosability via Symbolic Model Checking

Alessandro Cimatti — ITC-irst — Povo, Trento, Italy — mailto:cimatti@irst.itc.it
Charles Pecheur — RIACS/NASA Ames Research Center — Moffett Field, CA, U.S.A. — pecheur@ptolcmy.arc.nasa.gov
Roberto Cavada — ITC-irst — Povo, Trento, Italy — cavada@irst.itc.it

Abstract

This paper addresses the formal verification of diagnosis systems. We tackle the problem of diagnosability: given a partially observable dynamic system, and a diagnosis system observing its evolution over time, we discuss how to verify (at design time) if the diagnosis system will be able to infer (at run-time) the required information on the hidden part of the dynamic state. We tackle the problem by looking for pairs of scenarios that are observationally indistinguishable, but lead to situations that are required to be distinguished. We reduce the problem to a model checking problem. The finite state machine modeling the dynamic system is replicated to construct such pairs of scenarios; the diagnosability conditions are formally expressed in temporal logic; the check for diagnosability is carried out by solving a model checking problem. We focus on the practical applicability of the method. We show how the formalism is adequate to represent diagnosability problems arising from a significant, real-world application. Symbolic model checking techniques are used to formally verify and incrementally refine the diagnosability conditions.

1 Introduction

Diagnosis is the process of inferring the (most plausible) causes for the behavior of a given system, given a set of observations. In many control applications, ranging from industrial plants (e.g. production, power) to transportation (e.g. railways, avionics, space), diagnosis needs to be carried out *on-line,* in parallel with the control process. This is needed to identify whether the controlled system is working correctly or not, and to provide the controller with information on the degraded conditions (e.g. what are the malfunctioning devices), so that the appropriate counter-measures can be taken. The ability to validate such diagnosis systems becomes very important, in particular in the case of applications operating in hazardous or inaccessible conditions and carrying out vital functions.

In this paper, we focus on the key issue *of diagnosability,* i.e. the possibility for an ideal diagnosis system to infer accurate and sufficient run-time information on the behavior of the observed system. We propose a new, practical approach to the *verification* of diagnosability, making the following contributions. First, we provide a formal characterization of diagnosability problem, using the idea of *context,* that explicitly takes into account the run-time conditions under which it should be possible to acquire certain information.

Second, we show that a diagnosability condition for a given plant is violated if and only if a *critical pair* can be found. A critical pair is a pair of executions that are indistinguishable (i.e. share the same inputs and outputs), but hide conditions that should be distinguished (for instance, to prevent simple failures to stay undetected and degenerate into catastrophic events). We define the *coupled twin model* of the plant, and show that it can be used to search for critical pairs.

Third, we recast the problem in the framework of *Model Checking* [Clarke *et al,* 1999], a verification technique that is gaining increasing interest also in AI. With model checking, it is possible to exhaustively analyze the (possibly infinite) behaviors of (large sized) finite state machines, and to check if requirements expressed in terms of Temporal Logics [Emerson, 1990] are met. We show how to represent a diagnosability problem in terms of temporal logic formulae, and how to reduce it to a model checking problem over the coupled twin model.

Finally, we demonstrate the practical applicability within the Livingstone framework, a model-based diagnosis system developed at NASA Ames Research Center [Williams and Nayak, 1996]. We developed a platform able to generate formal models for the twin plant, starting from Livingstone models. Several diagnosability problems corresponding to interesting scenarios from real-world applications were tackled by means of the NuSMV model checker [Cimatti *et al,* 2002], An experimental analysis shows that the verification of diagnosability can be practical: large Livingstone models of space transportation systems are automatically analyzed within seconds by means of SAT-based symbolic model checking techniques.

The paper is organized as follows. In Section 2, we state our working hypotheses. In Section 3 we formalize the problem, while in Section 4 we characterize our approach. In Section 5, we discuss the application of model checking techniques. In Section 6, we describe the experimental framework. Finally, Section 7 reviews similar work, and Section 8 draws some conclusions and outlines future lines of activity.

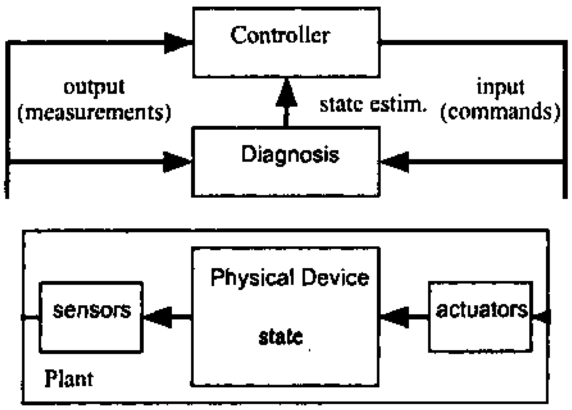

Figure 1: Architecture of a diagnosis system

2 Working Hypotheses

We consider a *diagnosis system* connected to a feedback control loop between *a plant* and its *controller* (Figure 1). The inputs of the plant are the *commands* issued by the controller; its outputs are *measurements* returned back to the controller. The role of the diagnosis system is to observe the inputs and the outputs to the plant, and to report a *state estimation,* tracking the evolution of the unobservable state of the plant. This task is substantially more complex than diagnosis for a combinatorial, memory-less system. An estimation consists of a set of possible states of the plant, in the following referred to as a *belief state.* Although diagnosis might rank these states based on likelihood, as a first approach we ignore that aspect. We also disregard issues such as the correspondence between the model of the plant and the plant itself. We will focus on plants modeled as finite-state discrete systems. We assume that we have a model of the plant as a partially observable transition system, according to the following definition.

Definition 1 *A (partially observable)* plant *is a structure* $\langle X, U, Y, \delta, \lambda\rangle$, *where X,* 17, *Y are finite sets, respectively called the* state space, input space *and* output space, $\delta \subset X \times U \times X$ *is the* transition relation, *and* $\lambda \subset X \times Y$ *is the* observation relation.

We assume that a p $P = \langle X, U, Y, \delta, \lambda\rangle$ is give. We use $x, x_0, x_1, \ldots$ to denote states of P, $u, u_0, u_1, \ldots$ to denote inputs of P, y, y_0, y_1, • • • to denote outputs of P. We write $x \xrightarrow{u} x'$ for $(x, u, x') \in \delta$, and x/y for $(x, y) \in \lambda$. The state is the "hidden" part of the plant: only the sequences of inputs and outputs are observable. P covers all types of behaviors that diagnosis is expected to handle — including faulty behaviors, with *X* containing faulty states. In general, *P* need not be deterministic. Thus, the state after a transition may not be uniquely determined the state before the transition and by the input. Observability is modeled by associating to each state a (non empty) set of possible outputs. It is therefore possible to model different forms of observation, e.g. when the information conveyed by the sensors is also uncertain. In this paper, we present the (input, state and output) spaces of a plant by means of assignment to (input, state and output) variables ranging over finite sets of values; the transition and observation relation can be compactly presented with boolean formulae. (Notice however that the content of the paper is independent of the specific description used to present P.)

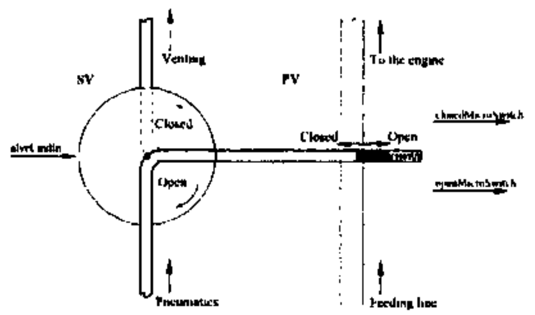

Figure 2: A simple example

Definition 2 *A* feasible execution *of $k \geq 0$ steps in P is a sequence $\sigma = x_0, y_0, u_1, x_1, y_1, u_2, \ldots, u_k, x_k, y_k$, such that $x_{i-1} \xrightarrow{u_i} x_i$ for $1 \leq i \leq k$, and x_i/y_i for $0 \leq i \leq k$. We define Σ_P as the set of all feasible executions of P. The* observable trace *of a feasible execution σ is $w = y_0, u_1, y_1, \ldots, u_k, y_k$. We write $\sigma : x_0 \xrightarrow{w} x_k$, if such a a exists.*

The above definition defines the dynamics of a plant and its observable counterpart. Notice that if an execution σ has *k* steps, then the corresponding t $w \in Y \times (U \times Y)^k$ The set of traces is in general a subset of $Y \times (U \times Y)^*$. In the following we use σ to denote a feasible execution, and *w* to denote the corresponding (observable) trace. For explanatory purposes, we consider the plant outlined in Figure 2, that is a simplified segment of the real-world application described in Section 6. A pneumatic valve (PV) controls the flow of a feeding line to the engine, and it is closed in its resting position (determined by the action of a spring). A solenoid valve (SV), when open, can open PV by directing the pneumatic flow against the spring. When SV is closed, the action of the spring is no longer contrasted and PV can close. SV can receive an input (open/close), but has no observables. PV has two sensors, each providing information on a position (e.g. whether it is open or not). Both valves can become stuck in their current position. The position of the valves influences the pneumatic and engine flows. Each of the valves has two unobservable variables: state, with values open and closed, and failure mode, with values stuck and ok. The input variable can take the values no-cmd, open, and close. The effect of a command depends on the current status. Failures can occur at any time, and are permanent. Sensors of PV are also associated with a failure_mode variable, that determines what information is conveyed.

3 Diagnosability

Run-time diagnosis starts from a (possibly partial) initial knowledge, observes the sequence of inputs and outputs, and tries to update a belief state that is an estimate of the possible states of the plant. For instance, if S V is commanded open, and the PV is not sensed open, then some problem should be diagnosed. It could be that SV or PV are stuck, or even that both PV sensors are not working correctly. Ideally, a diagnosis function should return belief states that are as specific (i.e. as small) as possible, but include the actual state of the plant.

Definition 3 *A* diagnosis function *for P is a function $\Delta : 2^X \times Y \times (U \times Y)^* \to 2^X$. Let $\hat{x}_0 \subset 2^X$. A diagnosis value $\hat{x} = \Delta(\hat{x}_0, w)$ is* correct *w.r.t. $\hat{x}_0$ and w, if and only if, for any $x_0 \in \hat{x}_0$ and x such that $x_0 \xrightarrow{w} x$, we have $x \in \hat{x}$. Δ is* correct *if and only if $\Delta(\hat{x}_0, w)$ is correct for any $\hat{x}_0$ and w.*

The intuition behind correct diagnosis values is that they encompass all potential current states, so that they cannot miss the actual state. In the following, we restrict ourselves to correct diagnosis functions.

Diagnosability aims at detecting if (parts of) the hidden state can be accurately tracked by looking at the observable traces. We use diagnosis conditions to specify which information on the state we are interested in distinguishing.

Definition 4 *A* diagnosis condition *for a plant P is a pair of nonempty sets of states $c_1, c_2 \subset X$, written $c_1 \perp c_2$.*

We can express *fault detection,* i.e. telling if any fault is present ($fault \perp \neg fault$), or *fault separation,* i.e. distinguishing between different faults (or fault classes) ($fault_a \perp$ *faulty).* Intuitively, a diagnosis value is not satisfactory if it intersects with both sides of the condition. In the example, a fault separation condition is not satisfied if we have a belief state containing both a state where PV is faulty, and one where SV is faulty.

However, it would be unrealistic to require that diagnosis provide correct and exact state estimations, instantaneously and under all circumstances. For instance, a stuck PV will stay unnoticed at least as long as SV is not operated. Rather, we require a diagnosis system to be able to decide between alternative conditions on the state of the system in the *context* where the distinction becomes critical.

Definition 5 *A* diagnosis context *for P is a structure $C = \langle\theta, \Sigma_{12}\rangle$, where θ is an equivalence relation on X, and $\Sigma_{12} \subset \Sigma_P \times \Sigma_P$. $\hat{x}_0$ satisfies θ ($\hat{x}_0 \models \theta$), iff $\hat{x}_0 \times \hat{x}_0 \subset \theta$. $(\hat{x}_0, w) \models C$ iff $\hat{x}_0 \models \theta$, and there exist $(\sigma_1, \sigma_2) \in \Sigma_{12}$, $x_{01}, x_{02} \in \hat{x}_0$, such that $\sigma_1 : x_{01} \xrightarrow{w} x_1$ and $\sigma_2 : x_{02} \xrightarrow{w} x_2$.*

Intuitively, *0* defines the initial conditions under which diagnosability is to be investigated, by inducing of a set of disjoint belief states. Basically, the initial belief state *XQ* must fall within one of the belief states induced by θ (the condition on θ can be stated as $(x_{01}, x_{02}) \in \theta$ for all $x_{01}, x_{02} \in \hat{x}_0$). For example, θ can partition states according to the positions of PV and SV, to capture the assumption that we initially (only) know those positions. Σ_{12} characterizes pairs of relevant executions. For instance, we may want to express the fact that the controller commands SV to be open and closed at least one time. In the case of fault detection, we might want to state that the elements of Σ_{12} are pairs of traces, there the first are without faults, while the second ones have exactly one fault. Notice that expressing Σ_{12} as $\Sigma_1 \times \Sigma_2$ would be inadequate: for instance, in fault separation it would be impossible to constrain the two runs to have different failures. In the following we assume that a context $C = \langle\theta, \Sigma_{12}\rangle$ is given. The notion of diagnosability is precisely characterized as follows.

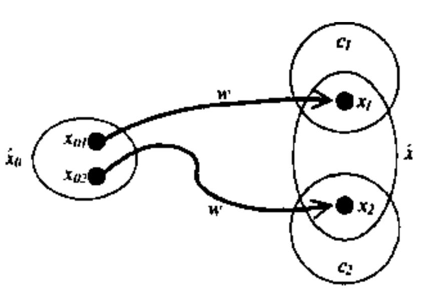

Figure 3: Critical pair

Definition 6 *A diagnosis value $\hat{x}$ satisfies $c_1 \perp c_2$, written $\hat{x} \models c_1 \perp c_2$, if and only if either $\hat{x} \cap c_1 = \emptyset$ or $\hat{x} \cap c_2 = \emptyset$. A diagnosis function Δ satisfies $c_1 \perp c_2$ over C, written $\Delta, C \models c_1 \perp c_2$, if and only if, for all $(\hat{x}_0, w) \models C$, we have $\Delta(\hat{x}_0, w) \models c_1 \perp c_2$. A condition $c_1 \perp c_2$ is* diagnosable *in P over C if and only if there exists a correct diagnosis function that satisfies it.*

4 Diagnosability as Coupled Reachability

Our approach to reasoning about diagnosability is based on refutation, i.e. we search for ways in which the diagnosability property can be violated. Intuitively, we verify the diagnosability of $c_1 \perp c_2$ by checking that *P* does not have a *critical pair,* i.e. two executions with identical observable traces, one leading to c_1, one leading to c_2 (see Figure 3).

Definition 7 *A* critical pair *of a plant P, with trace w, for a diagnosis condition $c_1 \perp c_2$, is a pair of feasible executions $\sigma_1 : x_{01} \xrightarrow{w} x_1$ and $\sigma_2 : x_{02} \xrightarrow{w} x_2$, written $\sigma_1|\sigma_2 : x_{01}|x_{02} \xrightarrow{w} x_1|x_2$, such that $x_1 \in c_1$ and $x_2 \in c_2$. The critical pair $\sigma_1|\sigma_2$ is covered by a diagnosis context $C = \langle\theta, \Sigma_{12}\rangle$, iff $(x_{01}, x_{02}) \in \theta$, $(\sigma_1, \sigma_2) \in \Sigma_{12}$.*

The absence of critical pairs for $c_1 \perp c_2$ in a given context is a necessary and sufficient condition for $c_1 \perp c_2$ to be diagnosable in that context.

Theorem 8 *$c_1 \perp c_2$ is diagnosable over C in P, if and only if P has no critical pair for $c_1 \perp c_2$ in C.*

To prove the theorem, we introduce the notion *of perfect diagnosis,* written $\overline{\Delta}$, i.e. the most specific correct diagnosis function that can be made assuming full knowledge of *P.*

Definition 9 *The* perfect diagnosis *for a plant P is the diagnosis function $\overline{\Delta}(\hat{x}_0, w) = \{x \mid \exists x_0 \in \hat{x}_0 . x_0 \xrightarrow{w} x\}$*

Given an initial belief state X_0 and trace w, $\overline{\Delta}(\hat{x}_0, w)$ returns exactly all states that can be reached from x_0 through *w.* It is easy to see that $\overline{\Delta}$ is correct, and that it is the most informative diagnosis function, i.e. $\overline{\Delta}(\hat{x}_0, w) \subset \Delta(\hat{x}_0, w)$ for any correct Δ. We observe that a condition $c_1 \perp c_2$ is diagnosable in a plant *P* over a context C if and only if $\overline{\Delta}, C \models c_1 \perp c_2$: from Definition 6, if $\hat{x} \subset \hat{x}'$ and $\hat{x}' \models c_1 \perp c_2$, then $\hat{x} \models c_1 \perp$ C_2. Therefore, $\overline{\Delta}$ satisfies any diagnosable condition, since its diagnosis values are more specific than any other correct diagnosis function.

Proof of Theorem 8 $c_1 \perp c_2$ is not diagnosable in *P* over *C;* if and only if there exist X_0 and *w* such that $\overline{\Delta}(\hat{x}_0, w), C \not\models$

$c_1 \perp c_2$; if and only if there exist $(\sigma_1,\sigma_2) \in \Sigma_{12}$ and $x_{01}, x_{02} \in \hat{x}_0$ such that $\sigma_1 : x_{01} \xrightarrow{w} x_1$ and $\sigma_2 : x_{02} \xrightarrow{w} x_2$ if and only if there exists a critical pair for $\sigma_1|\sigma_2$ for $c_1 \perp c_2$ in C. Q.E.D.

In order to search for critical pairs in P, we build the *coupled twin plant* for P, written $P \cdot P$, i.e. a "Siamese twins" plant, made out of two copies of P, whose inputs and outputs are forced to be identical.

Definition 10 *The* coupled twin plant *of P is the plant $P{\cdot}P = \langle X{\cdot}X, U, Y, \delta{\cdot}\delta, \lambda{\cdot}\lambda\rangle$, where $X{\cdot}X \subset X^2$, and, for all $x_1, x_2 \in X$, $(x_1,x_2) \in X \cdot X$ iff there exists $y \in Y$ such that $(x_1, y) \in \lambda$ and $(x_2, y) \in \lambda$; $((x_1,x_2),u,(x'_1,x'_2)) \in \delta\cdot\delta$ iff $(x_1,u,x'_1) \in \delta$ and $(x_2,u,x'_2) \in \delta$; $((x_1,x_2),y) \in \lambda\cdot\lambda$ iff $(x_1,y) \in \lambda$ and $(x_2,y) \in \lambda$.*

In the following, we assume that the coupled twin plant $P \cdot P$ of P is given. The importance of the coupled twin plant construction is shown by the following theorem.

Theorem $\sigma_1 : x_{01} \xrightarrow{w} x_1$ *and* $\sigma_2 : x_{02} \xrightarrow{w} x_2$ *are two feasible executions in P iff and only if* $\sigma_{12} : (x_{01}, x_{02}) \xrightarrow{w} (x_1, x_2)$ *is a feasible execution in* $P \cdot P$.

This is easily demonstrated by induction on the length k of executions. When $k = 0$, then $w = y_0$. From Definition 10, x_{01}/y_0 and x_{02}/y_0 iff $(x_{01}, x_{02}) \in X \cdot X$ and $((x_{01}, x_{02}), y_0) \in \lambda \cdot \lambda$. The step case is proved by extending with a transition the executions of length k, for which the theorem holds. From Definition 10, the transitions in $P \cdot P$ mimic the transitions in P. For the last observation, we reason as for the base case.

5 Diagnosability via Model Checking

We work in the framework of Model Checking [Clarke *et al*, 1999], a formal verification technique that is increasingly applied to the design of industrial digital systems (e.g. communication protocols, hardware design). Model checking allows to verify if the (possibly infinite) behaviors of a system satisfy a given property. The system being analyzed is presented as a Kripke structure, while properties are expressed in a temporal logic [Emerson, 1990]. Model checking algorithms are based on the exhaustive exploration of the Kripke structure. The technique is completely automatic and, if a property is violated, it can produce a counterexample, i.e. a simulation witnessing the violation. Symbolic model checking [McMillan, 1993] is based on the manipulation of sets of states described by boolean formulae; efficient engines, such as Binary Decision Diagrams (BDDs) [Bryant, 1986] and SAT solvers [Moskewicz *et al.*, 2001], provide a basis for efficient representation and exploration of the search space.

Our approach to diagnosability inherits several elements from model checking. The first is that a plant P can be associated with a *Kripke structure Kp* representing its behavior. This makes it possible for us to directly analyze a plant with model checking techniques. A Kripke structure is basically a nondeterministic transition system, where transitions are not labeled, while information is associated with states (i.e. each state is associated with a valuation to the state variables of the structure). The mapping from plant to Kripke structure is rather simple, and is based on the idea that the state, input and output spaces of the plant can be encoded into the state space of the Kripke structure. The information on the inputs, labeling the transitions in P, simply becomes part of the state of K_P, and is to be interpreted as "the input that will be processed in the next transition". More formally, each state s in K_P is associated with a valuation that characterizes a triple $x/y \xrightarrow{u}$, i.e. P is in state x, output y is observed and input u is received. The dynamics of P directly induce the dynamics of K_P. For each feasible execution σ in P of the form $x_0, y_0, u_1, x_1, y_1, u_2, \ldots, u_k, x_k, y_k$ there is a corresponding *path* in K_P, i.e. a sequence of states $s_0, s_1, \ldots s_k$ where each s_i is associated with the triple $x_i/y_i \xrightarrow{u_{i+1}}$. The same mapping lifts to the coupled twin plant construction. (In the above description, we omit a few technical details, having to do in particular with the fact that a Kripke structure is assumed to be total, i.e. every state has at least one successor. In this way, it is possible to assume that the analyzed paths are infinite. Since inputs become part of the state, it is possible that the transition relation is no longer total. There are standard workarounds to this problem, routinely used in verification and in AI planning. See for instance [Cimatti and Roveri, 2000].) In the following, we assume that Kripke structure $K_{P\cdot P}$ corresponding to the coupled twin plant $P \cdot P$ is given.

The second element from model checking is the *symbolic representation.* The state of $K_{P\cdot P}$ is defined with a vector of variables (x_1, x_2, u, y), respectively ranging over X, X, U and Y. We can use formulae to characterize sets of states. In $K_{P\cdot P}$ we express atomic propositions over such variables. We can have equalities between variables and the corresponding values: `P1.PV.input = close` denotes the set of states where the first instance of PV receives the `close` command. Variables of the same type can be equated: `P1.SV.failure_mode = P2.SV.failure_mode` describes the set of states where the two instances of SV in the twin plant have the same failure mode. Any subset c of $X \times X \times U \times Y$ can be described with a formula $c(x_1, x_2, y, u)$. For instance, the formula $c_1(\mathbf{x_1}) \wedge c_2(\mathbf{x_2})$ expresses a state of the twin plant where the first instance is in C_1 and the second is in C_2. Similarly, $\theta(\mathbf{x_1}, \mathbf{x_2})$ is a (propositional) formula expressing, in symbolic form, the equivalence relation θ.

The third ingredient from model checking is the use of *temporal logics* to characterize behaviors of the system over time. We use LTL (Linear Temporal Logic [Emerson, 1990]), where formulae are constructed starting by combining atomic propositions with boolean operators and *temporal modalities.* If ϕ and ψ are LTL formulae, so are F ϕ (sometimes in the future ϕ), G ϕ (always in the future ϕ), ψ U ϕ (sometimes in the future ϕ and, until then, ψ), X ϕ (in the next time step ϕ). An LTL formula ϕ holds on a path π (written $\pi \models \phi$), iff ϕ is true in π at (step) 0 (written $\pi^0 \models \phi$). If p is an atomic proposition, then $\pi^i \models p$ iff p is true according to the assignment associated with the i-th state of π. Boolean connectives have the standard interpretation (e.g. $\pi^i \models \phi \wedge \psi$ iff $\pi^i \models \phi$ and $\pi^i \models \psi$). The interesting cases of temporal operators are as follows. $\pi^i \models$ F ϕ iff there exists $j \geq i$ such that $\pi^j \models \phi$. $\pi^i \models$ G ϕ iff for all $j \geq i$ $\pi^j \models \phi$. $\pi^i \models \psi$ U ϕ iff there exists

```
MODULE SV_type (cmd_in)


MODULE PV_type ( )
 VAR pneumaticsLineln : {abvThresh,blwThresh};
 VAR valvePositionValue : {open,closed};
 VAR mode : {stuckOpen,nominal,stuckClosed};

 DEFINE _faultmodes := {stuckOpen,stuckClosed};
 DEFINE _broken := mode in __faultmodes;
 DEFINE _brokencount := _broken;
 DEFINE _brokenprob :=
  case
    mode = stuckOpen   : 3;
    mode = stuckClosed : 3;
    else               : 0;
  esac;

MODULE EX_type (sv_cmd_in, ...)
 SV : SV_type(sv_cmd_in);
 PV : PV_type( );

 DEFINE _brokencount := SV.__broken +
                        PV. broken +
```

Figure 4: The SMV model for the example

$j \geq i$ such that $\pi^j \models \phi$ and, for all $i \leq h < j$, $\pi^h \models \psi$. $\pi^i \models X\phi$ iff $\pi^{i+1} \models \phi$. An (existential) model checking problem $K \models \phi$, read ϕ (existentially) holds in K, amounts to detecting if K admits a path where ϕ holds.

A diagnosis condition with a context is represented with temporal logic formula ϕ such that $K_{P.P} \models \phi$ holds iff there is a critical pair. We express reachability in $P \cdot P$ of a critical pair for a diagnosis condition $c_1 \perp c_2$ with the formula: $\mathsf{F}\,(c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2))$. Model checking can be run on the problem $K_{P.P} \models \mathsf{F}\,(c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2))$. If the answer is true, then we have a witness for the a critical pair. Given the context $C = \langle\theta, \Sigma_{12}\rangle$, we enforce the initial condition θ with the formula $\theta(\mathbf{x}_1, \mathbf{x}_2) \wedge \mathsf{F}\,(c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2))$. The conjunction has the effect of restricting the analysis to the paths in *Kp.p* starting from the states that satisfy θ. Σ_{12} is taken into account assuming that a characterization of the sets of traces is described by an LTL formula $\Sigma_{12}(\mathbf{x}_1, \mathbf{x}_2\mathbf{y}, \mathbf{u})$. The model checking problem corresponding to diagnosability is $K_{P.P} \models \theta(\mathbf{x}_1, \mathbf{x}_2) \wedge \Sigma_{12}(\mathbf{x}_1, \mathbf{x}_2\mathbf{y}, \mathbf{u}) \wedge \mathsf{F}\,(c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2))$. In practice, several simplifications are possible. Often $\Sigma_{12}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{u}, \mathbf{y})$ can be expressed in terms of propositional constraints $\phi_{12}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{u}, \mathbf{y})$, that must hold on all the states of the execution; in such cases diagnosability in context *C* is basically represented by the LTL formula $\theta(\mathbf{x}_1, \mathbf{x}_2) \wedge (\phi_{12}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{u}, \mathbf{y})\ \mathsf{U}\ (c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2)))$. Notice that the formula holds if the path condition ϕ_{12} holds until the conditions c_1 and c_2 are reached. When the context does not constraint the executions, the above formula is equivalent to $\theta(\mathbf{x}_1, \mathbf{x}_2) \wedge \mathsf{F}\,(c_1(\mathbf{x}_1) \wedge c_2(\mathbf{x}_2))$.

```
MODULE main
VAR
  sv_cmd : { no_cmd, open, close };
  PI : EX_type(sv_crnd, ...);
  P2 : EX_type(sv_cmd, ...);
INVAR
 (PI.PV.sense_open = P2.PV.sense_open) & ...
```

Figure 5: The SMV schema for the twin plant

6 Experimental Evaluation

We analyzed the practical applicability of our approach within the Livingstone framework. Livingstone is a model-based health monitoring system developed at NASA Ames [Williams and Nayak, 1996]. It uses a model of a physical system, such as a spacecraft, to infer its state and diagnose faults from observations. Livingstone is one of the three parts of the Remote Agent (RA), an autonomous spacecraft controller developed by NASA Ames Research Center jointly with the Jet Propulsion Laboratory. RA was the first AI software to control an operational spacecraft [Muscettola *et al,* 1998]. Livingstone has also been used in other applications such as the control of the In-Situ Propellant Production system (1SPP) for Mars missions [Clancy *et al,* 1999], the monitoring of a mobile robot [Simmons *et al.,* 2001], and Intelligent Vehicle Health Management (IVHM) for experimental space transportation vehicles [Bajwa and Sweet, 2002].

Livingstone uses a qualitative relational model describing the evolution of observable and hidden variables. Continuous physical domains are abstracted into discrete intervals such as {low, nominal, high}. Each component has a mode variable identifying its nominal and fault modes. Livingstone models are specified in a hierarchical, declarative formalism called JMPL, or using a graphical development environment. Livingstone observes the commands issued to the plant and uses the model to predict the plant state. It then compares the predicted state against observations received from the actual sensors. If a discrepancy is found, Livingstone performs a diagnosis by searching for the most likely configuration of component modes that are consistent with the observations.

Livingstone models directly induce a synchronous transitions systems, very similar to a plant model. Pecheur and Simmons [Pecheur and Simmons, 2000] have developed a translator able to convert a Livingstone model and a related set of specifications in the language of the SMV model checker [McMillan, 1993], and to convert back the diagnostic traces in terms of the Livingstone model. Figure 4 outlines the structure of the example plant in SMV language. For each component type, there is a corresponding module. The first module statements define the model for the SV and the PV. For each of the components, a set of variables is defined, the dynamics of which is directly induced from the Livingstone model. Notice the _brokenprob variable, whose numerical value is the (negated) logarithm of the probability of failure for the component (e.g. 10^{-3}). This enables for a (rough) analysis of failure probabilities. These modules are instantiated in the EX module, with a parameter representing the commands to PV.

In order to tackle diagnosability, we devised a systematic way of constructing the SMV coupled twin plant of a Livingstone model. The construction is outlined in Figure 5. The EX plant is instantiated twice in the main, at top level, generating PI and P2. The same input variable (sv_cmd) is given in input to both instances. Then, the outputs of the two instances of the plant are constrained to exhibit the same behavior by means of an INVAR statement, i.e. a condition that must hold in all states. The SMV language enables for the specification of context. The basic building blocks of the properties are propositional conditions over states, that can be expressed by means of the DEFINE construct. For instance, it is possible to express conditions on the number of failures, using the _brokencount variable defined in Figure 4. For instance, PI._brokencount < 2 in the main module states that at most one failure can occur in the first instance of the circuit.

In the experimental analysis, we tackled several problems for the Livingstone model of the Main Propulsion System for the X-34, a next-generation vehicle for the Space Shuttle [Bajwa and Sweet, 2002]. We interacted with the NASA experts of diagnosis, to check the representational adequacy of our formalism, and to characterize diagnosability problems of practical relevance. The problems were defined starting from simulation runs, that tested a specific fault in a specific context. We remark the impact of our approach is far beyond the one of testing, since it performs an exhaustive analysis (though at a higher abstraction level) within the cases captured by the context. The experimental evaluation was carried out by running different symbolic model checking tools on models described in SMV language, such as the CMU SMV [McMillan, 1993], Bwolen Yang's version of SMV [Yang *etal,* 1999], and NuSMV [Cimatti *et al,* 2002]. While the former ones are based on Binary Decision Diagrams (BDDs) [Bryant, 1986], NuSMV also enables the use of SAT-based techniques [Moskewicz *et al,* 2001]. This activity suggested several considerations. First, the experimental analysis was fundamental to tune the formalism. Several improvements (e.g. the notion of context) were conceived while trying to encompass representational issues arising in practice. Second, contexts are incrementally characterized. Although the requirements for the diagnosis system can usually suggest an initial version for the context, it is seldom the case that the precise conditions for diagnosability are known (or can be precisely stated) in advance. The ability of the model checker to find critical pairs was very useful in the refinement, since it helped to explain why diagnosability fails, i.e. to understand whether the context is not strict enough, or because a problem was found.

In terms of performance, the critical factor was the size of the models to be analyzed. The most significant plant we analyzed has about 800 scalar variables. This number almost doubles in the case of the twin plant, and (after the elimination of equivalent variables) we are left with models having about 600 state variables. Different versions of SMV were used to tackle the resulting diagnosability problems. All the BDD-based verification engines were defeated by the size of models for coupled twin plants (i.e. no solution found after running for 24hs). This failure occurred despite the use of advanced techniques such as dynamic reordering, invariant discovery, and conjunctive partitioning. We remark that some of these BDD-based engines, most notably Bwolen Yang's SMV, were able to tackle verification problems on *single* plants very efficiently. On the other hand, the use of model checking techniques based on SAT solving, implemented in the NuSMV system [Cimatti *etal,* 2002], proved to be very effective on these problems. We used a combined approach, integrating bounded model checking (oriented to finding bugs) and inductive reasoning. The SAT-based engine of NuSMV was able to solve all the verification problems in less than two seconds. The analysis of the results shows that Livingstone models tend to feature a huge state space but little depth; therefore, the symbolic processing provided by SAT turns out to be very appropriate. It is worth mentioning that, while trying to refine a diagnosability property, we discovered an unexpected behavior in the model of the X-34. Further analysis highlighted a missing statement in the description. This result is quite significant, since the model had been repeatedly tested.

7 Related Work

The idea of diagnosability has received a lot of attention in the framework of Discrete Event Systems. In [Sampath *et al,* 1995; 1996], diagnosability is precisely defined and an algorithm for checking diagnosability is presented. The approach is limited to failures represented as reachability properties. Jiang and Kumar [Jiang and Kumar, 2002] generalize the approach to the case of failures described as formulae in linear temporal logics. The approach is based on a polynomial algorithm for testing the diagnosability, formulated with techniques from automata theory [Jiang *et al,* 2001]. In particular, they define a self-product automaton similar to our twin plant. Console, Picardi and Ribaudo [Console *et al,* 2000] propose the use of a particular form of process algebras, PEPA, for the formalization and the analysis of diagnosis and diagnosability problems.

Our work is rather different from the works mentioned above, that are mostly oriented to the definition of the theoretical framework, and do not address the problems related to the practical application of the proposed techniques. Our objective is the definition of an effective platform for the analysis of diagnosability, that can be practically applied in the development process of diagnosis systems. The "twin-models" approach allows us to directly reuse standard model checking tools, without having to reimplement a complex tableau construction described in [Jiang *et al,* 2001]. Furthermore, our approach preserves the semantics of the problem, thus making it possible to tune the decision procedure to the application domain. In terms of expressivity, our work shares several underlying assumptions with [Sampath *et al,* 1995; 1996], considering failures that can be represented as reachability conditions. Compared to [Jiang *et al,* 2001], we only tackle zero-delay diagnosability, although it seems that our framework could be extended in this respect.

Our approach makes no hypothesis on the way the controller exploits the information provided by the diagnosis system. For this reason, we introduce the notion of context in order to qualify the conditions under which diagnosability should hold. In active diagnosis [Sampath *et al,* 1998], the controller is designed taking into account the issues of diagnosability. Similar problems are also tackled in planning under partial observability, where the planner can decide the most appropriate actions to diagnose the fault, e.g. by probing the system with actions that will provide suitable information, and recover from it (see for instance [Bertoli *et al,* 2002]).

8 Conclusions

In this paper, we have proposed a novel approach to the verification of diagnosability, with emphasis on its practical applicability. Our work is based on a new conceptualization of the problem, with the twin plant construction and the use of temporal logic formulae to describe the context of a diagnosability problem. To the best of our knowledge, this is the first approach to diagnosability that enables the direct exploitation of symbolic model checking technology. We tackled significant diagnosability problems from a real-world application, discussed a practical methodology for the incremental refinement of diagnosis contexts, and were able to verify large-sized problems. In the future, we will try to take into account the fact that diagnosis can propose several candidates, with different degrees of likelihood. A compositional approach to verification, exploiting the modular structure of the design, will be investigated. In the longer term, we plan to tightly integrate the approach within the Livingston toolset, in order to allow Livingstone application developers to use model checking to assist them in designing and correcting their models, as part of their usual development environment.

References

[Bajwa and Sweet, 2002] A. Bajwa and A. Sweet. The Livingstone Model of a Main Propulsion System. In *Proc. IEEE Aerospace Conference.* IEEE, 2002.

[Bertoli et al.,2002] P. Bertoli, A. Cimatti, J. Slaney, and S. Thiebaux. Solving power supply restoration problems with planning via symbolic model checking. In *Proc. ECAI'02,* Lyon, France, 2002.

[Bryant, 1986] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers,* C-35(8):677-691, August 1986.

[Cimatti and Roveri, 2000] A. Cimatti and M. Roveri. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research (JAIR),* 13:305-338, 2000.

[Cimatti *et al,* 2002] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. Integrating BDD-based and SAT-based symbolic model checking. In *Proc. FROCOS,* volume 2309 of *LNAI,* pages 49-56. Springer-Verlag, April 2002.

[Clancy *et al,* 1999] D. Clancy, W. Larson, C. Pecheur, P. Engrand, and C. Goodrich. Autonomous control of an in-situ propellant production plant. In *Proceedings of Technology 2009 Conference,* November 1999.

[Clarke *et al.,* 1999] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* The MIT Press, Cambridge, Massachusetts, 1999.

[Console *etai,* 2000] L. Console, C. Picardi, and M. Ribaudo. Diagnosis and diagnosability using Pepa. In *Proc. ECA1 '00,* pages 131-136, Berlino, Germany, August 2000. IOS Press.

[Emerson, 1990] E. A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics,* chapter 16, pages 995-1072. Elsevier, 1990.

[Jiang and Kumar, 2002] S. Jiang and R. Kumar. Failure diagnosis of discrete event systems with linear-time temporal logic fault specications, 2002. IEEE Trans, on Automatic Control.

[Jiang *et al,* 2001] S. Jiang, Z. Huang, V. Chandra, and R. Kumar. A polynomial algorithm for testing diagnosability of discrete event systems. *IEEE Transactions on Automatic Control,* 46(8): 1318-1321, August 2001.

[McMillan, 1993] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publ., 1993.

[Moskewicz et al., 2001] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Proc. ofDAC-38,* pages 530-535. ACM, 2001.

[Muscettola *et al.,* 1998] N. Muscettola, P. P. Nayak, B. Pell, and B. Williams. Remote Agent: To boldly go where no AI system has gone before. *Artificial Intelligence,* 103(1-2):5-48, August 1998.

[Pecheur and Simmons, 2000] Charles Pecheur and Reid Simmons. From Livingstone to SMV: Formal verification for autonomous spacecrafts. In *Proc. First Goddard Workshop on Formal Approaches to Agent-Based Systems,* 2000.

[Sampath *et al.,* 1995] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control,* 40(9): 1555-1575, September 1995.

[Sampath *et al.,* 1996] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Failure diagnosis using discrete event models. *IEEE Transactions on Control Systems,* 4(2): 105-124, March 1996.

[Sampath *et al,* 1998] M. Sampath, S. Lafortune, and D. Teneketzis. Active diagnosis of discrete event systems. *IEEE Transactions on Automatic Control,* 43(7):908-929, July 1998.

[Simmons *etal,* 2001] R. Simmons, R. Goodwin, K. Zita Haigh, S. Koenig, J. CVSullivan, and M. Veloso. Xavier: Experience with a layered robot architecture. *Intelligence,* 2001.

[Williams and Nayak, 1996] B. C. Williams and P. P. Nayak. A model-based approach to reactive self-configuring systems. In *Proceedings of AAA 1-96,* 1996.

[Yung et al, 1999] B. Yang, R. Simmons, R. E. Bryant, and D.R. O'Hallaron. Optimizing symbolic model checking for contraint-rich models. In *Proc. CAV'99,* number 1633 in LNCS, pages 328-340. Springer-Verlag, 1999.