Formal Verification of Diagnosability via Symbolic Model Checking

Charles Pecheur[1] and **Alessandro Cimatti**[2]

Abstract. This paper addresses the formal verification of diagnosis systems. Given a physical system and a diagnosis system that observes it, we tackle the problem of verifying that the diagnosis system will be able to recognize the situations that it is required to recognize. In our approach, the physical system is formally modeled as a Kripke structure, while diagnosability is checked by looking for pairs of scenarios that (1) are indistinguishable based on the observable part of the system but (2) lead to situations that are required to be distinguished. We focus on the practical applicability of the method: diagnosability is recast in terms of a model checking problem, and allows for the direct use of state-of-the-art symbolic model checking techniques.

1 Introduction

Diagnosis systems are of paramount importance in many application domains, ranging from industrial plants (e.g. production, power) to transportation (e.g. railways, avionics, space). Diagnosis systems provide the ability to identify whether a certain plant, possibly operating in hazardous or unaccessible situations, is working correctly. They can help in the process of control, and prevent simple failures to stay undetected and degenerate into catastrophic events.

When diagnosis is carried out in critical domains, it becomes a critical step, and the validation of diagnosis systems is of fundamental importance. A key problem is diagnosability, i.e. the ability to ensure that no simple failure can go undetected by the diagnosis system while "preparing" for a more serious problem. In this paper we propose a new, practical approach to the verification of diagnosability. The idea is to reduce a diagnosability problem for a given plant to a problem of reachability over a *coupled twin model* of the plant, where two traces of the plant produced by the same inputs and sharing the same outputs can be compared. The reachability problem over the coupled twin model is then tackled by means of state-of-the-art symbolic model checking techniques, that allow for the analysis of finite state systems with extremely large state spaces.

As a driving application, we expect to apply these principles to diagnosis applications based on Livingstone, a model-based diagnosis system developed at NASA Ames Research Center [27]. Livingstone features discrete qualitative models that are amenable to the proposed formal analysis, and we already have a tool to convert them into SMV models [22]. We will be feeding those models to NuSMV, the new generation of the SMV symbolic model checker [6]. Although we have no experimental results at this early stage, we plan to experiment our approach on large Livingstone models of space transportation sub-systems in the coming months.

The paper is organized as follows. In section 2, we provide the context for verification of diagnosis systems. In section 3 we formalize the problem, and provide the basis of the approach. In section 4, we describe how our approach can be tackled as a model checking problem. In section 5, we describe the applicative framework that we target with our approach. Finally, section 6 reviews similar work, and section 7 draws some conclusions and outlines future lines of activity.

2 Verification of Diagnosis Systems

In this section, we lay out our assumptions on the nature of the diagnosis applications that we want to take into consideration: essentially, state estimators based on partial observations of a physical device. We then decompose the verification of such a system into different pieces and identify the piece that we want to address—namely, verification of diagnosability.

2.1 Diagnosis System

At an abstract level, we consider a diagnosis system as depicted in Figure 1. The diagnosis system connects to a feedback control loop between a *plant* and its *controller*. The inputs of the plant are the *commands* issued by the controller; its outputs are *measurements* returned back to the controller. The diagnosis system observes both the inputs and the outputs and reports a *state estimation* that seeks to track the unobservable physical state of the plant. In general, the estimation will consist of a *belief state* that covers a *set* of possible states of the plant (diagnosis might rank these states based on likelihood, but as a first approach we ignore that aspect).

Diagnosis works on an abstract view of the plant, hiding away the complex details of the physical artifacts that constitute the physical system. Hardware and software interfaces massage the information coming from the plant to provide that abstraction. From here on, we will only consider that abstract view and ignore the internals of the plant—verifying that this abstraction is indeed correctly implemented is a complex and critical task that we do not address here.

2.2 Verification of Diagnosis Systems

Consider the person or company in charge of designing a diagnosis system for a given application, and facing the need to verify and validate that design. The ultimate goal is to verify that the complete application (diagnosis system and plant) provides appropriate diagnosis in all expected operational conditions.

[1] RIACS / NASA Ames Research Center, Moffett Field, CA, U.S.A., pecheur@ptolemy.arc.nasa.gov
[2] IRST, Trento, Italy, cimatti@irst.itc.it

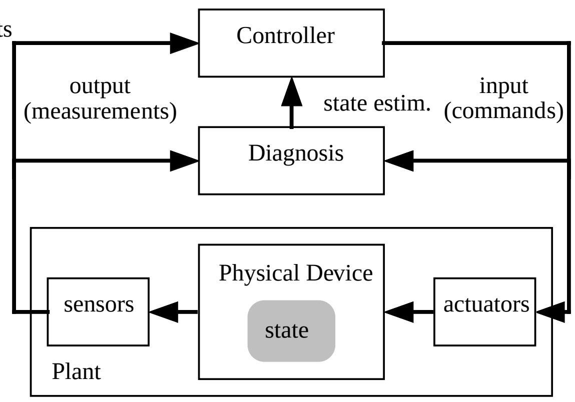

Figure 1. Architecture of a diagnosis system

Let us assume that we have a discrete model of the plant as defined above, that is used as the basis for the construction of the actual diagnosis system—either by the people building it (design), or by a tool that generates it (synthesis), or at run-time by a generic diagnosis engine (model-based reasoning). We can then decompose the global verification goal as follows:

Model Correctness Verify that the model is a valid abstraction of the actual physical plant.

Diagnosis Correctness Verify that the actual diagnosis system provides accurate diagnosis for the given model.

Diagnosability Verify that the desired diagnosis is possible given the available observations.

Assuming model correctness, diagnosis correctness checks that all that can be diagnosed is correctly diagnosed, whereas diagnosability checks that all that needs to be diagnosed can be diagnosed. In principle, if we can fulfill all three conditions, then we can guarantee that the desired diagnosis will be achieved. This paper focuses only on the diagnosability part.

3 Formalization

In this section, we develop a formal definition of diagnosis systems and their correctness criteria, that serves as a basis for argumenting the formal verification using model checking in the next setion. We successively formalize the devices we want to diagnose (*plants*), the functions that provide the estimation (*diagnosis functions*), the desired properties of those functions (*diagnosis pairs* and *diagnosability*), violations of those properties (*critical pairs*), and derived structures in which those violations are conveniently expressed (*coupled twin plants*).

3.1 Plant Models

In this paper, we will focus on diagnosis over 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 transition system, *or* plant, *is a structure $\langle X, U, Y, \delta \rangle$, where X, U, Y are respectively the* state space, input space *and* output space *of the plant, and $\delta \subseteq X \times U \times Y \times X$ is the* transition relation. *Where P is known from the context, we write $x \xrightarrow{u/y} x'$ for $(x, u, y, x') \in \delta$.*

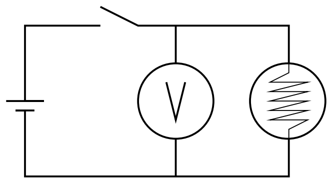

Figure 2. A simple circuit

The state x is the hidden part of the plant: only the sequences of inputs u and outputs y are observable. We assume that P covers all types of behaviours that diagnosis is expected to handle—typically, that means including faulty behaviours, with X containing faulty states. Note that, in general, P need not be deterministic. Thus, the values of y and x after the transition do may not be uniquely determined by the values of u and x before the transition. Without loss of generality, we assume that all transitions are "visible", that is, have a u and y associated to it. We are interested in sequences of consecutive states and their observable traces:

Definition 2 *Given a plant $P = \langle X, U, Y, \delta \rangle$, a* feasible execution *of P is a sequence of consecutive transitions $\sigma = ((x_{i-1}, u_i, y_i, x_i) \in \delta \mid 1 \leq i \leq k)$, which we denote as $x_0 \xrightarrow{u_1/y_1} x_1 \ldots x_{k-1} \xrightarrow{u_k/y_k} x_k$. We define Σ_P as the set of all feasible traces of P. The* trace *of σ is $w = ((u_1, y_1), \ldots, (u_k, y_k))$. We write $\sigma : x_0 \xrightarrow{w} x_k$, and $x_0 \xrightarrow{w} x_k$ if such a σ exists.*

In the following, we use the simple plant depicted in figure 2 for explanatory purposes. The plant is an electric circuit composed of a battery, a switch, a light bulb and a voltmeter. Possible commands (inputs) are: open the switch; close the switch; replace the bulb with a new one; and do nothing. It is possible to observe the light of the bulb, and the value of the voltmeter. Some components are subject to failures: the switch can become stuck, in which case it maintains its position independently of the command; the voltmeter can break down, in which case it always reports a zero value, as if the switch was open; the bulb can be burned, or short. Major problems are a fire, occurring if a short bulb is powered for three time instants, or a shock, occuring when trying to change a short bulb when powered.

3.2 Diagnosis

The goal of diagnosis is, starting from (possibly partial) knowledge of the initial state and observing the sequence of inputs and outputs, to keep updating a belief state that estimates the possible states of the plant. Ideally, diagnosis should return belief states that are as specific (i.e. small) as possible, but still cover the real state of the plant. From here on, we will only consider *correct* diagnosis, that is, diagnosis values that cover all potential current states, so that they cannot miss the actual state.

Definition 3 *A diagnosis function for a plant $P = \langle X, U, Y, \delta \rangle$ is a function $\widehat{\Delta} : 2^X \times (U \times Y)^* \rightarrow 2^X$. A diagnosis value $\hat{x} = \widehat{\Delta}(\hat{x}_0, w)$ is* correct *w.r.t. P, $\hat{x}_0$ 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 $x \in \hat{x}$. $\widehat{\Delta}$ is correct if and only if $\widehat{\Delta}(\hat{x}_0, w)$ is correct for any $\hat{x}_0$ and w.*

Note that this model assumes that the first outputs occur as the result of the first inputs. It does not consider the possibility of initial outputs that could improve diagnosis by adding information on the initial state. This would amount to considering "$(k+\frac{1}{2})$-step" executions $? \xrightarrow{?/y_0} x_0 \xrightarrow{u_1/y_1} \cdots \xrightarrow{u_k/y_k} x_k$. Our formalization should be adaptable to that alternative, although that has not been formally confirmed at this stage.

Given a plant P, we define *perfect diagnosis* Δ_P as the most specific correct diagnosis that can be made assuming full knowledge of P. Given an initial belief state $\hat{x}_0$ and trace w, $\Delta_P(\hat{x}_0, w)$ returns exactly all states that can be reached from $\hat{x}_0$ through w.

Definition 4 *The* perfect diagnosis *for a plant P is the diagnosis function $\Delta_P(\hat{x}_0, w) = \{x \mid \exists x_0 \in \hat{x}_0 . x_0 \xrightarrow{w} x\}$.*

Theorem 5 *Δ_P is correct and $\Delta_P(\hat{x}_0, w) \subseteq \widehat{\Delta}(\hat{x}_0, w)$ for any correct $\widehat{\Delta}$.*

Proof Straightforward from definition 3.

3.3 Diagnosability

Informally, a plant is diagnosable when its observable traces allow to accurately track its hidden state. In the example, we would like to be able to identify the case in which the bulb is not working correctly. However, it would be unrealistic and unnecessary to require that diagnosis provides correct exact state estimations, instantaneously and under all circumstances—that is, that $\widehat{\Delta}(\{x_0\}, w) = \{x\}$ for any $x_0 \xrightarrow{w} x$. For example, a burned bulb will stay unnoticed as long as the switch is open. Instead, the key requirement for a diagnosis system is to be able to decide between alternative conditions on the state of the system, in the context where that distinction becomes critical. Accordingly, we formalize diagnosability requirements in two parts:

- the conditions to be separated, and
- the context in which that separation is required.

Given two conditions c_1 and c_2 on the state of the plant, we consider *diagnosis conditions* $c_1 \perp c_2$, expressing that diagnosis can decide between c_1 and c_2. Typically, those conditions will have to do with the presence and characterization of faulty components in the plant. In particular, *fault detection* consists in deciding whether any fault is present ($\mathit{fault} \perp \neg \mathit{fault}$), whereas *fault separation* consists in telling different faults (or fault classes) apart ($\mathit{fault}_a \perp \mathit{fault}_b$). An example of fault detection is checking whether any component of the circuit is not working correctly, while fault separation is, for instance, deciding whether a bulb is short or burned.

Definition 6 *A* diagnosis condition *for a plant $P = \langle X, U, Y, \delta \rangle$ is a pair of subsets of states $c_1, c_2 \subseteq X$, written $c_1 \perp c_2$.*

We express context as additional conditions on the executions and initial belief states to be taken into consideration. The *execution context* is a subset Σ of the feasible executions of the plant. The *initial belief context* is modeled as a relation θ between states that can occur in a same initial belief state.

Definition 7 *A* diagnosis context *for a plant $P = \langle X, U, Y, \delta \rangle$ is a pair $C = \langle \Sigma_C, \theta_C \rangle$ such that $\Sigma_C \subseteq \Sigma_P$ and θ_C is an equivalence relation on X. $\hat{x}_0$ satisfies θ_C, written $\hat{x}_0 \models \theta_C$, iff $\hat{x}_0 \times \hat{x}_0 \subseteq \theta_C$, or equivalently, iff $(x_{01}, x_{02}) \in \theta_C$ for all $x_{01}, x_{02} \in \hat{x}_0$. $(\hat{x}_0, w) \models C$ iff $\hat{x}_0 \models \theta_C$ and there exists $\sigma \in \Sigma_C$ such that $\sigma : x_0 \xrightarrow{w} x$ and $x_0 \in \hat{x}_0$.*

A diagnosis value satisfies a diagnosis condition if it does not intersect with both sides of the condition. To put it otherwise, diagnosis fails a condition $c_1 \perp c_2$ when it reports an estimation such that both c_1 and c_2 are possible. A diagnosis function satisfies a diagnosis condition over a context if all its values in that context do. A condition is diagnosable if such a function exists.

Definition 8 *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 $\widehat{\Delta}$ satisfies $c_1 \perp c_2$ over $C = \langle \Sigma_C, \theta_C \rangle$, written $\widehat{\Delta}, C \models c_1 \perp c_2$, if and only if, for all $(\hat{x}_0, w) \models C$, we have $\widehat{\Delta}(\hat{x}_0, w) \models c_1 \perp c_2$. 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.*

The following theorem justifies the definition of "perfect diagnosis": if a diagnosis condition is diagnosable (in some context), then it is satisfied by perfect diagnosis.

Theorem 9 *A condition $c_1 \perp c_2$ is diagnosable in a plant P over a context C if and only if $\Delta_P, C \models c_1 \perp c_2$.*

Proof From Definition 8, observe that, if $\hat{x} \subseteq \hat{x}'$ and $\hat{x}' \models c_1 \perp c_2$, then $\hat{x} \models c_1 \perp c_2$. Therefore, Δ_P satisfies any diagnosable condition, since, from Theorem 5, its diagnosis values are more specific than any other correct diagnosis function. Q.E.D.

3.4 Critical Pairs

We propose to verify the diagnosability of a diagnosis condition $c_1 \perp c_2$ by checking that the plant does not have two executions with identical observable traces, one leading to c_1, one leading to c_2. We call such a pair of executions a *critical pair* for $c_1 \perp c_2$.

Definition 10 *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 $C = \langle \Sigma_C, \theta_C \rangle$, iff $\sigma_1, \sigma_2 \in \Sigma_C$ and $(x_{01}, x_{02}) \in \theta_C$.*

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. This justifies the search for critical pairs as a mean to verify diagnosability. If a critical pair exists then no correct diagnosis can satisfy that diagnosis condition (the converse is not true: an "imperfect" diagnosis may fail even in the absence of critical pairs).

This is illustrated in Figure 3, showing a critical pair $x_{01}|x_{02} \xrightarrow{w} x_1|x_2$. If diagnosis starts from an initial belief state $\hat{x}_0$ covering both initial states x_{01}, x_{02} of the pair, then the final belief state after observing w has to include both x_1 and x_2 and therefore fail $c_1 \perp c_2$.

Theorem 11 *Perfect diagnosis for a plant P satisfies $c_1 \perp c_2$ in context C, and therefore $c_1 \perp c_2$ is diagnosable over C in P, if and only if P has no critical pair for $c_1 \perp c_2$ in C.*

Proof We prove the contraposed of the theorem. There is a critical pair $\sigma_1|\sigma_2 : x_{01}|x_{02} \xrightarrow{w} x|_1 x_2$ in $C = \langle \Sigma_C, \theta_C \rangle$, if and only if $(x_{01}, x_{02}) \in \theta_C$ and $\sigma_1, \sigma_2 \in \Sigma_C$ and $x_{01} \xrightarrow{w} x_1$ and $x_{02} \xrightarrow{w} x_2$ and $x_1 \in c_1$ and $x_2 \in c_2$, if and only if $(\{x_{01}, x_{02}\}, w) \models C$ and $\Delta_P(\{x_{01}, x_{02}\}, w) \not\models c_1 \perp c_2$. Q.E.D.

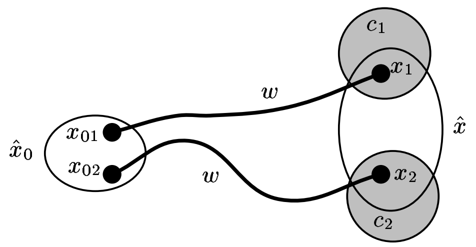

Figure 3. Critical pair

3.5 Coupled Twin Plants

We can interpret critical pairs of P as traces of a "siamese twins" plant made out of two copies of P whose inputs and outputs are forced to be identical. We call it P^2, the *coupled twin plant* of P.

Definition 12 *The* coupled twin plant *of a plant* $P = \langle X, U, Y, \delta \rangle$ *is the plant* $P^2 = \langle X^2, U, Y, \delta' \rangle$*, where* $X^2 = X \times X$ *and* $\delta' = \{((x_{01}, x_{02}), u, y, (x_1, x_2)) \mid (x_{01}, u, y, x_1), (x_{02}, u, y, x_2) \in \delta\}$.

An execution of P^2 corresponds to a pair of executions with identical traces, just like a twin pair $\sigma_1|\sigma_2 : x_{01}|x_{02} \xrightarrow{w} x_1|x_2$, and we will keep the same notation. Finding critical pairs in P becomes a standard reachability search in P^2:

Theorem 13 *A diagnosis condition* $c_1 \perp c_2$ *is diagnosable in* P *over* C *if and only if* $c_1 \times c_2$ *is not reachable in* P^2 *over* C.

Proof From Theorem 11, $c_1 \perp c_2$ is diagnosable iff P has a critical pair $\sigma_1|\sigma_2 : x_{01}|x_{02} \xrightarrow{w} x_1|x_2$ such that $x_1 \in c_1$ and $x_2 \in c_2$, which is equivalent to an execution of P^2 with $(x_1, x_2) \in c_1 \times c_2$. Q.E.D.

In the example, we can consider the case where the context restricts the number of failures in the circuit to one.

4 Diagnosability via Model Checking

In the last section, we formalized diagnosability and showed that it amounted to a reachability question in the coupled twin plant. In this section, we show how that reachability question can be phrased and efficiently answered as a model checking problem.

4.1 Kripke Structures

Model checking [10] is a formal verification technique, where the behaviour of a reactive system (e.g. a communication protocol, a hardware design) is presented as a class of transition system called a *Kripke structure*.

Definition 14 *A* Kripke structure *over a set of atomic propositions* AP *is a structure* $K = \langle S, \tau, L \rangle$*, with* $\tau \subseteq S \times S$ *and* $L : S \to 2^{AP}$. S *is the* state space *of* K*,* L *is the* labeling function *of* K *and* τ *is the* transition relation *of* K. τ *is required to be* total*:* $\forall s \in S. \exists s' \in S. (s, s') \in \tau$. *A trace of* K *is a sequence of states* $\pi = s_0 \ldots s_k$ *such that* $(s_{i-1}, s_i) \in \tau$. *Let* Π_K *be the set of all traces of* K. *When* K *is known from the context, we write* $s \to s'$ *for* $(s, s') \in \tau$ *and* $s \models p$ *for* $p \in L(s)$.

In essence, a Kripke structure is a directed graph of states with atomic propositions attached to them. Each of the states $s \in S$ represents one possible configuration of the system, while the labeling function L associates to each state the propositions holding in it. As we will see, AP is typically defined as assignments to state variables, with $L(s)$ describing the value of each variable in s. The paths of the graph represent the behaviour of the behaviour of the system over time.

Requirements over the behaviours of the system are modeled as formulae in temporal logic, for instance Computation Tree Logic (CTL) or Linear Temporal Logic (LTL) [14]. In temporal logic it is possible to predicate over the paths of the Kripke structure: for instance, the CTL formula $\mathsf{EF}\, p$ states that there exists a path (E) such that somewhere in the future (F) a state is labeled by p, while $\mathsf{E}[p \mathsf{U} q]$, read "$p$ until q", states that there exists a path such that somewhere in the future q holds, and, for all the precedings states, p holds. (For a comprehensive treatment of temporal logic see [14].)

Given a Kripke structure K, a set of initial states $Q \subset S$, and a formula ϕ, the model checking problem $K, Q \models \phi$ is to detect if the requirements expressed by the formula hold over the infinite paths/trees originating from any $s \in Q$. Model checking algorithms are based on the exhaustive exploration of the Kripke structure. When the specification is not satisfied, they are able to construct a counterexample, i.e. to produce a description of the system behaviour that does not satisfy the specification.

4.2 From Plants to Kripke Structures

A plant can be analyzed by means of model checking techniques, by defining a corresponding a Kripke structure, where it is possible to predicate over the inputs, outputs and state of the plant. We could derive a Kripke structure K_P from any plant P, but since diagnosability in P amounts to reachability in the coupled twin plant P^2, we will directly define the corresponding Kripke structure K_P. In this case, the state is defined as a vector of variables $(v_{x_1}, v_{x_2}, v_u, v_y)$, respectively ranging over X, X, U and Y, and the propositions in AP are assignments of these variables over their respective domains.

Definition 15 *Let* $P^2 = \langle X^2, U, Y, \delta' \rangle$ *be a coupled twin plant. Let* $AP_P = \{v_{x_1} = x \mid x \in X\} \cup \{v_{x_2} = x \mid x \in X\} \cup \{v_u = u \mid u \in U\} \cup \{v_y = y \mid y \in Y\}$ *be the set of atomic propositions over* P^2. P^2 *induces the Kripke structure* $K_P = \langle S_P, \tau_P, L_P \rangle$ *over* AP_P*, where* $S_P = X \times X \times U \times Y$*,* $\tau_P((x_1, x_2, u, y), (x_1', x_2', u', y'))$ *iff* $\delta'((x_1, x_2), u, y', (x_1', x_2'))$*, and* $L_P((x_1, x_2, u, y)) = \{(v_{x_1} = x_1), (v_{x_2} = x_2), (v_u = u), (v_y = y)\}$.

Figure 4 depicts the intuition underlying the definition. At each time instant, we associate inputs to the state before the transition and outputs to the state after. Actually, the previous definition is not totally appropriate, because the transition relation is not guaranteed to be total. Indeed, the executions of P^2 may be finite, even though the executions of P are infinite. This is due to the fact that two traces of P can exhibit the same output sequences only up to a certain point, and differ from there on. However, we can obtain a total model by introducing a new boolean variable $SameOutputs$, that is used to "complete" the trace being constructed when the outputs are diverging. The intuition is that the variable $SameOutputs$ should

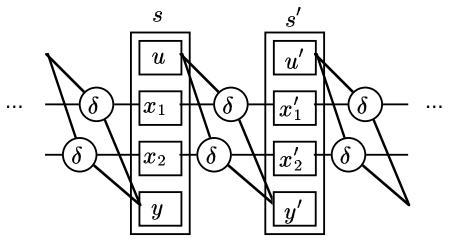

Figure 4. Transition relation as a Kripke structure

become false when the outputs of the traces differ, and stay false thereafter. We then must adjust the properties we verify to require $SameOutput$ to be true. For now we will implicitly assume that transformation, so that we can perform model checking on non-total models. We may also adjust the framework to take initial outputs into account, by decomposing $\delta(x, u, y, x')$ as $\delta_1(x, u, x') \wedge \delta_2(x', y)$, expressing that y depends on x' only. Then we can restrict S to states that satisfy δ_2 and define τ based on δ_1. In this way, the initial y will not be undetermined any more; it will be related to the initial twin internal states through δ_2.

Now, we can express a condition c on states as a formula $c(v_x)$ over assignments to the corresponding variable v_x. In particular, this applies to conditions c_1, c_2 of a diagnosis condition $c_1 \perp c_2$. In principle, notice indeed that each set $c \subseteq X$ can be characterized by the propositional formula

$$\bigvee_{x \in c} (v_x = x)$$

In practice, X itself is usually expressed as a product of variables and c is more appropriately expressed as a boolean formula over those variables. Similarly, we write $\theta(v_{x_1}, v_{x_2})$ for

$$\bigvee_{(x_1,x_2)\in\theta} (v_{x_1} = x_1 \wedge v_{x_2} = x_2)$$

Notice the close correspondence between executions $\sigma \in \Sigma_{P^2}$ and $\pi \in \Pi_{K_P}$, the difference being one extra undetermined initial y and final u in the Kripke structure. On this basis, we will assimilate a Kripke structure execution π to its plant counterpart σ.

4.3 Verification of Diagnosability

If we forget about contexts for a moment, then the problem of deciding the existence of a critical pair in P^2 for $c_1 \perp c_2$ can expressed as a model checking problem on the Kripke structure induced by P^2:

$$K_P, S_P \models \neg \mathsf{EF}\ (c_1(v_{x_1}) \wedge c_2(v_{x_2}))$$

The property states that K_P can not reach a state where c_1 holds over the first half of the state vector, and c_2 holds over the second half. If the property is violated, it is possible to produce an execution witnessing the violation (in this case, the critical pair leading to the failed diagnosis).

In order to make this useful, we need to take into account the context of the diagnosis $C = \langle \theta_C, \Sigma_C \rangle$. For the initial belief context, we can use θ_C to replace S_P in the above definition:

$$K_P, \theta_C(v_{x_1}, v_{x_2}) \models \neg \mathsf{EF}\ (c_1(v_{x_1}) \wedge c_2(v_{x_2}))$$

```
MODULE circ_type (switch_cmd, bulb_cmd)
  ...
MODULE main
VAR
  switch_cmd : { nocommand, open, close };
  bulb_cmd   : { nocommand, replace };
  c1 : circ_type(switch_cmd, bulb_cmd);
  c2 : circ_type(switch_cmd, bulb_cmd);
DEFINE SameOut := (c1.light = c2.light) & ...
VAR SameOutputs : boolean;
ASSIGN
  init(SameOutputs) := SameOut;
  next(SameOutputs) :=
    case
      !SameOutputs : 0;
       SameOutputs : next(SameOut);
    esac;
```

Figure 5. The model of the twin circuit

thus restricting the set of initial states of the model checking problem. Similarly, in order to take into account Σ_C, the model checking problem that we consider is

$$K_P, \theta_C(v_{x_1}, v_{x_2}) \models_{\Sigma_C} \neg \mathsf{EF}\ (c_1(v_{x_1}) \wedge c_2(v_{x_2}))$$

where the restricted entailment $\models_{\Sigma_C}$ limits the scope of the E operator to the paths that satisfy the Σ_C condition. There are several ways in which this can be implemented in practice, depending on the characteristics of Σ_C. In general, if we have an LTL property $\Sigma_C(v_x, v_u, v_y)$ that characterizes the traces in Σ_C, then, diagnosability in context C amounts to the following LTL model checking problem:

$$K_P \models (\theta_C(v_{x_1}, v_{x_2}) \wedge \Sigma_C(v_x, v_u, v_y)) \Rightarrow \neg \mathsf{F}\ (c_1(v_{x_1}) \wedge c_2(v_{x_2}))$$

that can be tackled by means of the standard tableau construction reported in [9]. If Σ_C is a simple set of conditions on the inputs and state of the plant, it is possible to restrict the entailment by composing the Kripke structure with additional conditions (e.g. on the transition relation).

4.4 Symbolic Model Checking

Symbolic model checking [19] is a particular form of model checking, where the behaviour of the model is encoded in the boolean space. Propositional formulae are used for the compact representation of models, and transformations over propositional formulae provide a basis for efficient exploration. This allows for the analysis of extremely large systems [5]. As a result, symbolic model checking is routinely applied in industrial hardware design, and is taking up in other application domains (see [11] for a survey). Symbolic Model Checking is traditionally implemented by means of Binary Decision Diagrams (BDDs) [4]. SMV [19] was the first symbolic model checker based on BDDs. More recently, the use of efficient propositional satisfiability (SAT) techniques has been proposed [2]. SAT-based model checking is currently enjoying a substantial success in several industrial fields (see, e.g., [13], but also [3]), and opens up new research directions.

Within the scope of our activity, we concentrate on NUSMV [7], a symbolic model checker originated from the reengineering, reimplementation and extension of CMU SMV. The NUSMV project aims

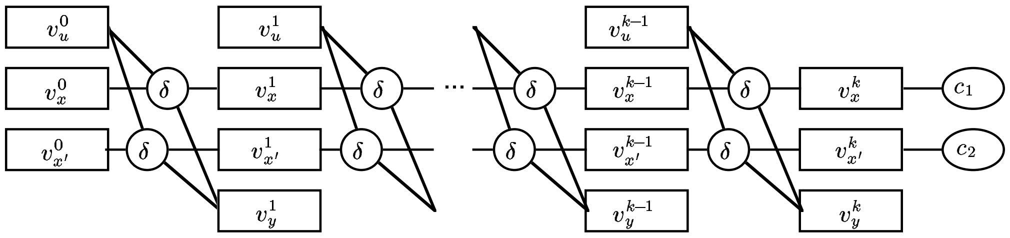

Figure 6. Constraint network for verifying diagnosability conditions

at the development of a state-of-the-art symbolic model checker, designed to be applicable in technology transfer projects: it is a well structured, open, flexible and documented platform for model checking, and is robust and close to industrial systems standards [6]. NuSMV is able to process the description of a Kripke structure, written in an extension of the SMV language: the system can be characterized by means of synchronous and asynchronous composition of modules. The state of the system is characterized by discrete (boolean and scalar) variables, the evolution of which is defined by means of propositional constraints and/or (possibly nondeterministic) assignments.

The NuSMV model for the coupled twin plant for the circuit is outlined in figure 5. The first module statement defines a (single) plant in form of a module type named `circ_type`, with two parameters representing the commands to the switch and to the bulb. The detailed definition of the circuit dynamics is omitted for lack of space. In the main module, the circuit is instantiated twice, generating `circ1` and `circ2`. Notice that the same command variables (`switch_cmd` and `bulb_cmd`) are given in input to both modules. Then, we express the constraint that both instances of the circuit must exhibit the same behaviour as follows. We first name as `SameOut` the condition that the observables in the two circuit instances (light and voltmeter reading) are the same. Then, we define the `SameOutputs` boolean variable, that is initially true on the states where `SameOut`; the `next` assignment, based on a case switch, defines the value of `SameOutputs` depending on its previous value and on the value of `SameOut`. In particular, if it was false then it stays false (boolean value `0`), otherwise it assumes the value of `SameOut`. It is then possible to use `SameOutputs` in the properties being checked, to require then that the critical pairs being looked have the same outputs.

In the case of the circuit, it is possible to define different properties and diagnosability problems. The basic building blocks of the properties are propositional conditions over states, that can be expressed in NuSMV by means of the `DEFINE` construct. For instance, it is possible to express that the number of failures within a module as

```
DEFINE failures :=
  battery.failure + bulb.failure + ...
```

where the sum allows to count how many boolean predicates are true (i.e. have value `1`). Then, in the main module it is possible to constaint the number of failures, e.g. by specifying that at most one failure can occur in the first instance of the circuit, as follows:

```
DEFINE single_failure := (c1.failures <= 1)
```

The catastrophic fire condition, occurring when a short is powered for three time instants in a row, can not be expressed as a condition on the state. We encode such a condition by introducing two history variables, whose value represents the fact that a short powered occurred at one or two time instants ago, as follows:

```
VAR
  short_powered : boolean;
  short_powered_1 : boolean;
  short_powered_1_1 : boolean;
ASSIGN
  short_powered :=
    (failure = short) & power_in;
  init(short_powered_1) := 0;
  init(short_powered_1_1) := 0;
  next(short_powered_1) := short_powered;
  next(short_powered_1_1) := short_powered_1;
DEFINE catastrophe_fire := short_powered &
  short_powered_1 & short_powered_1_1;
```

As an example of fault detection property, we check if it is possible that a fire occurs without being detected. The answer is that this is not possible under the single failure assumption, while it is possible to have an undetectable problem if we allow for a double failure condition. In particular, the model checker finds this possible in a situation where in one of the two instances, the switch is open and no failure is presend, while in the other the switch is closed, the bulb is short and the meter is broken.

In terms of verification capabalities, NuSMV provides an effective integration of BDD-based and SAT-based model checking. (In the case of the circuit, the analysis described above is carried out in fractions of a second.) This allows for an important widening of its spectrum of applicability. Indeed, BDD-based and SAT-based model checking are often able to solve different classes of problems, and can therefore be seen as complementary techniques. In the specific case of diagnosability, a SAT-based analysis of the problem amounts to checking a network of propositional constraints in the following variables

$$\begin{array}{cccc}
v_x^0 & v_{x'}^0 & v_u^0 & \\
v_x^1 & v_{x'}^1 & v_u^1 & v_y^1 \\
\dots & \dots & \dots & \dots \\
v_x^{k-1} & v_{x'}^{k-1} & v_u^{k-1} & v_y^{k-1} \\
v_x^k & v_{x'}^k & & v_y^k
\end{array}$$

where v^i captures the value of v at step i. Note that we drop v_y^0 and v_u^k, whose value is unconstrained. The constraints are

$$\begin{array}{rcl}
\delta(v_x^0, v_u^0, v_y^1, v_x^1) & \wedge & \delta(v_{x'}^0, v_u^0, v_y^1, v_{x'}^1) \\
\wedge \dots & \wedge & \dots \\
\wedge\, \delta(v_x^{k-1}, v_u^{k-1}, v_y^k, v_x^k) & \wedge & \delta(v_{x'}^{k-1}, v_u^{k-1}, v_y^k, v_{x'}^k) \\
\wedge\, c(v_x^k) & \wedge & c'(v_{x'}^k)
\end{array}$$

The intuition is pictured in Figure 6. Basically, the existence of a critical pair corresponds to the existence of a model to the set of propositonal constraints.

5 Target applicative framework

In this section, we outline the applicative framework that we target with our approach. We focus on the Livingstone framework, and we show how the models can be reduced to SMV models, that can then be tackled with the NuSMV system.

5.1 Livingstone

Livingstone is a model-based health monitoring system developed at NASA Ames [27]. It uses a symbolic, qualitative model of a physical system, such as a spacecraft, to infer its state and diagnose faults.[3] 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.[4] Remote Agent was demonstrated in flight on the Deep Space One mission (DS-1) in May 1999, marking the first control of an operational spacecraft by AI software [21]. Livingstone is also used in other applications such as the control of a propellant production plant for Mars missions [8], the monitoring of a mobile robot [26], and intelligent vehicle health management (IVHM) for experimental space transportation vehicles (X-34, X-37, next-generation space shuttle).

Livingstone peruses a model that describes the normal and abnormal functional modes of each component in the system. The model defines observable and hidden variables (here called "attributes"). Livingstone models are discrete and finite, and are meant to be qualitative: continuous physical domains have to be abstracted into discrete intervals such as {`low`, `nominal`, `high`} or {`neg`, `zero`, `pos`}. Each component has a "mode" variable identifying its nominal and fault modes (with fault probabilities). 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.

5.2 Verification of Livingstone Models

Livingstone models define synchronous transitions systems and correspond exactly to the plant models defined in section 3.1. All attributes have a small finite range, so the models are amenable to finite model checking techniques. For that purpose, Livingstone models need to be translated into the input formalism of the model checker—SMV in our case.

Pecheur and Simmons have developed a translator to automate the conversion from Livingstone to SMV [22]. The translator supports three kinds of translation, as shown in Fig. 7:

- The Livingstone model is translated into an SMV model amenable to model checking.
- The specifications to be verified against this model are expressed in terms of the Livingstone model and similarly translated.

[3] Livingstone also has a recovery part, that can suggest an action to recover a given goal configuration. We are not concerned with that part here.

[4] The two other components are the Planner/Scheduler [20], which generates flexible sequences of tasks for achieving mission-level goals, and the Smart Executive [23], which commands spacecraft systems to achieve those tasks.

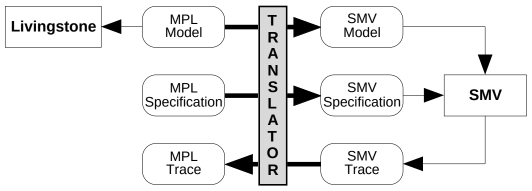

Figure 7. Translation between MPL and SMV

- Finally, the diagnostic traces produced by SMV are converted back in terms of the Livingstone model.[5]

The original translator applied to an earlier, Lisp-style syntax for Livingstone models. The translator has recently been upgraded to the current Java-like syntax (called JMPL) by Reid Simmons. It is written in Java.

The translator has been successfully applied to several Livingstone models, such as DS-1 [21], the Xavier mobile robot [26] and the In-Situ Propellant Production system (ISPP) [8, 15]. The ISPP experience was the most extensive; it did produce useful feedback to the Livingstone model developers, but also experimented with the potentials and challenges of putting such a tool in the hands of application practitioners. Models of up to 10^{55} states could still be processed in a matter of minutes with an enhanced version of SMV [28]. Experience shows that Livingstone models tend to feature a huge state space but little depth, for which the symbolic processing of SMV is very appropriate.

6 Related Work

The idea of diagnosability has received a lot of attention in the framework of DES. In [24, 25], 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 [17] 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 [16].

Console, Picardi and Ribaudo [12] propose the use of a particular form of process algebras, PEPA, for the formalization and the analysis of diagnosis and diagnosability problems.

In terms of expressivity, our work shares several underlying assumptions with [24, 25]. In particular, our approach considers failures that can be represented as reachability conditions. 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 [16]. Furthermore, our approach preserves the semantics of the problem, thus making it possible to tune the decision procedure to the application domain.

The work in [18] addresses the problem of designing the controller taking into account the issues of diagnosability (active diagnosis).

[5] This part is not completed yet.

Similar problems are also addressed in planning under partial observability [1], 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.

7 Conclusions and Future Work

In this paper, we have proposed a new, practical approach to the problem of diagnosability. We reduce the problem of diagnosability to the problem of searching for critical pairs in a coupled twin plant. This problem can be encoded into a model checking problem, that can be tackled by means of state-of-the-art symbolic model checking techniques. A practical framework for the application of the approach is also presented.

In the future, we will consider a generalization of our approach to the case of more complex failures, described in terms of logical specifications. In particular, we will try to take into account the fact that diagnosis can propose several candidates, with different degrees of likelihood. Furthermore, we will apply the approach to different real-world Livingstone models, both with BDD-based and SAT-based model checking, to evaluate the practical applicability of the technique. 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

[1] P. Bertoli, A. Cimatti, J. Slaney, and S. Thiebaux, 'Solving power supply restoration problems with planning via symbolic model checking', in *Proceedings of the European Conference on Artificial Intelligence*, Lyon, France, (August 2002).

[2] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, 'Symbolic model checking without BDDs', in *Proceedings of the Fifth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*, ed., R. Cleaveland, volume 1579 of *LNCS*, pp. 193–207. Springer-Verlag, (1999).

[3] A. Borälv, 'A Fully Automated Approach for Proving Safety Properties in Interlocking Software Using Automatic Theorem-Proving', in *Proceedings of the Second International ERCIM Workshop on Formal Methods for Industrial Critical Systems*, eds., S. Gnesi and D. Latella, Pisa, Italy, (July 1997).

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

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang, 'Symbolic Model Checking: 10^{20} States and Beyond', *Information and Computation*, **98**(2), 142–170, (June 1992).

[6] A. Cimatti, E.M. Clarke, F. Giunchiglia, and M. Roveri, 'NuSMV: a new Symbolic Model Verifier', in *Proceedings Eleventh Conference on Computer-Aided Verification (CAV'99)*, eds., N. Halbwachs and D. Peled, number 1633 in Lecture Notes in Computer Science, pp. 495–499, Trento, Italy, (July 1999). Springer-Verlag.

[7] A. Cimatti, E. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, 'Integrating BDD-based and SAT-based symbolic model checking', in *Proceedings of the Fourth International Workshop on Frontiers of Combining Systems*, ed., A. Armando, volume 2309 of *LNAI*, pp. 49–56. Springer-Verlag, (April 2002).

[8] 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).

[9] E. Clarke, O. Grumberg, and K. Hamaguchi, 'Another Look at LTL Model Checking', *Formal Methods in System Design*, **10**(1), 57–71, (February 1997).

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

[11] E. M. Clarke and J. M. Wing, 'Formal methods: State of the art and future directions', *ACM Computing Surveys*, **28**(4), 626–643, (December 1996).

[12] L. Console, C. Picardi, and M. Ribaudo, 'Diagnosis and diagnosability using pepa', in *Proceedings of the European Conference on Artificial Intelligence*, pp. 131–136, Berlino, Germany, (August 2000). IOS Press.

[13] Fady Copty, Limor Fix, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Vardi, 'Benefits of bounded model checking at an industrial setting', in *Proceedings of CAV 2001*, pp. 436–453, (2001).

[14] E. A. Emerson, 'Temporal and modal logic', in *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, ed., J. van Leeuwen, chapter 16, 995–1072, Elsevier, (1990).

[15] Peter Engrand, 'Model checking autonomy models for a martian propellant production plant', in *Proceedings of the AAAI Symposium on Model-Based Validatin of Intelligence*, (March 2001).

[16] 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).

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

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

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

[20] N. Muscettola, 'HSTS: Integrating planning and scheduling', in *Intelligent Scheduling*, eds., Mark Fox and Monte Zweben. Morgan Kaufman, (1994).

[21] 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).

[22] Charles Pecheur and Reid Simmons, 'From Livingstone to SMV: Formal verification for autonomous spacecrafts', in *Proceedings of First Goddard Workshop on Formal Approaches to Agent-Based Systems*, (April 2000). To appear in Lecture Notes in Computer Science, Springer Verlag.

[23] B. Pell, D. E. Bernard, S. A. Chien, E. Gat, N. Muscettola, P. P. Nayak, M. Wagner, and B. C. Williams, 'An autonomous spacecraft agent prototype', *Autonomous Robots*, **5**(1), (March 1998).

[24] 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).

[25] 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).

[26] Reid G. Simmons, Richard Goodwin, Karen Zita Haigh, Sven Koenig, Joseph O'Sullivan, and Manuela M. Veloso, 'Xavier: Experience with a layered robot architecture', *Intelligence*, (2001). to appear.

[27] B. C. Williams and P. P. Nayak, 'A model-based approach to reactive self-configuring systems', in *Proceedings of AAAI-96*, (1996).

[28] B. Yang, R. Simmons, R. Bryant, , and D. O'Hallaron, 'Optimizing symbolic model checking for invariant-rich models', in *Proc. of International Conference on Computer-Aided Verification*, (1999).