System health tracking and safe testing

A. Bos[a] and A.J.C. van Gemund[a] and J. Zutt[b]

[a] S&T BV and Delft University of Technology, Delft, The Netherlands
[b] Delft University of Technology, Delft, The Netherlands

Keywords: Model-based diagnosis, health tracking, safe testing, test vector generation

ABSTRACT

As the health situation of a system is only indirectly accessible, often conclusive explanations for observed abnormal behavior can not be given. In order to discriminate further between possible diagnoses, more information about system behavior is necessary. Testing techniques are especially useful in situations where it is not possible to probe additional process variables, such as in remote diagnosis applications. However as Scarl[1] has pointed out, care must be taken as test vectors may induce new errors. He introduced the notion of so-called hazard condition constraints that should not be violated by the test input.

In this paper, we apply the notion of safe test vector generation to the domain of *dynamic* systems. Dynamic systems are characterized by the fact that the current behavior does not depend on the current input only, but also on the history of the system. Therefore, safe testing for dynamic systems needs a technique akin to model-predictive control. That is, before one can say that a particular test vector will discriminate between two possible diagnoses, or that it will not violate a hazard condition, the behavior of the system has to be simulated over a number of time steps.

1. INTRODUCTION

As the actual health situation is only partially and indirectly observable, in general a diagnosis system will come up with multiple possible explanations for observed behavior. In order to pin point the exact cause, more information about system behavior is necessary. One technique is to probe additional process variables, *i.e.*, incremental diagnosis. Incremental diagnosis is only possible when there is easy access to the variables. In many diagnostic applications, however, incremental probing is not really possible. For example, remote diagnosis of satellite instruments uses all the telemetry data available. If more information about the health state is needed, one needs to explore an alternative way, *e.g.*, feeding the system with test input such that it is forced to behave in an observable different way for each of the possible health states.

Struss[2] has given an elegant description of a theory of test generation. The core of his theory is that an input vector should discriminate between the possible behavioral modes. That is, the input vector should be a *hitting set* of the family of sets representing the inputs that discriminate between the behavioral modes. However, care must be taken, as Scarl[1] has shown, to choose test vectors carefully. As the structure of the system might be changed due to a fault, certain inputs may induce new errors. Scarl introduced the concept of so-called hazard condition constraints that should not be violated by the system reacting on the test input.

In this paper, we investigate how the concept of *safe testing*, as introduced by Struss and Scarl, can be applied to the domain of *dynamic systems*. As the behavior of a dynamic system may evolve even without a change on the input, discrimination between possible diagnoses and a possible violation of safety constraints may occur *after* a number of time steps. This means that generation of appropriate test vectors involves simulation over a number of time steps.

The main structure of the paper is as follows: We start with a description of the diagnostic process of a satellite instrument where test vectors help to discriminate between possible diagnoses and also where hazard conditions constraints are essential to guarantee safe testing. Next, we summarize the framework of diagnosis of dynamic systems as introduced by Williams and Nayak[3] in their article on the Livingstone system for model-based diagnosis. This is followed by a description of the process of generating test vectors and checking hazard condition constraints, respectively, for dynamic systems.

Contact: A. Bos; `bos@science-and-technology.nl`; Phone: +31 15 2787794; Fax: +31 15 2786697; `http://www.science-and-technology.nl`; P.O.Box 608; NL-2600 AP Delft, The Netherlands

2. MOTIVATING EXAMPLE

Our example system is the GOME instrument[4] for which we developed a model-based diagnosis system. The instrument, GOME (Global Ozone Measuring Experiment), measures Ozone concentrations by comparing direct sun light and sun light that has traveled through the earth's atmosphere. Basically, GOME is an accurate spectrometer and Figure 1 shows the most relevant parts of the instrument. Light from the earth's atmosphere enters the instrument via the shutter, and

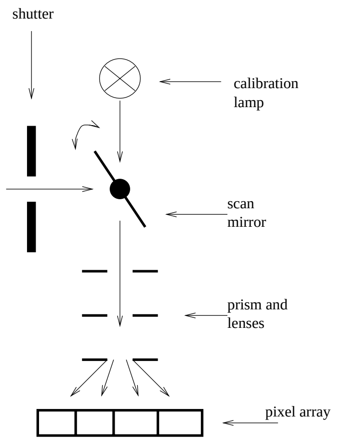

Figure 1: Example satellite instrument

is being reflected via the scan mirror and the prism and lens section onto the pixel array. The prism and lens section separate the light so that each of the pixels measures a wavelength region. The scan mirror can be set in different bias positions along which the mirror can rotate such that the instrument can scan different regions of the earth. Every now and then, the instrument must be re-calibrated by lighting a spectral calibration lamp (with distinct spectral features). During re-calibration, the shutter must be closed and the mirror must be set in the so-called calibration position.

Multiple explanations The diagnosis system for the GOME-instrument found a number deviations from expected behavior. As only a limited amount of information is available, non-conclusive results may result. For example, consider the situation where no significant ouput is read from the pixel array, while we commanded the shutter to be open, and the mirror is in such a position that light from the outside will be reflected on the pixel array. One of the explanations is that the shutter is stuck closed so that no light enters the instrument. Another explanation is that the mirror is stuck in such a position that no sufficient light is reflected on the pixel array, and yet another is that the pixel array is faulted. The prism and lens section are less likely candidates to fail. By trying to command the mirror to the calibration position and lighting the calibration lamp, we can verify the workings of the pixel array itself (that is, if output is read from the pixel array we gain in confidence that the pixel array is working correctly). This example shows that additional input may help to discriminate between possible explanations for observed behavior.

Safe testing Simply generating test vectors may lead to dangerous situations as the following example shows. During the operation of GOME we found that due to some error in the command processor of the instrument, a particular sequence of commands is not executed correctly. That is, the command processor can be brought into such a state that the mirror was not set in its commanded bias position, instead the mirror remained in one of its extreme positions. If now the scan command was given (either by hand or by an automatic test vector generator) so that the mirror would have start rotating, the mirror would have hit the instrument's chassis and most likely destroyed itself as no protection with limit switches is present.

Thus, we conclude that test vector generation should be done extremely careful as it might induce new and possibly fatal errors.

3. FRAMEWORK

3.1. Diagnosis of dynamic systems

We follow the framework[3] of Nayak and Williams That is, we concentrate on situations where the target System Under Scrutiny (SUS) can be modeled as a timed (Moore) transition system as is shown in Figure 2. The state governs the

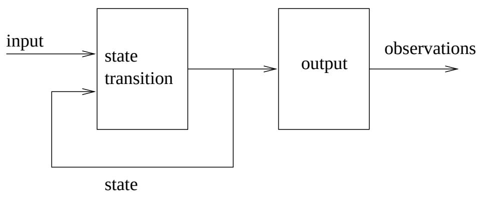

Figure 2: Dynamic system

behavior of the system over time but is not directly observable. The state is only revealed to the observer via the output function. The behavior of a transition system is described by the notion of a *trajectory*. A trajectory of a transition system is a sequence $(s_0, s_1, s_2, \ldots)$ of states. In this paper, the state transition function models *normal* behavior of the system as well as *faulty* behavior. Thus, a state transition $s_i \to s_{i+1}$ may indicate nominal behavior as well as a component failure. In order to model component failures, the state vector of the transition system also includes so-called *health mode* values for each component of the system. A health mode value describes the physical condition of a component, *e.g.*, mirror is working normally, mirror is stuck, or –to indicate a non-anticipated health mode– mirror is working abnormally.

A transition system is defined by a set of finite-domain constraints and the $\bigcirc$ operator. A finite-domain constraint is either a simple equation of the form `v1` $=$ `V` or `v1` $=$ `v2`, where `v1` and `v2` are finite-domain variables and `V` is an element of the finite-domain of `v1`, or a Boolean combination of these simple equations. Variables are used to describe the input, output, and state of the system. Symbol $\bigcirc$ is the so-called next-operator and is used denote truth in the next state of the trajectory. The constraints have two purposes:

- To define the state transitions, using the so-called *inter-state* constraints. A possible state transition τ is defined using a conjunction of transition expressions, each of the form

  $$\Phi_l \supset \bigcirc\Psi_l, \tag{1}$$

  where Φ_l and Ψ_l are finite-domain constraints. The variables in Φ_l may be input and state variables; those in Ψ_l are restricted to state variables.

  Let $(\ldots, s_i, s_{i+1}, \ldots)$ be a trajectory of a transition system for which Constraint 1 holds. Then, if in state s_i it is known that Φ_l holds, then Ψ_l holds in state s_{i+1}.

- To define the output function, using the so-called *intra-state* constraints. The relationship between the actual state and the observations is defined using a set of finite-domain constraints.

The set Π of variables of a transition system will be divided into (i) input Π_u (or control), (ii) output Π_o, and (iii) state (and health) variables Π_s. State variables are not observable; input and output variables are.

Example 1: The scan mirror of the GOME instrument can be modeled using a transition system consisting of three variables, *i.e.*, `bpos`, `scan`, and `cmnd`, where `bpos` represents the bias position of the mirror, `scan` the scanning mode of the mirror, and `cmnd` the input command. There are only a finite number of bias positions possible, that will be denoted by `PI`, where $1 \leq$ `I` $\leq$ `N`, and the corresponding command to put the mirror in the bias position is `To-pI`. The `Sscan` value starts the scanning process of the mirror. Thus, we have the following inter-state constraints:

$$\texttt{cmnd=To-pI} \supset \bigcirc(\texttt{bpos=PI} \wedge \texttt{scan=No}),$$
$$\texttt{cmnd=Sscan} \wedge \texttt{bpos=PI} \supset \bigcirc(\texttt{scan=Yes} \wedge \texttt{bpos=PI}).$$

Assume furthermore that we also like to describe the change (in time) of the electrical current that feeds the mirror motor. The variable `di` will be used to denote the absolute value of current change, and (`di=0`) represents a constant current and (`di!=0`) a changing current. The following intra-state constraint can be used to encode the feasible states with respect to motor current and scan state:

$$\texttt{scan=No} \supset \texttt{di=0}, \text{ and,}$$
$$\texttt{scan=Yes} \supset \texttt{di!=0}.$$

□

As we already remarked, the state of the system is not directly observable, it can only be observed via a set of intra-state constraints that link state variables to output variables. To formalize this, we introduce the observation function Obs mapping a state s onto a "restricted" state $s^o = Obs(s)$ such that s^o assigns a value to output variables only. If $\sigma = (s_0, s_1, s_2, \ldots)$ is a trajectory, then the *observed trajectory* is denoted by $Obs(\sigma) = (Obs(s_0), Obs(s_1), Obs(s_2), \ldots)$. Note that two different trajectories may lead to the same observed trajectory.

We further assume that the description of the overall SUS is composed out of a collection of transition systems corresponding to the system's components together with a description of how these components are connected (by equating variables of different components). Each of the component's transition system defines how the component may change state. So, each conjunct of an overall transition $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \bigcirc\Psi_{kl}$ describes a possible state transition of one of the system's component. Furthermore, we assume that each of the transition systems changes state synchronously, meaning that on each time tick all the subtransition systems make a local transition. The overall system description is denoted by SD.

In the sequel, we will use the following sets of variable-value pairs: For a point in time i, we let U_i represent the set of input (or control) variable-value pairs, S_i the set of state variable-value pairs, and $O_i = Obs(S_i)$ the set of output variable-value pairs. Usually, we will call a set (say U) the current set if its index is i (*i.e.,* U_i), the previous set if its index is $i-1$ (*i.e.,* U_{i-1}), and the next set if its index is $i+1$ (*i.e.,* U_{i+1}).

The diagnosis task is to derive the actual state S_i –including the health values– from the inputs U_{i-1}, the observations O_i, and the previous behavior (as we are dealing with so-called Moore transition systems we only need to remember the previous state S_{i-1}). Although nominal behavior of our transition system may be assumed to be deterministic, due to possible failures and only partially and indirectly observable state information, actual state determination will in general not be conclusive. That is, a transition in the trajectory may be a nominal one, or may be caused by a failure in one or more of the system's components.

The task of the diagnosis system (or state/mode identifier) is to determine the trajectory $\sigma = (s_0, s_1, s_2, \ldots)$ belonging to the observed trajectory $Obs(\sigma)$. The state identification task can now be defined as follows, where implicitly is assumed that the starting state S_0 is known.

DEFINITION 3.1 (STATE/MODE IDENTIFICATION). *Given a system with system description SD, set of previous inputs U_{i-1}, set of current outputs O_i, and a previous state S_{i-1}, the state identification task is to find a transition $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \bigcirc\Psi_{kl}$ such that*

$$\left\{ \bigwedge_l \Psi_{kl} \right\} \cup \mathit{SD} \cup O_i \text{ is consistent,}$$

where

$$\mathit{SD} \cup S_{i-1} \cup U_{i-1} \models \Phi_{kl}.$$

The current state S_i is a set of variable-value pairs that is consistent with $\bigwedge_l \Psi_{kl}$. In general, and especially during fault situations, multiple candidates for the state S_i are possible. Often, as a heuristic, likelihood information is assigned to each possible candidate for S_i. The likelihood is based[3, 5] on the a-priori probability $p(\tau_k)$ that the SUS will select a transition τ_k given that the SUS is in state S_{i-1}, and on the number of (output) signals being "explained". That is, the

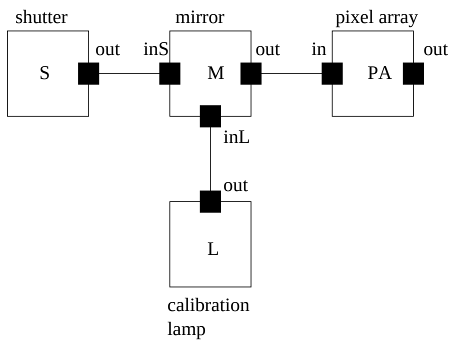

Figure 3: Simplified schema of the GOME instrument

probability $p(\tau_k \mid O_i, S_{i-1})$ that transition τ_k is selected, given the current observations O_i and previous state S_{i-1} is determined by

$$\begin{aligned} p(\tau_k \mid O_i, S_{i-1}) &= \frac{p(O_i, S_{i-1} \mid \tau_k)p(\tau_k)}{p(O_i, S_{i-1})} \\ &\propto p(O_i, S_{i-1} \mid \tau_k)p(\tau_k), \end{aligned}$$

where $p(O_i, S_{i-1} \mid \tau)$ can be estimated as follows: Given a transition $\tau_k \equiv \Phi_{kl} \supset \bigcirc\Psi_{kl}$, for which SD $\cup\, S_{i-1} \cup U_{i-1} \models \Phi_{kl}$, then:

- $p(O_i, S_{i-1} \mid \tau_k) = 0$, if $\{\bigwedge_l \Psi_{kl}\} \cup$ SD $\cup\, O_i$ is inconsistent;
- $p(O_i, S_{i-1} \mid \tau_k) = 1$, if $\{\bigwedge_l \Psi_{kl}\} \cup$ SD $\models O_i$;
- $0 < p(O_i, S_{i-1} \mid \tau_k) < 1$, if neither of the two following cases hold. The actual probability value used can for example be estimated by incorporating[3,5] the number of output variables that logically follow from $\bigwedge_l \Psi_{kl} \cup$ SD.

If multiple candidates with comparable likelihood remain, we say that the observed trajectory is ambiguous.

Of special interest are trajectories $(s_0, \ldots, s_m, \ldots, s_{m+n}, \ldots)$ where during the state range $(s_m, \ldots, s_{m+n})$ no new input is applied to the system, *i.e.*, for all $m \leq i \leq m+n$, it holds that the set U_i of input signals is empty. For such a part $(s_m, \ldots, s_{m+n})$ of the trajectory, we say that the behavior of the transition system is *periodic* iff there are i and j, with $0 \leq i < j \leq n$, such that $s_{m+i} = s_{m+j}$. A special case of a sub-trajectory, during which no new input is applied, is when $s_{i+1} = s_i$. We say that in such a case, the transition system is in steady-state.

3.2. Gome example

Let us consider the GOME system again. A further simplified component description is given in Figure 3. The four main components are the shutter (S), the scan mirror (M), the calibration lamp (L), and the pixel array (PA). We concentrate, for the moment, at how sun light is processed by the different components. We use the following abstraction for the various process variables that carry light information: Each of these variables has as domain the set $\{\texttt{L}, \texttt{NL}\}$ representing the presence of light and no ligth, respectively. The shutter has two "normal" state values (variable `sS`), *i.e.*, `Closed` or `Open`, and two faulted state values, `Stuck-closed` and `Stuck-open`. This leads, among other constraints, to the following inter-state constraints:

$$\texttt{cmnds=Open} \wedge (\texttt{sS=Closed} \vee \texttt{sS=Open}) \supset \bigcirc\texttt{sS=Open},$$
$$\texttt{cmnds=Open} \wedge \texttt{sS=Stuck-closed} \supset \bigcirc\texttt{sS=Stuck-closed},$$
$$\ldots,$$

where cmnds is an input command to open (Open) or to close (Close) the shutter. The intra-state constraints are:

$$(\texttt{sS=Open} \vee \texttt{sS=Stuck-open}) \supset \texttt{out=L},$$
$$(\texttt{sS=Closed} \vee \texttt{sS=Stuck-closed}) \supset \texttt{out=NL}.$$

The inter-state constraints of the scan mirror were already given in Example 1. We will now only give the intra-state constraints with respect to the light information. Light coming from the shutter is reflected iff the bias position of the mirror is one of P1,...,PN-1, and light from the calibration lamp is reflected iff the mirror's bias position is PN:

$$(\texttt{bpos=P1} \vee \cdots \vee \texttt{bpos=PN-1}) \supset \texttt{out=inS},$$
$$\texttt{bpos=PN} \supset \texttt{out=inL}.$$

The domain of the state sL of the calibration lamp includes On to indicate that the lamp is emitting light, Off to indicate the lamp is off, and Dead to indicate a broken lamp. However, switching the lamp on, using command cmndL=On, doesn't immediately lead to a burning lamp. It may take an extra one or two time steps for the lamp actually to emit light. Therefore, we introduce two extra states On1 and On2 (for which no intra-state constraints with respect to light behavior can be given):

$$\texttt{sL=Off} \wedge \texttt{cmndL=On} \supset \bigcirc\texttt{sL=On1},$$
$$\texttt{sL=On1} \supset \bigcirc\texttt{sL=On2},$$
$$\texttt{sL=On2} \supset \bigcirc\texttt{sL=On},$$
$$\texttt{cmndL=Off} \supset \bigcirc\texttt{sL=Off},$$
$$\texttt{sl=Dead} \supset \bigcirc\texttt{sL=Dead}.$$

The intra-state constraints are:

$$\texttt{sl=Off} \supset \texttt{out=NL},$$
$$\texttt{sl=On} \supset \texttt{out=L},$$
$$\texttt{sl=Dead} \supset \texttt{out=NL}.$$

For this example we assume that the pixel array can be described using a static model, and the state is only used to decode the health state of the system. The output reading out of the pixel array is High if the sum of all its pixels exceeds a certain limit value and is Low if the sum is below that limit value:

$$\texttt{pa-health=ok} \supset \left(\begin{array}{l} \texttt{in=L} \supset \texttt{out=High} \\ \texttt{in=NL} \supset \texttt{out=Low} \end{array} \right).$$

Consider now the situation where the shutter close command (S.cmnds=Open) is given as input, in the previous state the mirror bias position is P1 (M.bpos=P1), and the pixel array gave low output (PA.out=Low). One of the likely current states S_i will contain sS=Stuck-closed as an explanation for observed behavior, and another will contain $\neg$(pa-health=ok). Without extra information, it is hard to say which one is correct.

4. TESTING

As described earlier, diagnosis with a limited and fixed set of sensors easily leads to multiple possible and likely explanations for observed behavior. To further discriminate within the set of likely candidates, it is possible to feed the system with additional test vectors. The general idea is given in Figure 4. Assume that the trajectory until state s_i can be estimated unambiguously. At s_i, however, it is not possible to infer a single next state. For example, we may come to the conclusion that either the pixel array is dead, or the shutter is stuck-closed. In such a case, additional input signals may discriminate between the possible state values for s_{i+1} as the new input may lead to two observably distinct trajectories.

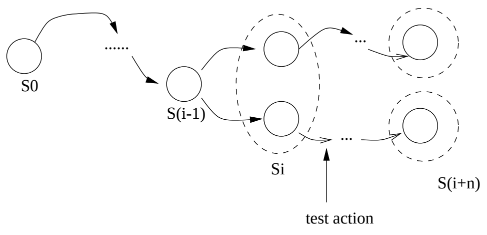

Figure 4: Discriminate possible candidates.

4.1. Finding discriminating input

Struss[2] has given an elegant and general exposition of the process of test generation. The core of his theory is that an input vector should discriminate between the possible behavioral modes. That is, the input vector should be an *hitting set* of the family of sets representing the inputs that discriminate between the behavioral modes.

We assume that the system is faulted and that for a trajectory σ at a certain state s_i it is not possible to derive the next state unambiguously, but that we have two options, s_i^a and s_i^b, both of comparable likelihood. The problem now is to find a set of actions U_i such that trajectory $\sigma^a = (\sigma, s_i^a, s_{i+1}^a, \ldots)$ is observably contradictory from trajectory $\sigma^b = (\sigma, s_i^b, s_{i+1}^b, \ldots)$; this will be denoted by $Obs(\sigma^a) \perp Obs(\sigma^b)$. Note that it may be necessary to “wait” more than one time step before the trajectories σ^a and σ^b become observably distinct, as the state of the system is only indirectly observable. Consider, for example, the calibration lamp of the GOME instrument. Switching on the calibration lamp doesn’t immediately lead to the lamp emitting light; due to its start-up behavior, it may take an extra two time steps before the lamp actually is on. In the meantime, we can’t really distinguish between a lamp being broken and a lamp that is starting up. However, after the two extra time steps, we know that the lamp must be emitting light (given the availability of electrical power, of course), and a broken lamp will for sure not be emitting light. Thus, we have to solve two problems:

- How to find a set U_i of actions that will discriminate between both possible trajectories.
- What is the time span in which we are willing to wait before both possible trajectories will be observably distinct. We assume at least that a so-called horizon parameter, representing the maximum number of simulation iteration, is given.

Finding test actions We assume that the ambiguity is caused by an discrepancy between expected normal behavior and observed behavior. That is, the SUS is faulted. In this case, we do not have to test all possible inputs of the SUS, but only a subset.

Let o be an output variable causing the inconsistency between the system description SD, the “normal” expected state S_{i+1}, and the observations O_{i+1}. Furthermore, let $Causes \subseteq \Pi_u \times \Pi_o$ be a relation between input variables and output variables, where $(\mathsf{u}, \mathsf{o}) \in Causes$ iff the variable u may affect the value of variable o. The relation $Causes$ can be determined at forehand, by inspecting the so-called constraint network[5] of the system description SD. It holds that $(\mathsf{u}, \mathsf{o}) \in Causes$ if there is a path in the constraint network from the node representing u to the node representing o, and where is assumed that for each state variable s there is always a connection between the node representing s and the node representing $\bigcirc\mathsf{s}$. For all variables that cause an inconsistency between normal expected behavior and observed behavior, we only have to consider the input variables that might affect one of these output variables.

Stop criterion Although we assume a maximum number (the so-called $Horizon$) of simulation steps we are willing to wait before the two trajectories become observably distinct, it is also possible to give an estimate the minimum number of simulation steps. Recall that for the calibration lamp we have to wait two extra time units before we are sure that it emits light. In general, an action *may* cause two trajectories to be observably distinct: (i) If in the next state, at least one state variable, say $\mathsf{v\text{-}s}$, has a different value in S_{i+1}^a and in S_{i+1}^b; and (ii) if the states S_{i+1}^a and S_{i+1}^b lead to an evolution of system behavior such that the observational part of one trajectory is contradictory with the observational part of the other.

In order to obtain a contradiction, there must be a transition $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \bigcirc\Psi_{kl}$ such that $\{\bigwedge_l \Psi_{kl}\} \cup$ SD is contradictory with $\{\bigwedge_l \Psi_{kl}\} \cup$ SD for a $j \geq 1$. In such a case, the condition $\bigwedge_l \Phi_{kl}$ must constrain, via SD, output variables. The number of simulation steps is at least the number of state transitions such that the state variable v-s constrains, via SD and $\bigwedge_l \Psi_{kl}$, one or more output variables.

Furthermore, the simulation can be stopped if (i) the simulation horizon has been exceeded, or (ii) both trajectories become periodic or are in steady-state (then the state sequence will repeat itself), or (iii) both new simulated states S^a_{i+1+t} and S^b_{i+1+t} are equal (then both sequences will never be observably distinct).

Algorithm Algorithm 1 is a (non-deterministic) generate-and-test algorithm to find a set of input variable value pairs that distinguish two possible trajectories $\sigma^a = (\sigma, S^a_i)$ and σ^b. Basically, the algorithm simply performs a simulation, taking the two possible states S^a_i and S^b_i as a starting point, until a difference occurs. During the simulation phase, we assume that no new faults occur. Recall that a transition operator $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \bigcirc\Psi_{kl}$ is composed out of the transition operators of the l individual components of the SUS. As we expect at least one fault in the SUS, one of the transition elements, say $\Phi_{kl_1} \supset \bigcirc\Psi_{kl_1}$, represents a transition to a fault situation. For the simulation we assume that the governing transition functions τ^a_k and τ^b_k for the two alternatives are such that the components remain in their respective fault modes and that the normally assumed components take their nominal transitions. Furthermore, we also assume that for any $\bigwedge_l \Psi_{kl}$, the set $\{\bigwedge_l \Psi_{kl}\} \cup$ SD is consistent. Thus, given a transition function τ_k, the next state S_{i_1+t} can be determined as the set of variable-value pairs that is consistent with $\bigwedge_l \Psi_{kl}$ for which SD$\cup S_{i+t} \cup U_t \models \Phi_{kl}$. Note that an empty input

Algorithm 1 Find discriminating test input

Input: Trajectories $\sigma^a = (\sigma, S^a_i)$ and $\sigma^b = (\sigma, S^b_i)$.
Input: A set $\{\mathtt{o_1}, \ldots, \mathtt{o_n}\}$ of contradictory outputs.
Output: A set U of input variables that causes trajectory σ^a and σ^b to be observably distinct.

```
1:  while not all possibilities for $U$ checked do
2:      Guess a set $U$ of variable - value pairs, where each variable u in $U$ $(\mathtt{u}, \mathtt{o_i}) \in Causes$ with $1 \leq i \leq n$.
3:      $t \leftarrow 0$.
4:      repeat
5:          /* In this algorithm, $\tau^a_k$ and $\tau^b_k$ refer to transition functions such that the system remains in the failure modes as identified by $S^a_i$ and $S^b_i$, respectively, and otherwise take the nominal transitions. */
6:          Compute the next states $S^a_{i+t+1}$ and $S^b_{i+t+1}$ by
            using the transitions $\tau^a_k \equiv \bigwedge_l \Phi^a_{kl} \supset \Psi^a_{kl}$ and $\tau^b_k \equiv \bigwedge_l \Phi^b_{kl} \supset \Psi^b_{kl}$, where: $SD \cup S^a_{i+t} \cup U_t \models \Phi^a_{kl}$, $SD \cup S^b_{i+t} \cup U_t \models \Phi^b_{kl}$, with $U_t = U$ for $t = 0$ and $U_t = \emptyset$ for $t > 0$.
7:          if $Obs(S^a_{i+1+t}) \perp Obs(S^b_{i+1+t})$ then
8:              return $U$.
9:          end if
10:         $\sigma^a \leftarrow (\sigma^a, S^a_{i+1+t})$, and $\sigma^b \leftarrow (\sigma^b, S^b_{i+1+t})$.
11:         $t \leftarrow t + 1$.
12:     until See text for stop criterion
13: end while
14: return no-discriminating-input-set-found.
```

set U may result in two observably distinct trajectories.

Computing the next state can be done in linear time (in the number of sub-transition systems) if use is made of an incomplete[3, 5] constraint propagator. In practice, when converting the non-deterministic algorithm to a deterministic one, the process of guessing a set U of possible input commands can be guided by: (i) The possible minimum number of time steps needed before the two trajectories become observably distinct. We prefer actions that constrain variables as soon as possible. (ii) The cost of a change in state of a component caused by a variable-value pair from U in a way similar to the method[3] of Williams and Nayak.

4.2. Making use of Built In Tests

The process of finding an appropriate set U of discriminating input variable-value pairs may be an expensive one. In many systems, however, we often find a number of so-called Built In Tests (BITs) that are specially engineered in such a way that a particular part of the state space of the SUS is inspected. In the GOME example described earlier, the re-calibration sequence (calibration lamp set on, and the scan mirror in the calibration position) is an example of a BIT.

In general, a BIT is sequence $(U_0, U_1, \ldots, U_n)$, where each U_i, with $1 \leq i \leq n$, is a set of input variable-value pairs. Applying a BIT $= (U_0, U_1, \ldots, U_n)$, at a point t in time means that U_0 is applied as input to the system at time t, U_1 at time $t + 1$, and, in general, U_i at time $t + i$. Algorithm 1, with minor changes, can also be used to test whether a BIT discriminates between possible diagnoses. The changes needed are: Instead of a guessed values for U, we use the input set U_t from the BIT to compute S^a_{i+1+t} and S^b_{i+1+t}.

In order to possibly speed up the process of finding an appropriate set U, we may make use the available BITs of the system. First, we may test whether one of the BITs lead to two observably distinct trajectories; if this is not the case, the test system may apply Algorithm 1.

5. SAFE TESTS

The test procedure as described in the previous section may help to discriminate between possible candidates. Care must be taken, however, as a test vector changes the state of the system. As we do not know the actual health state of the SUS, the newly applied test vector may induce new errors. Recall the fault in the GOME instrument where the mirror was in a different bias position than we expected.

Scarl[1] introduced the notion of so-called hazard condition constraints that if not satisfied indicate that the system's health is in jeopardy. A test action should not be put into action if simulation with a possible candidate shows that one of the hazard condition constraints are violated.

5.1. Specifying hazard conditions

One of the advantages of model-based diagnosis is that specification of system behavior is done by a kind of type system of component descriptions. Once a component type has been defined, it can be re-used, by instantiating the type definition, at will. Similarly, in the ideal case, safety constraints should also be specified at the type level only. As we will see, sometimes, this is possible, but unfortunately, not always. First, we describe an example where hazard constraints can be given at the type level. Figure 5 shows an electrical circuit where electrical power is delivered by a battery. Assume

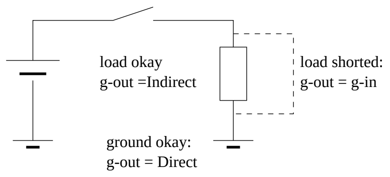

Figure 5: Simple battery circuit.

that one of the possible diagnoses is that the (resistive) load is shorted. At all times, it should be avoided that the battery will be shorted. This can be avoided by assigning a pair `g-in` and `g-out` of variables to each component, where each variable has three possible values: `Open`, `Direct`, and `Indirect`. The value `g-in=Open` means that the component is not connected to ground (*e.g.*, somewhere downstreams a switch is open); `g-in=Direct` means that the component is directly connected to ground; and `g-in=Indirect` means that the component is connected to ground via another, resistive, element. The variable `g-out` is used to connect the components; thus, given the configuration as given in Figure 5, we have the following behavioral specification with respect to the variables describing whether a component is connected to ground, and (ab)normality of the load:

$$\begin{gathered}
\texttt{bat.g-in} = \texttt{switch.g-out}, \\
\texttt{load.g-out} = \texttt{switch.g-in}, \\
\texttt{grnd.g-out} = \texttt{Direct}, \\
\texttt{switch.pos=Opened} \supset (\texttt{switch.g-out=Open}) \\
\texttt{switch.pos=Closed} \supset (\texttt{switch.g-out=switch.gin}) \\
\texttt{load.shorted} \supset (\texttt{grnd.g-out} = \texttt{load.g-in}), \\
\texttt{load.normal} \supset (\texttt{grnd.g-out} = (\texttt{load.g-in} \oplus \texttt{Indirect})).
\end{gathered}$$

Note that we have extended here the finite-domain constraint language with the $\oplus$ operator over the "connected to ground" domain. The $\oplus$ operator is defined as follows:

$\oplus$	Open	Indirect	Direct
Open	Open	Open	Open
Indirect	Open	Indirect	Indirect
Direct	Open	Indirect	Direct

The safety condition that must not be violated, for every battery, is that $\neg(\texttt{bat.g-in} = \texttt{Direct})$.

Unfortunately, specifying hazard conditions can not always be done at the type level. In general, we also have to check at the level of the instantiated components. Consider, for example, the scan mirror example of the GOME instrument. If the scan mirror is possibly in one of its extreme positions we should avoid to send a scan command to the mirror. This is, however, not a generic property of a scan mirror; it is only because the scan mirror is placed close to the chassis of the instrument that giving a scan command at one of the extreme mirror positions causes harm.

This significantly complicates the process of specifying hazard constraints, as the whole system description must be manually inspected.

5.2. Verifying conditions

We will assume that the hazard condition constraints of all the instantiated components are specified in the same (finite-domain constraint) language as the system description SD is, and that all hazard condition constraints are collected in the set HC. We say that a SUS with a system description SD, previous state S_{i-1} and input U_{i-1} will not violate a hazard condition if SD$\cup S_i \cup$HC is consistent, where the current state S_i is determined by assuming that during the state transition no new faults occurs.

As we remarked earlier, it is not enough to test only the next state, but, in principle, the whole trajectory must be inspected. As in the case of finding a set of discriminating input signals, we have to simulate the evolution of the system, and at each time instance check the hazard condition constraints, as is shown in Algorithm 2. In practice, this simulate-and-check algorithm works in conjunction with the test algorithm (Algorithm 1). That is, Algorithm 2 can be used as an extra stopping criterion in Algorithm 1 (as soon as a set U_i violates one of the hazard condition constraints, the simulation should stop), and the computation of the next state S_{i+t+1} can be shared by both algorithms.

Furthermore, once a hazard condition violation has been detected, the culprit (in general, represented as a negative clause) of the inconsistency (in terms of elements of the state S_i and control action U_i) can be stored in a so-called "nogood"[5] database. If a clause in the nogood database is a subset of the current $S_i \cup U_i$, the search process can immediately stop.

Algorithm 2 Check hazard condition constraints

Input: Current likely state S_i of the system.
Input: A sequence (BIT) $(U_0, U_1, \ldots, U_n)$ or a single set U_0 computed by Algorithm 1.
Output: true iff no hazard constraint is violated by the BIT or U_0.

```
 1: $t \leftarrow 0$.
 2: /* In this algorithm, $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \bigcirc\Psi_{kl}$ refers to a transition function such that the system remains in those failure modes as identified by $S_i$, and otherwise
    take the nominal transitions. */
 3: Compute the state $S_{t+1}$ by using the transition $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \Psi_{kl}$, where $SD \cup S_i \cup U_0 \models \Phi_{kl}$.
 4: while $(t < Horizon)$ And ($S_{i+t+1}$ not periodic) do
 5:   if SD $\cup$ $S_{i+t+1}$ $\cup$ HC is inconsistent then
 6:     Return false.
 7:   end if
 8:   $t \leftarrow t + 1$.
 9:   Compute the state $S_{i+t+1}$ by using the transition $\tau_k \equiv \bigwedge_l \Phi_{kl} \supset \Psi_{kl}$, where $SD \cup S_{i+t} \cup U_0 \models \Phi_{kl}$ and where
      $U_t$ is taken from the input BIT (or single $U_0$) for $1 \leq t \leq n$, and $U_t = \emptyset$ for $t > n$.
10: end while
11: Return true.
```

6. APPLICATION AND RELATED WORK

We are developing a tool to support the construction of model-based diagnosis systems for systems such as satellite instruments. Part of the tool will be a subsystem generating safe test vectors that present their findings to a human operator. That is, test vectors will not be applied autonomously, but will only be given as an advice.

Finding a set of input actions that discriminate between two possible explanations for observed behavior shares many similarities with the process of Mode Configuration as found in the Livingstone controller as described by Williams and Nayak.[3] The conflict-directed search algorithm they described can be re-used as a core for Algorithm 1.

The process of finding safe test actions is similar to proving the safety properties of a concurrent system.[6] Safety properties are often defined as constraints defined over the state space of the concurrent system. The difference with finding safe test actions is that safety conditions are verified at forehand,[7, 8] where safe test actions are to be generated during the operation of the SUS.

The work described in this paper can also be viewed as a form of so-called Model Predictive Control (MPC).[9] MPC is usually associated with optimal control of continuous systems (although much work has also been done on the control of hybrid and discrete systems). Like model-based diagnosis, MPC uses an explicit model of the behavior of the system and an optimalization criterion to determine the control actions. The MPC algorithm also performs a simulation during a number of time steps of the system under control, in order to select the best control actions according to the criterion.

7. CONCLUSIONS

In this paper, we have investigated so-called safe testing for dynamic systems. Although the dynamic nature of systems doesn't change the notion of safe testing conceptually, it does affect the operational side. In order to check whether a certain set of actions does indeed discriminate between possible explanations for observed behavior, or whether that set of actions violates a hazard condition, it is not enough to check the so-called next state only. As the state of the system is only indirectly observable, distinctions between two possible trajectories or violations of hazard constraints may become apparent only after some time steps.

8. ACKNOWLEDGEMENTS

This paper is partly the result of the research programme Freight Transport Automation and Multimodality (FTAM), carried out within the TRAIL research school for Transport, Infrastructure and Logistics, and financed by Delft University of Technology

REFERENCES

1. E. Scarl, “Testing safely,” in *DX-96, The Seventh International Workshop on the Principles of Diagnosis*, S. Abu-Hakima, ed., (Val Morin, Quebec, Canada), October 1996.
2. P. Struss, “Testing for discrimination of diagnoses,” in *DX-94, The Fifth International Workshop on the Principles of Diagnosis*, G. Provan, ed., (New Paltz, NY), October 1994.
3. B. C. Williams and P. P. Nayak, “A model-based approach to reactive self-configuring systems,” in *AAAI-96*, pp. 971–978, 1996.
4. A. Bos, J. Callies, and A. Lefebvre, “Model-based monitoring and diagnosis of a satellite-based instrument,” *Telematics and Informatics* **3 and 4**(12), pp. 161–170, 1995.
5. J. de Kleer and B. C. Williams, “Diagnosing multiple faults,” *Artificial Intelligence* **32**, pp. 97–130, 1987.
6. Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Connurent Systems – Specification*, Springer Verlag, 1991.
7. H. Weinberg and N. Lynch, “Correctness of vehicle control systems – a case study,” in *17th IEEE Real-Time Systems Symposium*, pp. 62–72, 1996.
8. L. Lamport, “The temporal logic of actions,” *ACM Transactions on Programming Languages and Systems* **16**, pp. 872–923, 1994.
9. A. Bemporad and M. Morari, “Robust model predictive control: A survey,” in *Robustness in Indentification and Control*, *Lecture Notes in Control and Information Sciences*, 1999.