SMT-based Validation of Timed Failure Propagation Graphs

Marco Bozzano and **Alessandro Cimatti** and **Marco Gario** and **Andrea Micheli**
Fondazione Bruno Kessler – Italy
{bozzano,cimatti,gario,amicheli}@fbk.eu

Abstract

Timed Failure Propagation Graphs (TFPGs) are a formalism used in industry to describe failure propagation in a dynamic partially observable system. TFPGs are commonly used to perform model-based diagnosis. As in any model-based diagnosis approach, however, the quality of the diagnosis strongly depends on the quality of the model. Approaches to certify the quality of the TFPG are limited and mainly rely on testing. In this work we address this problem by leveraging efficient Satisfiability Modulo Theories (SMT) engines to perform exhaustive reasoning on TFPGs. We apply model-checking techniques to certify that a given TFPG satisfies (or not) a property of interest. Moreover, we discuss the problem of refinement and diagnosability testing and empirically show that our technique can be used to efficiently solve them.

1 Introduction

Timely detection, identification and recovery of faults is an essential component for the correct operation of complex critical systems (e.g., trains, satellites, cars, industrial plants). Failures typically originate from basic faults, can manifest with delays, and can interact with each other over time in very complex ways. Timed Failure Propagation graphs (TFPGs) have been introduced in (Karsai, Abdelwahed, and Biswas 2003) as a framework to capture the temporal propagation of faults in complex systems, and to support important run-time activities such as diagnosis and prognosis. Intuitively, a TFPG is a graph-like model that accounts for the temporal progression of failures in dynamic systems and for the causality between failure effects, taking into consideration time delays, system reconfiguration, partial observability, and sensor failures.

In practice, TFPGs have been applied in several industrial contexts. In aerospace, NASA (Hayden et al. 2006) positively evaluated them in the context of diagnostic technology for manned aircraft. Moreover, Boeing has been using them (Ofsthun and Abdelwahed 2007) for performing maintenance monitoring of systems; in particular, in (Atlas et al. 2001) they present an integrated vehicle health management system based on TFPGs. Furthermore, the European Space Agency has funded projects (European Space Agency 2011; 2012) where TFPGs are used as the basis for the specification of fault detection and isolation components (Bittner et al. 2014). Recently, in addition to monitoring of safety critical systems, there has been some interest in using TFPGs for performing software monitoring (Dubey, Karsai, and Mahadevan 2010; 2011). Other applications of TFPGs are described in (Srivastava and Han 2012).

Unfortunately, the practical application of TFPGs is currently clashing against the lack of suitable validation methods and tools. TFPGs are usually built manually by safety engineers, based on their own knowledge of the relations between faults in the system, and based on safety artifacts such as Fault-Trees (Vesely et al. 2002). In the few cases in which they can be generated from a system model, there is no way for the users to modify or adapt them and still be able to certify their quality. TFPGs are usually validated via extensive testing, and no approach exists for performing a more exhaustive analysis and validation of the model. However, since TFPGs are used as a basis for diagnosis, it is fundamental to be able to establish their formal properties, and to gain confidence in their adequacy — similarly to other model-based approaches, the effectiveness of the reasoning relies on the accuracy of the model.

Interest in validation is witnessed in (Strasser and Sheppard 2011), that introduces a data-driven method called *alarm-sequence maturation* in order to correct errors in the causality relations of the model. This method, however, requires data to be already available, and thus can be applied only after the overall system has been put into production. In many contexts, e.g., aerospace, it is not possible or desirable to wait until deployment of the system to validate the TFPG. Thus the validation should be performed in the design phase.

In this paper, we propose a practical approach to support the analysis and validation of TFPG models. We define some important design-time queries, and show how they can be efficiently answered by performing symbolic reasoning. Our approach is based on a logical characterization of TFPGs, and cast in the field of Satisfiability Modulo Theory (SMT) (Barrett et al. 2009). The set of possible executions of a TFPG is modeled as a formula in SMT. The reasoning tasks are expressed in form of logical queries in SMT, and the implementation is based on the use of an efficient SMT solver. We explore the problems of *validation*, *refinement testing*, and *diagnosis and diagnosability*. We im-

plemented and experimentally evaluated our approach, and show its ability to leverage symbolic SMT-based techniques to deal with the state-space explosion problem. In this paper, we focus only on three possible analyses, but using an SMT characterization of the TFPG, it becomes possible to easily define new analyses, or implement other more classical problems, such as prognosis.

Our approach is complementary to the available tools, such as the state-of-the-art FACT tool-set (Karsai 2013), where limited support for model validation is provided. In FACT, for example, it is not possible to provide a condition on a TFPG and obtain a complete execution satisfying it automatically, nor verify properties against the TFPG. One could use our approach to test the correctness of the TFPG model, and then perform diagnosis and prognosis using the FACT tool-set.

The rest of the paper is structured as follows. Section 2 and 3 provide background on SMT and TFPGs. Section 4 and 5 describe the SMT encoding and the reasoning tasks. In Section 6 we experimentally evaluate the approach, and Section 7 concludes with ideas for future work.

2 Background

Our setting is standard first order logic and we use the standard semantic notions of interpretation and satisfiability (Kleene 1967). We use the notation $x, y, v, \ldots$ for variables, and $\vec{x}, \vec{y}, \vec{v}, \ldots$ for vectors of theory or Boolean variables. Terms and formulae are referred to as expressions. We write $\phi(x)$ to highlight that x is free in ϕ, and $\phi(\vec{x})$ to highlight that the free variables of ϕ are variables in $\vec{x}$. We use the notation $Q\vec{x}.\phi(\vec{x})$, with $Q \in \{\forall, \exists\}$, to denote the formula $Qx_1.Qx_2.\ldots.Qx_n.\phi(x_1, \ldots, x_n)$.

Given a first-order formula ψ with non-logical symbols interpreted in a decidable background theory T, *Satisfiability Modulo Theory* (SMT) (Barrett et al. 2009) is the problem of deciding whether there exists a satisfying assignment to the free variables in ψ. In this paper, we use the theory of linear arithmetic over the real numbers ($\mathcal{LRA}$). A formula in $\mathcal{LRA}$ is an arbitrary Boolean combination, a universal ($\forall$) or an existential ($\exists$) quantification, of atoms in the form $\sum_i a_i x_i \bowtie c$ where $\bowtie \in \{>, <, \leq, \geq, \neq, =\}$, each x_i is a real variable, each a_i and c are real constants. Difference logic ($\mathcal{RDL}$) is the subset of $\mathcal{LRA}$ such that atoms have the form $x_i - x_j \bowtie c$. We write $x - y \in [a, b]$ meaning the formula $(x - y) \geq a \wedge (x - y) \leq b$. If a is $-\infty$ then the first conjunct is omitted and similarly, if b is $+\infty$ then the second conjunct is omitted.

An SMT solver is a decision procedure which solves the satisfiability problem for a formula in a decidable subset of First-Order Logic. Due to their efficiency, SMT solvers have been recently employed in various fields such as Formal Verification (Beyer et al. 2009), Temporal Reasoning (Cimatti, Micheli, and Roveri 2014) and many others.

3 Timed Failure Propagation Graphs

The Battery Powered Sensor System (BPSS) (Figure 1) will be our running example. The BPSS provides a redundant reading of the sensors to a device. Internal batteries provide backup in case of failure of the power supply. The safety of the system depends on at least one of the sensors providing a reading at any time.

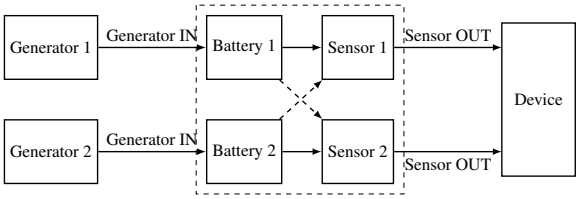

Figure 1: Battery-Powered Sensor System schema

A *failure mode* is a failure of a component of the system. A component might fail in more than one way, therefore having more than one failure mode associated with it. We call *fault* the occurrence of a failure of the component. A fault in a component will produce anomalies in the system behavior, that we call *discrepancies*. In the BPSS, we define one failure mode for the generator, and one for the sensor. The generator can fail and stop supplying power (G_{Off}), and the sensor can stop providing a reading (S_{Off}). After the generator fails, the battery will start discharging. When the battery is exhausted, the attached sensors will stop working. Examples of discrepancies are the absence of power from the generator, the battery level going below a threshold and the sensor not providing a reading.

Monitored Discrepancies are discrepancies to which we attach some monitor. In our example, we monitor the level of the battery, and are warned when the level goes below a certain threshold. Not all discrepancies can be monitored, due to physical or design limitations. Monitors can also fail, providing false positives or false negatives.

System modes (simply *modes*) are configurations of the system that are relevant for capturing the propagation of faults. In the BPSS the sensors are powered by their own battery, however, in case of faults, one battery can power both. We define 3 modes: $Primary$, $Secondary_1$, $Secondary_2$ (e.g., in mode $Secondary_1$ Battery 1 is powering both sensors). If Generator 1 fails while the system is powered by Generator 2, there will be no impact on the system.

Timed Failure Propagation Graphs (TFPG) were introduced in (Karsai, Abdelwahed, and Biswas 2003) to model the progression of faults in a system and perform online diagnosis and prognosis. A TFPG is a directed graph model where nodes represent *failure modes* and *discrepancies*. Edges represent the causality between nodes, and provide information on the delay in the propagation. By labeling the edges with *system modes*, we can encode switching-systems in which different propagations are possible in the different modes. Figure 2 shows a TFPG for the BPSS. Boxes with dotted lines are failure modes, whereas discrepancies are either circles (OR discrepancies) or boxes (AND discrepancies) with solid lines. B_{LOW} and Sys_{DEAD} are monitored discrepancies, thus we show the monitor graphically (with diamonds). Edges include information on the propagation time and modes in which they are active.

The non-determinism on the propagation time between $G1_{DEAD}$ and $B1_{LOW}$ models the fact that we do not

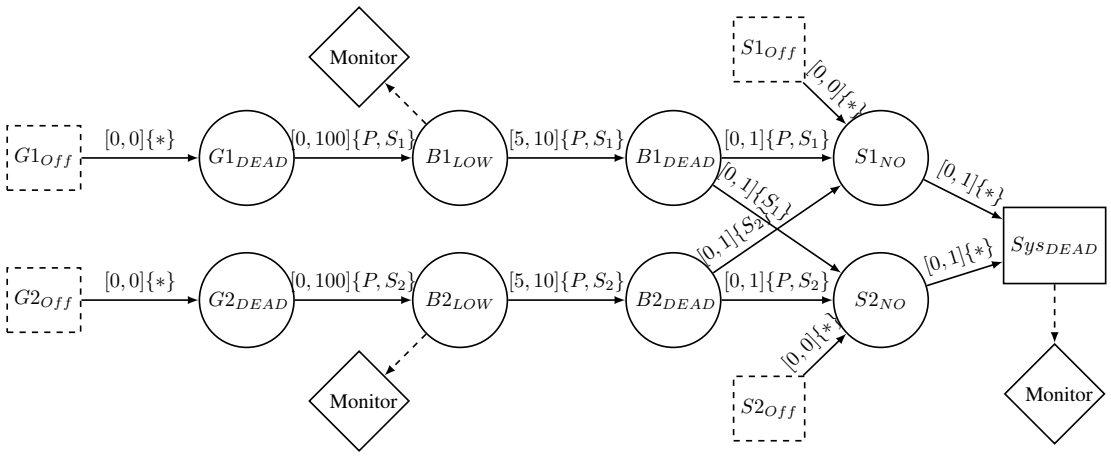

Figure 2: Example TFPG for the BPSS example. The "$*$" is used to indicate all modes (i.e., P,S_1,S_2)

know the charge level of the battery until we get to a critical level (B_{LOW}). Additionally, we allow for a small non-determinism between the activation of the discrepancy, the absence of output (S_{NO}) and the failure of the system. The uncertainty on the propagation time between B_{LOW} and B_{DEAD} is motivated by the fact that the depletion of the battery will take more time if we are in the *Primary* mode rather than in the *Secondary*$_1$ (or *Secondary*$_2$) mode.

Definition. A TFPG is a structure $G = \langle F, D, E, M, ET, EM, DC, DS\rangle$, where F is a non-empty set of failure modes; D is a non-empty set of discrepancies; $E \subseteq V \times V$ is a set of edges connecting the nodes $V = F \cup D$; M is a non-empty set of system modes. At each time instant the system can be in only one mode; $ET : E \rightarrow I$ is a map that associates every edge in E with a time interval $[t_{min}, t_{max}] \in I$ indicating the min/max propagation time on the edge (where, $I \in \mathbb{R}_+ \times (\mathbb{R}_+ \cup \{+\infty\})$ and $t_{min} \leq t_{max}$); $EM : E \rightarrow \mathcal{P}(M)$ is a map that associates to every edge in E a set of modes in M. We assume that $EM(e) \neq \emptyset$ for any edge $e \in E$. $DC : D \rightarrow \{\texttt{AND}, \texttt{OR}\}$ is a map defining the type; $DS : D \rightarrow \{\texttt{M}, \texttt{I}\}$ defines whether the discrepancy is monitored (M) or not (I – inactive). Failure modes are always root nodes, and all discrepancies must have at least one incoming edge. Circular paths (but not self-loops) are possible.

The semantics (Abdelwahed et al. 2009) of TFPGs says that the state of a node indicates whether the failure effects reached that node. A failure propagates through an edge $e = (v, w)$, only if the edge is active throughout the propagation, that is, from the moment v activates to the time w activates. An edge e is active if and only if the current mode m of the system is compatible with the modes of the edge ($m \in EM(e)$). For an OR node w and an edge $e = (v, w) \in E$, once a failure effect reaches v at time t, it will reach w at a time t', where $e.tmin \leq t' - t \leq e.tmax$ and the edge e is active during the whole propagation. On the other hand, the activation period of an AND node v' is the composition of the activation periods for each link $(v, w) \in E$. If an edge is deactivated any time during the propagation (due to mode switching), the propagation stops. Links are assumed memory-less thus a failure propagation is independent of any (incomplete) previous propagation. Finally, once a node activates, it changes permanently, and will not be affected by any future failure propagation.

In the rest of this paper, we consider TFPGs in which the mode is frozen. Namely, we fix any of the available modes beforehand, and we assume that it does not change for the entire execution of the system. This assumption has been validated by domain experts, and it is justified by the fact that most of the reasoning (e.g., diagnosis) is of interest when the system is in a stable state. Given the number of system changes and possibly unpredictable interactions, modeling of fault propagation during mode-switching is not considered in practice.

4 TFPG as an SMT formula

We translate a TFPG into an SMT formula using the $\mathcal{RDL}$ theory. The encoding closely follows the definition of TFPG, and has been extensively validated using existing case-studies, and randomly generated TFPGs.

We associate to each node in the TFPG a *state variable* and an *activation time-point*. The semantics of the TFPG is encoded by defining constraints for the OR and AND nodes.

Given a TFPG G we create a formula $\varphi(\vec{ud}, \vec{udt}, m)$, where m is the current system mode, and $\vec{ud}$ and $\vec{udt}$ are vectors that define, respectively, the state (active or not) and the activation time of each node in G. Those are *unobservable* information and for brevity we combine them in a single vector $\vec{u} = \vec{ud} \cup \vec{udt}$. Let us define $M(e, m) = \bigvee_{\mu \in EM(e)} (m = \mu)$ as an expression defining whether an edge e is compatible with the mode m; φ is defined as:

$$\varphi(\vec{u}, m) = \bigwedge_{v \in D.\ DC(v)=\texttt{OR}} B_{or}(v, m) \ \wedge \ T_{or}(v, m) \wedge \bigwedge_{v \in D.\ DC(v)=\texttt{AND}} B_{and}(v, m) \ \wedge T_{and}(v, m)$$

$$B_{or}(v, m) = \vec{ud}(v) \leftrightarrow \bigvee_{(w,v) \in E} [\vec{ud}(w) \ \wedge M((w, v), m)]$$

$$B_{and}(v, m) = \vec{ud}(v) \leftrightarrow \bigwedge_{(w,v) \in E} [\vec{ud}(w) \ \wedge M((w, v), m)]$$

$$T_{or}(v,m) = \vec{ud}(v) \rightarrow [\bigvee_{(w,v)\in E} \Big(\vec{ud}(w) \ \wedge \ (\vec{udt}(v) - \vec{udt}(w)) \in ET((w,v)) \Big) \wedge \bigwedge_{(w,v)\in E} \Big(\vec{ud}(w) \rightarrow \ (\vec{udt}(v) - \vec{udt}(w)) \leq t_{max}((w,v)) \Big)]$$

$$T_{and}(v,m) = \vec{ud}(v) \rightarrow [\bigwedge_{(w,v)\in E} \Big(\vec{ud}(w) \ \wedge (\vec{udt}(v) - \vec{udt}(w)) \geq t_{min}((w,v)) \Big) \wedge \bigvee_{(w,v)\in E} \Big(\vec{ud}(v) \wedge \ (\vec{udt}(v) - \vec{udt}(w)) \leq t_{max}((w,v)) \Big)]$$

For each OR and AND, we express constraints on the boolean part (activation state of the node) with B_{or}, B_{and} and the temporal part (activation time of the node) with T_{or}, T_{and}. Intuitively, the boolean part expresses the possible combinations of nodes that can be active, while the temporal part encodes the temporal relation between their activation.

In B_{or} we state that a node is active if at least one of the predecessors is active and the mode is compatible. In B_{and} we require all predecessors to be active. T_{or} states that given an active node, there must be at least one predecessor with an activation time that is compatible with the t_{min}/t_{max}. We additionally require that all *active* predecessors nodes have an activation time that is compatible with the t_{max}. Finally, T_{and} encodes that if a node is active, all the predecessors are active with timings satisfying the t_{min} constraint, and that at least one satisfies the t_{max}. Note that the second part of T_{and} is similar to that of T_{or}. However, the key difference is that in the AND case, we allow all but one active discrepancies to violate the t_{max}constraint. The failure modes are left unconstrained. If φ is satisfiable, then a model for φ represents a possible execution of the TFPG. This encoding is polynomial in the number of nodes in the TFPG. In particular, it uses $O(|E|)$ theory ($\mathcal{RDL}$) atoms.

Diagnosis requires the ability to reason on the monitored (i.e., observable) discrepancies. In order to do so, we extend the encoding by adding new variables for the monitors. The monitor has the same state and activation time as the discrepancy, however, in order to be able to consider monitoring faults, we condition these constraints to a set of health variables $\vec{h}$. We call $\vec{o} = \vec{od} \cup \vec{odt}$, the *observable* variables, and write as $\psi(\vec{o}, \vec{u}, m, \vec{h})$ the TFPG with monitors:

$$\psi(\vec{o}, \vec{u}, m, \vec{h}) = \varphi(\vec{u}, m) \wedge \bigwedge_{v\in D.DS(v)=\texttt{M}} \vec{h}(v) \rightarrow (\vec{od}(v) = \vec{ud}(v) \wedge \vec{odt}(v) = \vec{udt}(v))$$

5 SMT-based Reasoning

The SMT encoding can be used to perform reasoning on the TFPG. We distinguish between reasoning tasks that can be done at *design time* or *runtime* (i.e., when a system is running). The objective of design time reasoning is to help the designer validate the TFPG, i.e., show that the TFPG captures the situations of interest. We focus on design time analysis, but also provide an example of how to perform a runtime task (diagnosis).

The design time reasoning task that we consider are the following: *model validation*, *refinement testing* and *diagnosability*. We show that all these problems can be handled in a uniform way by leveraging the SMT solver, without the need of defining multiple ad-hoc algorithms.

Partial Traces The SMT encoding of the TFPG represents the set of all possible executions (i.e., traces) of the TFPG: every model of the formula is a trace of the TFPG (and vice versa). We can ask the SMT solver to provide us with a trace of the TFPG, and can also add constraints defining how this trace should look like. E.g., we can ask for a trace of the BPSS in which the discrepancy $B2_{LOW}$ is active by using the formula $\tau(\vec{u}, m) = ud(B2_{LOW})$. Such a trace can be obtained from the SMT solver by asking for a model of (1).

$$\varphi(\vec{u}, m) \wedge \tau(\vec{u}, m) \tag{1}$$

A *model* is an assignment of concrete values to all the $\vec{ud}$, $\vec{udt}$ and m variables, telling which discrepancies and failure modes are active, and their activation time. If no model exists (the formula is unsatisfiable) there is no trace in the TFPG that satisfies the given requirement (τ). A *partial trace* can be defined by describing (with τ) the activation of some discrepancies without the need of specifying the behavior of all nodes. The SMT solver will provide a complete trace that satisfies the constraints, thus enabling the user to explore the behavior of the TFPG. τ can be any $\mathcal{LRA}$ formula over the mode, state and time-point variables of the TFPG, thus providing a large degree of expressiveness.

Model Validation Building on the idea of partial traces, we can ask whether some behavior is possible or not in the TFPG. Knowledge of the domain and of the original model, can be used to define properties that we expect the TFPG to satisfy. In the BPSS the failure of the generator $G1$ *may* lead $S1$ to stop working. However, if the system is in mode $Secondary_2$ the failure of $G1$ will not have any impact on $S1$. We call this type of property a *possibility*. We are interested in checking that some behaviors are possible in the TFPG. To do so, we use the concept of partial traces, and ask whether there exists a trace in the TFPG that satisfies our requirement. This is done by checking the satifiability of (1) with $\tau(\vec{u}, m) = \vec{ud}(G1_{Off}) \wedge \vec{ud}(S1_{NO})$.

An example of possibility checking is whether all nodes of the TFPG can eventually be activated. E.g., it might be that not all nodes can be activated in a given mode. A node that cannot be activated in any mode represents a modeling error. We find such nodes by running multiple possibility checks, optionally specifying in which modes we expect the node to be enabled. E.g., we cannot activate the discrepancy $B1_{LOW}$ in the mode $Secondary_2$, thus the following is unsatisfiable: $\varphi(\vec{u}, m) \wedge \vec{ud}(B1_{LOW}) \wedge m = Secondary_2$

The counter-part of possibility is *necessity*. In the BPSS we know that a battery cannot discharge if the associated generator is working. This property can be specified as follows. For $B1_{LOW}$ to be active, it is necessary for $G1_{Off}$ to

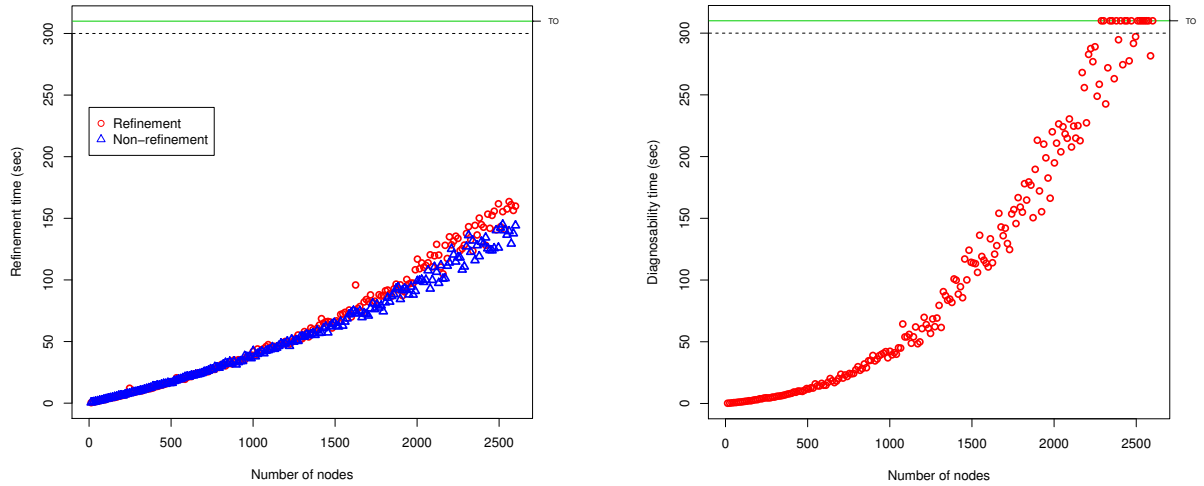

Figure 3: Scalability of the Refinement and Diagnosability. The "TO" line marks the examples that reached the timeout.

be active. Given $\tau(\vec{u}, m) = \vec{ud}(B1_{LOW}) \rightarrow \vec{ud}(G1_{Off})$, we check that the following is a validity:

$$\varphi(\vec{u}, m) \rightarrow (\vec{ud}(B1_{LOW}) \rightarrow \vec{ud}(G1_{Off}))$$

In words, every trace of the TFPG in which $B1_{LOW}$ is active requires $G1_{Off}$ to be active.

The purpose of model validation is to increase the confidence on the correct behavior of the model. Lets modify the BPSS TFPG so that the mode on the edge $(B2_{DEAD}, S1_{NO})$ becomes $Secondary_1$. This implies that there is no propagation between those two nodes in mode $Secondary_2$. This mistake could be detected with the following possibility requirement: It is possible for the single failure mode $G1_{Off}$ to cause Sys_{DEAD} (2). This execution is possible in the original BPSS but not in the modified one.

$$\begin{aligned}\tau(\vec{u}, m) =& \neg(\vec{ud}(G1_{Off}) \vee \vec{ud}(S1_{Off}) \vee \vec{ud}(S2_{Off})) \\ & \wedge \vec{ud}(G2_{Off}) \wedge \vec{ud}(Sys_{DEAD}) \end{aligned} \quad (2)$$

Refinement Changes in the models are a common activity during development. We are then usually interested in the relation between the original model and the modified one. In the BPSS example, there is uncertainty on the propagation time between $B1_{LOW}$ and $B1_{DEAD}$ due to the different discharge rates of the battery in the primary and secondary mode. This uncertainty can be removed by adding two intermediate discrepancies ($B1_P$, $B1_S$) that have an incoming edge from $B1_{LOW}$ and an outgoing edge to $B1_{DEAD}$. Intuitively, $B1_P$ can be reached only in mode $Primary$, thus providing an exact value for the propagation ($ET = [10, 10]$) (similarly for $B1_S$ in $Secondary_1$ with $ET = [5, 5]$). We need to check that this new TFPG ($BPSS'$) refines the original TFPG ($BPSS$). Therefore, we define a mapping between the two saying that all nodes that exist in both TFPGs must have the same state and activation times, and check that each execution in the new TFPG has a corresponding execution in the old one (i.e., is a refinement).

Given two TFPGs G_1, G_2 and a (partial) mapping $\gamma(\vec{u}_1, \vec{u}_2)$ between their nodes, we say that G_1 refines G_2 if every trace of G_1 can be mapped to a trace of G_2 (3). A simple example of mapping is (5).

$$\forall \vec{u}_1, m. \varphi_{G1}(\vec{u}_1, m) \rightarrow \exists \vec{u}_2. (\gamma(\vec{u}_1, \vec{u}_2) \wedge \varphi_{G2}(\vec{u}_2, m)) \quad (3)$$

$$\varphi_{G1}(\vec{u}_1, m) \wedge \forall \vec{u}_2. \neg(\gamma(\vec{u}_1, \vec{u}_2) \wedge \varphi_{G2}(\vec{u}_2, m)) \quad (4)$$

We do not feed (3) directly to the SMT solver, but instead use its negation (4), since satisfiability of (4) will provide the user with a trace that belongs to the first TFPG but that cannot be mapped to any trace of the second, thus simplifying the debugging process.

$$\begin{aligned}\gamma(\vec{u}_1, \vec{u}_2) = \bigwedge_{v \in V} & (\vec{ud}_1(v) \leftrightarrow \vec{ud}_2(v)) \wedge \\ & (\vec{ud}_2(v) \rightarrow \vec{udt}_1(v) = \vec{udt}_2(v)) \end{aligned} \quad (5)$$

Diagnosis and Diagnosability The goal of *diagnosis* is to understand which failure modes caused the observed discrepancies. This is achieved by generating all possible executions consistent with the observations and considering all the sets of faults that occur in those executions (Model Based Diagnosis (Reiter 1987)).

A *state condition* $\beta(\vec{u})$ is a relation on the unobservable discrepancies and time points. In the most basic case, a state condition is a single failure mode: $\beta(\vec{u}) = \vec{ud}(fm)$. In the general case, a state condition captures a situation of interest in the system. E.g., the fact that a failure mode occurred in a given time-frame: $\beta(\vec{u}) = \vec{ud}(fm) \wedge 5 \leq \vec{udt}(fm) \leq 10$ or that two discrepancies activated in a particular order:

$$\beta(\vec{u}) = \vec{ud}(D1) \wedge \vec{ud}(D2) \wedge \vec{udt}(D1) \leq \vec{udt}(D2)$$

To perform diagnosis, we ask if a given state condition is possible given the observations provided by the monitored discrepancies. This basically boils down to a possibility check in which we define $\tau = \beta(\vec{u}) \wedge obs(\vec{o})$, where $obs(\vec{o})$ denotes the *observation*. However, we are usually interested in situations in which there is no ambiguity on the diagnosis, i.e., there is only one possible explanation for the given observation (Bozzano et al. 2014). Therefore, instead of checking possibility, we check necessity. We can ask whether given the observable trace $obs(\vec{o}) = \vec{od}(Sys_{DEAD}) \wedge \vec{odt}(Sys_{DEAD}) = 10$ (Sys_{DEAD} was activated at time

$t = 10$) it is necessary for $S1_{Off}$ to have been activated before time $t = 10$: $\beta(\vec{u}) = \vec{ud}(S1_{Off}) \wedge \vec{udt}(S1_{Off}) \leq 10$. If $\psi(\vec{o}, \vec{u}, m, \vec{h}) \wedge obs(\vec{o}) \wedge \neg\beta(\vec{u})$ is unsatisfiable, the state condition is the only possible explanation for the observation, otherwise we are provided with a counter-example. The observable trace *obs* can be any $\mathcal{LRA}$ formula, thus we can express complex patterns of observations, e.g., temporal uncertainty: $obs = 5 \leq \vec{odt}(Sys_{DEAD}) \leq 10$.

Since the process of diagnosis is performed at runtime, we would like to have some sort of guarantee on its behavior at design time. Is it *always* possible to detect the occurrence of a state condition given any observation? If so, we say that the TFPG is *diagnosable*. To disprove diagnosability, we use the twin-plant construction (Jiang et al. 2001). Our goal is to try to find pair of traces that have the same observations s.t. one satisfies the condition but the other does not, this pair of traces is known as *critical pair* (Sampath et al. 1995). A system is diagnosable if it has no critical pair.

In (6) we test for the existence of a critical pair. Moreover, we enforce some relation on the health variables of the monitors.

$$\psi(\vec{o}, \vec{u}_1, m, \vec{h}_1) \wedge \psi(\vec{o}, \vec{u}_2, m, \vec{h}_2) \wedge \beta(\vec{u}_1) \wedge \neg\beta(\vec{u}_2) \wedge Healthy(\vec{h}_1, \vec{h}_2) \quad (6)$$

Usually, we are interested in checking diagnosability when all sensors are working correctly. However, we can explore other configurations for the health variables. For example, we can explore how many sensor failures we can afford before a given state condition becomes non-diagnosable by changing the relation $Healthy$.

The definition of diagnosability provided in (6) requires that all observations have been made. Due to the importance of timing in detection and recovery of faults, it might not be possible to wait until all observations have been made. This can be checked by performing the diagnosability analysis for subsets of observables. E.g., for the partition $\vec{o}_A$, $\vec{o}_B$ of $\vec{o}$ we check whether the following is satisfiable:

$$\psi(\vec{o}_A \cup \vec{o}_{B1}, \vec{u}_1, m, \vec{h}_1) \wedge \psi(\vec{o}_A \cup \vec{o}_{B2}, \vec{u}_2, m, \vec{h}_2) \wedge \beta(\vec{u}_1) \wedge \neg\beta(\vec{u}_2) \wedge Healthy(\vec{h}_1, \vec{h}_2)$$

Intuitively, we ask that the two systems are observationally equivalent only for a given subset of the observable discrepancies ($\vec{o}_A$). If no critical pair exists in this case, we do not need to wait for the other observation to understand whether the state condition occurred. This technique can also be used to search for sets of sensors that are sufficient to guarantee diagnosability (Bittner et al. 2012).

6 Experimental Evaluation

We implemented a prototype tool using the techniques described in this paper. The tool is able to generate partial traces for a TFPG, test possibility and necessity of arbitrary conditions, check refinement, and perform diagnosis and diagnosability. Due to the lack of publicly available TFPGs, we evaluated the scalability of the approach on a benchmark of randomly generated TFPGs [1].

[1]Prototype tool and benchmarks are available at `http://es.fbk.eu/people/gario/aaai15/`

We run two sets of experiments: refinement and diagnosability. For the refinement benchmarks we take a TFPG and derive a positive and a negative refinement instance. We then ask the solver to verify whether those instances are refinements of the original TFPG. The transformation for the positive case requires picking a node of the TFPG, removing it and reconnecting the predecessors with the successors in a suitable way. Negative examples are obtained in a similar way, but we modify the propagation intervals to make them incompatible with the original ones. Since the check of refinement involves a quantifier alternation, we optimize the formula by inlining whenever possible. For the diagnosability problem, we test diagnosability of each failure mode in the TFPG, assuming that all sensors are healthy. We run our experiments using Z3 (de Moura and Bjørner 2008) as SMT Solver on an Intel i7 2.93GHz, using a time-out of 300 seconds and memory-out of 2GB for all experiments. The left plot of Figure 3 shows the runtime of the refinement testing when we increase the size of the TFPG, while the right plot shows the runtime for the diagnosability testing. The runtime of the diagnosability check increases quickly, since we are using the twin-plant construction: for every additional node in the TFPG we add four new variables to the problem.

The industrial TFPGs, to which we have access, are trivially analyzed by our approach. In the literature (Hayden et al. 2006), TFPGs with 400 nodes are considered medium size, having more than 1000 nodes is uncommon. Therefore, our benchmarks consider examples that are reasonably bigger than commonly developed TFPGs. These experiments show that we are able to analyze huge TFPGs in a reasonable time; hence, we believe that these techniques could be integrated in the design process loop, providing quick feedback to the designer. The choice of using the number of nodes of the TFPG as indicator of its complexity is justified by previous work (Abdelwahed, Karsai, and Biswas 2003) where the algorithmic complexity of the reasoning algorithm is defined in terms of the number of nodes. However, other factors might have an impact on the complexity of a TFPG, and finding a good metric is an open research question.

7 Conclusion

We presented a novel characterization of TFPGs based on symbolic techniques. We explored several important reasoning tasks that aim at increasing the confidence of the designer in the TFPG model, thus guaranteeing that the online reasoning tasks (e.g., diagnosis) can be effective.

Our framework provides a way to describe and perform model validation, refinement testing, diagnosis and diagnosability in a unified way. Other reasoning tasks can be defined in the future by relying on the SMT solvers to perform the reasoning, without implementing ad-hoc reasoning algorithms. Finally, we experimentally show that these techniques are applicable on TFPGs of considerable size.

As future work, we are going to explore ways to generate a TFPG from a given system model and perform verification of a TFPG against it. We would also like to integrate our validation techniques within existing TFPG reasoning tools, e.g. the FACT tool-set.

References

Abdelwahed, S.; Karsai, G.; Mahadevan, N.; and Ofsthun, S. C. 2009. Practical implementation of diagnosis systems using timed failure propagation graph models. *Instrumentation and Measurement, IEEE Transactions on* 58(2):240–247.

Abdelwahed, S.; Karsai, G.; and Biswas, G. 2003. System diagnosis using hybrid failure propagation graphs. Technical Report ISIS-02-302, Vanderbilt University.

Atlas, L.; Bloor, G.; Brotherton, T.; Howard, L.; Jaw, L.; Kacprzynski, G.; Karsai, G.; Mackey, R.; Mesick, J.; Reuter, R.; and Roemer, M. 2001. An evolvable tri-reasoner IVHM system. In *2001 IEEE Aerospace Conference, 11.0307*.

Barrett, C. W.; Sebastiani, R.; Seshia, S. A.; and Tinelli, C. 2009. Satisfiability modulo theories. In *Handbook of Satisfiability*. IOS Press. 825–885.

Beyer, D.; Cimatti, A.; Griggio, A.; Keremoglu, M. E.; and Sebastiani, R. 2009. Software model checking via large-block encoding. In *FMCAD*, 25–32.

Bittner, B.; Bozzano, M.; Cimatti, A.; and Olive, X. 2012. Symbolic synthesis of observability requirements for diagnosability. In *AAAI-12*.

Bittner, B.; Bozzano, M.; Cimatti, A.; de Ferluc, R.; Gario, M.; Guiotto, A.; and Yushtein, Y. 2014. An Integrated Process for FDIR Design in Aerospace. In *Proc. IMBSA 2014*.

Bozzano, M.; Cimatti, A.; Gario, M.; and Tonetta, S. 2014. Formal design of fault detection and identification components using temporal epistemic logic. In Ábrahám, E., and Havelund, K., eds., *Proceedings of TACAS'14*, volume 8413 of *Lecture Notes in Computer Science*, 326–340. Grenoble, France: Springer.

Cimatti, A.; Micheli, A.; and Roveri, M. 2014. Solving strong controllability of temporal problems with uncertainty using SMT. *Constraints* 1–29.

de Moura, L., and Bjørner, N. 2008. Z3: An efficient SMT solver. In *TACAS*, 337–340.

Dubey, A.; Karsai, G.; and Mahadevan, N. 2010. Fault-adaptivity in hard real-time component-based software systems. In *Software Engineering for Self-Adaptive Systems II - International Seminar, Dagstuhl Castle, Germany, October 24-29, 2010 Revised Selected and Invited Papers*, 294–323.

Dubey, A.; Karsai, G.; and Mahadevan, N. 2011. Model-based software health management for real-time systems. In *Aerospace Conference, 2011 IEEE*, 1–18.

European Space Agency. 2011. Statement of Work: FDIR Development and Verification & Validation Process. Appendix to ESTEC ITT AO/1-6992/11/NL/JK.

European Space Agency. 2012. Statement of Work: Hardware-Software Dependability for Launchers. Appendix to ESTEC ITT AO/1-7263/12/NL/AK.

Hayden, S.; Oza, N.; Mah, R.; Mackey, R.; Narasimhan, S.; Karsai, G.; Poll, S.; Deb, S.; and Shirley, M. 2006. Diagnostic Technology Evaluation Report For On-Board Crew Launch Vehicle. (September).

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

Karsai, G.; Abdelwahed, S.; and Biswas, G. 2003. Integrated diagnosis and control for hybrid dynamic systems. In *AIAA Guidance, Navigation and Control Conference*.

Karsai, G. 2013. Fact: Fault adaptive control technology tools for developing fault adaptive systems. Tool presentation at `https://fact.isis.vanderbilt.edu/FACT-Sum.pdf`.

Kleene, S. 1967. *Mathematical Logic*. J. Wiley & Sons.

Ofsthun, S., and Abdelwahed, S. 2007. Practical applications of timed failure propagation graphs for vehicle diagnosis. In *Autotestcon, 2007 IEEE*, 250–259.

Reiter, R. 1987. A theory of diagnosis from first principles. *Artificial intelligence* 32(1):57–95.

Sampath, M.; Sengupta, R.; Lafortune, S.; Sinnamohideen, K.; and Tneketzis, D. 1995. Diagnosibility of Discrete-Event Systems. *IEEE Transactions on automatic control* 40(9):1555–1576.

Srivastava, A. N., and Han, J., eds. 2012. *Machine learning and knowledge discovery for engineering systems health management*. Chapman & Hall/CRC data mining and knowledge discovery series. Boca Raton, FL: CRC Press.

Strasser, S., and Sheppard, J. 2011. Diagnostic alarm sequence maturation in timed failure propagation graphs. In *Autotestcon*.

Vesely, W.; Stamatelatos, M.; Dugan, J.; Fragola, J.; III, J. M.; and Railsback, J. 2002. Fault tree handbook with aerospace applications. *Technical report, NASA*.