

An improved algorithm for the design of testable subsystems

Stéphane Ploix Abed Alrahim Yassine Jean-Marie Flaus

*Laboratoire des sciences pour la conception, l'optimisation et la production, G-SCOP
BP 46, Saint Martin d'Herès 38402, France
Stephane.Ploix@g-scop.inpg.fr, Abed-Alrahim.yassine@g-scop.inpg.fr, Jean-marie.Flaus@g-scop.inpg.fr*

Abstract: In complex industrial plants, there are usually many sensors and the modeling of plants leads to lots of mathematical relations. Before using classical tools for fault detection, the first problem to solve is: what sensors and mathematical relations should be selected for the design of a detection test such as a state observer or a parity space based detection algorithms. This paper presents a general method for finding all the possible testable subsystems i.e. sets of relations, than can lead to detection tests, taking into account actuator and sensor locations. This method, which is based on a structural analysis, provides the constraints that have to be used for the design of each detection test and manages situations where constraints contain non deductible variables. Thanks to these results, it becomes possible to select the most interesting testable subsystems regarding detectability and diagnosability criteria.

Keywords: Automatic test design, Structural approach, Fault diagnosis, Analytical redundancy relations

1. INTRODUCTION

Generally speaking, diagnostic analyzes of physical systems rely on detection tests that provide symptoms. In the scientific literature, there are two main trends for the design of tests. The first one comes from the Artificial Intelligence community: Reiter [1987], De Kleer and Williams [1987], Pulido and Alonso [2002]. It relies on component-based approaches. The principle is to model the different components with relations, also named constraints, and to directly check the consistency between the models and the data. This approach raises a practical problem: in the industrial world, even if the constraints can be modeled, the tests usually do not result from simple consistency tests between constraints and data, especially in dynamic systems modelled by differential equations. Usually, constraints model only a part of the knowledge: noise and modeling uncertainties are not a priori taken into account in the constraints but only a posteriori. Consequently, consistency tests do not result only from formalized knowledge: an engineer usually only needs to know the sets of constraints that can lead to tests before designing and validating test algorithms. The second stream coming from the Fault Detection and Isolation community, Patton et al. [1989], Frank [1990], uses a complete model, usually a state space representation, of the system to be diagnosed. It is often called a structured or robust approach because it aims at projecting models in different subspaces in order to distinguish the different faults that may occur (and to remove uncertain parts). These approaches raise another issue: projections do not trace the components that are involved into tests. It is therefore difficult to interpret

the symptoms, especially with formal analysis such as in Nyberg and Krysander [2003], Ploix et al. [2003].

Complex industrial systems can be depicted by various kinds of relations, also named constraints, depending on the modeling approach: differential equations, qualitative relations, and rules expressed in a natural language may be encountered. Methods that can manage both complexity and models of different types are necessary. This fact necessitates the use of structural approaches and rules out approaches based on state space representations or on Grobner bases: Frisk [2000]. In diagnosis, structural modeling has been introduced in Davis [1984]. Using a semantic theory of abstraction, Chittaro and Ranon [2004] has pointed out that structural modeling is an abstraction of behavioral modeling. Then, discovering testable sets of constraints can be achieved thanks to a procedure based on a structural model such as the bipartite graph approach proposed in Blanke et al. [2003], Staroswiecki and Declerck [1989], Pulido and Alonso [2002]. Finding testable sets of constraints may indeed be done thanks to an elimination procedure that combines constraints in order to eliminate all the unknown physical variables and therefore getting constraint containing only known data i.e. testable constraints. However, this approach leads to a high level of complexity when searching all the testable sets of constraints. In Travé-Massuyès et al. [2006], an alternative method has been proposed but the level of complexity is still high. Krysander et al. [2005] has proposed an improved algorithm but it does not manage constraints containing no deductible variables.

This paper presents a general method that provides testable subsystems. A new algorithm, which improves

Ploix et al. [2005] and is more general than Krysanter et al. [2005], based on a join-operator coming from relational algebra is proposed. It relies also on a structural abstraction of the constraints and it traces all the constraints that are involved in testable subsystems thus making it possible to determine what physical components are examined by each test.

2. PROBLEM STATEMENT

This section introduces the concepts and the formalism used in the paper. In order to manage testable subsystem design, let us introduce a formalism for behavioral modeling before considering structural modeling.

2.1 Behavioral modeling

Behavioral knowledge starts with phenomena. A phenomenon is a potentially observable element of information about the actual state of a system. It is modeled by an implicitly time-varying variable, which has to be distinguished from a parameter that is model-dependent. Generally speaking, even if a phenomenon is observable, it is not possible to merge it with data because in fault diagnosis, data are only known provided that some actuators or sensors behave properly. Phenomena $V(t) = \{\dots, v_i(t), \dots\}$ are modeled by a phenomenological space $\mathcal{F}(T, V) = \{V(t); t \in T\}$ where T stands for continuous or discrete sets of time. At any given time t in T , these phenomena belong to a domain $dom(t, V) = dom(V(t))$ representing all the possible values that may have phenomena. Consequently, when considering all $t \in T$, $\{dom(V(t)); t \in T\}$ represents a tube in the phenomenological space $\mathcal{F}(T, V)$. A *data flow* models data provided by a source of information concerning a phenomenon. A data flow concerning a phenomenon v is denoted: $obs(t, v)$ with $obs(t, v) \in dom(v(t))$. It corresponds to a trajectory belonging to the tube $\{dom(v(t)); t \in T\}$. When information about v is coming from different sources, the different data flows can be denoted: $obs_i(t, v)$. Formally, a data flow provided by a component c can be linked to a phenomenon: $ok(c) \rightarrow \forall t \in T, obs(t, v) = v$, which means that if the component named c is in the state ok then the data $obs(t, v)$ corresponds to the actual value of the phenomenon v at any time $t \in T$. For testable subsystem design, it should be noted that all the phenomena have to be considered as unknown. Therefore, a mapping is testable if it does not contain any variable standing for phenomena but only data flows.

Let us now define the concept of testable subsystem (TSS) and its relationship with the concept of ARR. In the following, the set of variables appearing in a constraint k is denoted: $var(k)$ and the set of variables appearing in the set of constraints K : $var(K) = \bigcup_{k \in K} var(k)$.

Definition 1. Let K be a set of constraints and v a variable in $var(K)$ characterized by its domain $dom(v)$. K is a solving constraint set for v if using K , it is possible to find a value set S for v such that $S \subset dom(v)$. A solving constraint set for v is minimal if there is no subset of K , which is also a solving constraint set for v . A minimal solving constraint set K for v is denoted: $K \vdash v$

Definition 2. Let K be a set of constraints. K is testable if and only if there are two distinct subsets $K_1 \subset K$,

$K_2 \subset K$ such that $K_1 \not\subseteq K_2$ and $K_2 \not\subseteq K_1$, and a variable $v \in var(K)$ such that $K_1 \vdash v$ and $K_2 \vdash v$. If this property is satisfied, it is indeed possible to check if the value set S_1 deduced from K_1 is consistent with the value set S_2 deduced from K_2 : $S_1 \cap S_2 \neq \emptyset$.

This definition also applies to models containing ordinary differential equations. Indeed, testable state space representations, including state space observers, always have equivalent parity space representations Staroswiecki et al. [1991]. Adding any constraint to a testable set leads also to a testable set of constraints. Only minimal testable sets are interesting.

Definition 3. A testable set of constraints is minimal if it is not possible to keep testability when removing a constraint.

A global testable constraint that can be deduced from a TSS is called ARR.

2.2 Structural modeling

Generating testable subsystems means that cause-effect relationships modelled by mappings, defined from the domain of a set of variables to another domain of other variables, must be combined so that no more variables remain. A general method must abstract mappings away as they may be of very different types, ranging from automata to differential equations or qualitative modeling.

This is why structural modeling has to be considered although this advantage is offset by the fact that results may overestimate the effective testable subsystems because knowledge about mappings is not fully taken into account. Overestimation leads to testable subsystems that refer to many more combined constraints than necessary. The consequence is that, during test implementation, verification is needed to see whether all the constraints involved in a testable subsystem are effectively required.

As an abstraction of a mapping, the notion of *structure of constraint* is introduced: basically, the *structure of a constraint* \mathcal{K} corresponds to the variables $var(\mathcal{K})$. Nevertheless, a given constraint may be modeled by different mappings.

Firstly, although mappings to multidimensional spaces could be used, they are difficult to manage in testable subsystem design. It is better to break them down into equivalent sets of 1-dimensional mappings. In the following, 1-dimensional mappings modeling a constraint \mathcal{K} are named *realization* of \mathcal{K} .

Moreover, several realizations of a constraint may be equivalent. Let κ_i be a realization from $V \setminus \{v\}$ to $\{v\}$. There may be equivalent realizations defined on V that also model the constraint. Therefore, the notion of *structure of a constraint* can be extended to represent all the equivalent realizations representing a given constraint \mathcal{K} . For instance, consider a realization κ modeling a *logical xor*: $x_3 = x_1 \otimes x_2$. There is a mapping leading to x_3 from x_1 and x_2 but also a mapping from x_2 and x_3 leading to x_1 and another one to x_2 . In that case, these variables can be qualified as deductible or, when it is meaningful, as calculable in reference to Iwasaki and Simo

[1994]. This is denoted; $var(\kappa) = var^+(\kappa) = \{x_1, x_2, x_3\}$. It is different for a realization κ' modeling a *logical or*: $x_3 = x_1 \vee x_2$ where only x_3 is deductible. This is denoted: $var(\kappa) = var^+(\kappa') \cup var^-(\kappa')$ with $var^+(\kappa) = \{x_3\}$ and $var^-(\kappa) = \{x_1, x_2\}$. The set of equivalent realizations modeling a constraint \mathcal{K} is denoted: $K(\mathcal{K})$. Therefore, the structure of a constraint models at one and the same time the variables of its phenomenological space, and a set of possible equivalent 1-dimensional mappings that model this constraint. A structure may represent either a particular realization or a set of equivalent realizations.

The variables $V = var(\mathcal{K})$ can be broken down into a set of deductible variables and a set of non-deductible variables. A structure s will be written $\perp V^-, V^+ \lrcorner$ where V^- and V^+ satisfy $V^- \cap V^+ = \emptyset$. Therefore, the structure modeling a constraint \mathcal{K} is denoted: $s(\mathcal{K}) = \perp V^-, V^+ \lrcorner$. Because $\forall v \in V^+$, there is a realization from $K(\mathcal{K})$ leading to v :

$$\forall v \in V^+, \exists \perp (V^- \cup V^+) \setminus \{v\}, \{v\} \lrcorner$$

For the sake of simplicity, the following notations have been adopted: $\perp \emptyset, V^+ \lrcorner = \perp V^+ \lrcorner$ and, if S is a set of structures, $var(S) = \bigcup_{s \in S} var(s)$. Finally, an empty structure s is a structure satisfying: $var(s) = \emptyset$. It is denoted: $s = \perp \lrcorner$.

For the design of testable subsystem, some structures are particularly useful because they model what it is known in a system i.e. the controlled or measured variables: they are named terminal structure.

Definition 4. A terminal structure s satisfies $card(var(s)) = 1$. It usually involves a data flow and models the fact that a trajectory can be assigned to a variable. By extension, a constraint containing only one variable is also qualified as terminal.

Because of the possible over-estimations, it is useful to introduce the following definition.

Definition 5. A structure s_1 wraps another structure s_2 if $var(s_1) \supseteq var(s_2)$ and $var^+(s_1) \supseteq var^+(s_2)$. It is denoted: $s_1 \supseteq s_2$.

In order to show that values can be propagated between mappings, i.e. that the intersection of two constraints can be projected without loss, a join operator is defined. As a prerequisite, the notion of propagable variable has to be introduced.

Definition 6. Let s_1 and s_2 be two structures related to $\kappa_1 \in K(\mathcal{K}_1)$ and $\kappa_2 \in K(\mathcal{K}_2)$. The propagation of a variable v between $s_1 = s_1(\kappa_1)$ and $s_2 = s_2(\kappa_2)$ is possible only if $v \in var(s_1) \cap var(s_2)$ and if v is deductible in at least one structure. If this condition is satisfied, v is qualified as propagable between s_1 and s_2 . By extension, v is also said *propagable* between κ_1 and κ_2 and also between $K(\mathcal{K}_1)$ and $K(\mathcal{K}_2)$.

The join operator can now be defined.

Definition 7. Let s_1 and s_2 be two structures, with $V_1^+ = var^+(s_1)$, $V_1^- = var^-(s_1)$, $V_2^+ = var^+(s_2)$ and $V_2^- = var^-(s_2)$. The join operator, denoted \bowtie_v , where v is a propagable variable between s_1 and s_2 , is defined only in the following two situations:

- if $v \in V_1^+ \cap V_2^-$ then

$$s_1 \bowtie_v s_2 = \perp (V_1^- \cup V_1^+ \cup V_2^-) \setminus (V_2^+ \cup \{v\}), V_2^+ \lrcorner$$

- if $v \in V_1^+ \cap V_2^+$, then

$$s_1 \bowtie_v s_2 = \perp (V_1^- \cup V_2^-) \setminus (V_1^+ \cup V_2^+), (V_1^+ \cup V_2^+) \setminus \{v\} \lrcorner$$

If a formula $s_1 \bowtie_v s_2$ satisfies one of the two previous points, it is qualified as evaluable.

Using this operator results in a definition of a propagation method i.e. if the value of a variable can be deduced from one realization, then this value can be propagated into the other one. A link is thus created between two constraints: it corresponds partially to the join operator in relational algebra.

Theorem 8. Let K_1 and K_2 be two sets of equivalent realizations. Then a wrapping structure of the set of equivalent realizations resulting from the propagation of a variable v between K_1 and K_2 can be obtained using \bowtie_v operator on $s(K_1)$ and $s(K_2)$.

Proof. Let us denote $s(K_1) = \perp V_1^-, V_1^+ \lrcorner$, $s(K_2) = \perp V_2^-, V_2^+ \lrcorner$ and K the set of equivalent realizations resulting from the propagation of v between K_1 and K_2 . The set K depends on the presence of other propagable variables. If v is the only propagable variable between K_1 and K_2 , the exact structure of K can be deduced, else, only a wrapping structure can be found.

Consider the situation where there is only one propagable variable $v \in V_1^+ \cap V_2^-$. $\exists \kappa_{1,v} \in K_1$ such as $v = \kappa_{1,v}((V_1^+ \setminus \{v\}) \cup V_1^-)$ and v can be propagated into K_2 . If $V_2^+ \neq \emptyset$, $\forall w \in V_2^+$, $\exists \kappa_{2,w} \in K_2/w = \kappa_{2,w}((V_2^+ \setminus \{w\}) \cup V_2^-)$. Because $v \in V_2^-$, it yields that there is κ_w such that:

$$w = \kappa_w(((V_2^+ \setminus \{w\}) \cup (V_2^- \setminus \{v\})) \cup ((V_1^+ \setminus \{v\}) \cup V_1^-))$$

Because v is the only propagable variable, $w \notin (V_1^+ \cup V_1^-)$ and therefore, κ_w is a realization. Because this result is true $\forall w \in V_2^+$, the structure of the resulting set of equivalent realizations is: $\perp (V_1^- \cup V_1^+ \cup V_2^-) \setminus (V_2^+ \cup \{v\}), V_2^+ \lrcorner$. Moreover, if V_2^+ is empty, the previous result remains true: $\perp (V_1^- \cup V_1^+ \cup V_2^-) \setminus \{v\}, \emptyset \lrcorner$. It yields: $\forall v \in V_1^+ \cap V_2^-$, $s(K) = s(K_1) \bowtie_v s(K_2)$.

If the unique propagable variable satisfies $v \in V_1^+ \cup V_2^+$, additional deductible variables can be found. Indeed,

- if $V_2^+ \setminus \{v\} \neq \emptyset$, $\exists \kappa_{1,v} \in K_1$ such that $v = \kappa_{1,v}((V_1^+ \setminus \{v\}) \cup V_1^-)$ and v can be propagated into K_2 : $\forall w \in V_2^+ \setminus \{v\}$, $\exists \kappa_{2,w} \in K_2/w = \kappa_{2,w}((V_2^+ \setminus \{w\}) \cup V_2^-)$. Because $v \in V_2^+$, it yields: $\exists \kappa_w/w = \kappa_w(((V_2^+ \cup V_1^+) \setminus \{v, w\}) \cup V_1^- \cup V_2^-)$
- if $V_1^+ \setminus \{v\} \neq \emptyset$, $\exists \kappa_{2,v} \in K_2$ such that $v = \kappa_{2,v}((V_2^+ \setminus \{v\}) \cup V_2^-)$ and v can be propagated into K_1 : $\forall w \in V_1^+ \setminus \{v\}$, $\exists \kappa_{1,w} \in K_1/w = \kappa_{1,w}((V_1^+ \setminus \{w\}) \cup V_1^-)$. Because $v \in V_1^+$, it yields: $\exists \kappa_w/w = \kappa_w(((V_1^+ \cup V_2^+) \setminus \{v, w\}) \cup V_1^- \cup V_2^-)$

Because v is the only propagable variable, it yields κ_w are realizations and then that the structure of the resulting constraint is given by: $\perp (V_1^- \cup V_2^-) \setminus (V_1^+ \cup V_2^+), (V_1^+ \cup V_2^+) \setminus \{v\} \lrcorner$. Consequently, $\forall v \in V_1^+ \cap V_2^+$, $s(K) =$

$s(K_1) \bowtie_v s(K_2)$. These results remain true if V_1^+ or V_2^+ are empty.

When there are several propagable variables, it is no longer possible to prove that functions κ_w are realizations and hence the previous results become:

- $\forall v \in V_1^+ \cap V_2^-, s(K) \subseteq s(K_1) \bowtie_v s(K_2)$
- $\forall v \in V_1^+ \cap V_2^+, s(K) \subseteq s(K_1) \bowtie_v s(K_2)$

Remark 9. According to the definition, the join operator is not commutative nor associative. Indeed, consider 3 structures s_1, s_2 and s_3 satisfying $v_2 \in \text{var}^+(s_1) \cap \text{var}^-(s_3), v_2 \notin \text{var}(s_2), v_1 \in \text{var}^+(s_1) \cap \text{var}^+(s_2)$. Then, the formula $(s_1 \bowtie_{v_1} s_2) \bowtie_{v_2} s_3$ can be evaluated whereas $s_2 \bowtie_{v_2} s_3$ cannot. To show that the join-operator is not associative, $(s_1 \bowtie_{v_1} s_2) \bowtie_{v_2} s_3 \neq s_1 \bowtie_{v_1} (s_2 \bowtie_{v_2} s_3)$.

This section introduced the structures of constraints and a join operator that models value propagations between structures. These tools can then be used to design testable subsystems containing many different kinds of mappings. The way to use these tools is described in the next section.

3. FINDING TESTABLE SUBSYSTEMS

Some basic concepts have first to be introduced before tackling the design of testable subsystems. Then, some particular modeling contexts are examined.

3.1 Combining structures into formulae

A test results from consecutive propagations that can be modeled by a propagation formula: $f = (((s_1 \bowtie_{v_1} s_2) \bowtie_{v_2} s_3) \bowtie_{v_3} (s_4 \bowtie_{v_4} s_2)) \dots$ where s_i are structures and v_j are the variables to be propagated. A formula is composed of subformulae linked to the join operator, where the elementary formulae are structures. Thanks to this operator, a propagation formula f can be evaluated as a structure $s(f)$. The set of all the formulae is denoted \mathcal{F} .

Definition 10. A formula is qualified as evaluable if all its subformulae are evaluable according definition 7.

Definition 11. The support of a formula $f \in \mathcal{F}$, denoted $\sigma(f)$, is the set of all the structures involved in the formula.

Definition 12. The degree of a formula $f \in \mathcal{F}$, denoted $d(f)$, is equal to the number of times the join operator appears in the formula: it represents the number of elementary propagations.

Definition 13. Two formulae f_1 and f_2 are comparable if $\sigma(f_1) = \sigma(f_2)$ and if $\text{var}(s(f_1)) = \text{var}(s(f_2))$ i.e. they have the same constraints and the same variables. This is denoted: $f_1 \sim f_2$.

Moreover, during propagations, a given variable can be instantiated only once. However, in some formulae, a variable can appear several times and then be instantiated in different ways. In this situation, there will be a simpler formula where the variable has been instantiated only once.

Definition 14. A formula f_1 is simpler than a formula f_2 if:

- $\sigma(f_1) \subset \sigma(f_2)$ and $\text{var}(s(f_1)) \subset \text{var}(s(f_2))$

- or if $f_1 \sim f_2$ and $d(f_1) < d(f_2)$
- or if $f_1 \sim f_2$ and $d(f_1) = d(f_2)$ and $\text{var}^+(s(f_1)) \supset \text{var}^+(s(f_2))$

It is denoted: $f_1 \prec f_2$. It is said that f_2 overestimates f_1 .

A testable propagation formula $f \in \mathcal{F}$ is an evaluable formula that leads to an empty structure. A testable propagation formula is minimal over a set of structures S if there is no simpler formula over S . It is called the minimal testable propagation formula (MTPF) over S .

According to the definition of the join operator, all the formulae correspond to wrapping structures of constraints that can be found by combining elementary constraints K of a diagnostic problem. Therefore, the formulae may overestimate the propagations that lead to a test. Fortunately, it is easy to check if all the constraints related to the support $\sigma(f)$ of an MTPF f have been used during the design of a test.

3.2 Algorithms for finding testable subsystems

For the design of testable subsystems (TSS), all the possible MTPF have to be found because TSS are given by the support of MTPF. The principle is to iteratively propagate values until MTPF are found. But, unlike value propagation, a propagation can be envisaged even if related variables have not been instantiated. In order to reduce the computations, the propagations related to a variable that involve the fewest structures are achieved first.

Sets of formulae are represented by the letter F and the corresponding structures by $s(F) = \{s(f); f \in F\}$. Consider a set of constraints characterizing a diagnostic problem and F_0 , the corresponding structures, which are also elementary formulae. $s(F_0)$ denotes the structures corresponding to F_0 . The number of structures from a set $s(F)$ where a variable v appears, is named the order of v in $\sigma(F)$. It is denoted: $o_F(v)$

Firstly, a propagation cannot be achieved when a variable appears only once in the structures $s(F_0)$. Therefore, structures containing these variables, and its corresponding formula, should be removed because they have no usefulness in an MTPF. Nevertheless, when some structures are removed, it is possible to find new variables that only appear once. The procedure is then repeated until no more single-occurrence variables remain. This step is a clearing step. It is summarized by algorithm 2.

Algorithm 1 Remove useless structures

Require: F_0 , a set of formulae

$F \leftarrow F_0$

repeat

$V \leftarrow \{v \in \text{var}(s(F)); o_F(v) = 1\}$

$F \leftarrow \{f \in F; \text{var}(s(f)) \cap V = \emptyset\}$

until $V = \emptyset$

return F

The resulting cleared set is named F_1 . The propagations can now be achieved according to the orders of variables. The variables of lowest order are selected first. Let v be one of these variables. All the formulae where v appears are selected and, using the join operator, new evaluable formulae are then deduced and added to the current set of

formulae. Formulae that overestimate others are removed as well as formulae that contain v . This procedure is repeated until all the variables have been considered. Remaining structures are then empty structures and thus all the MTPF have been found. The procedure is summarized by the following algorithm.

Algorithm 2 Compute MTPF

Require: F_1 , a set of formulae
 $F \leftarrow F_1$
while $\text{var}(s(F)) \neq \emptyset$ **do**
 select $v \in \text{var}(s(F))$ **such that** $o_F(v) \leq o_F(v_i), \forall v_i \in \text{var}(s(F))$
 $F' \leftarrow \{f/v \in \text{var}(s(f))\}$
 $F'' \leftarrow \{f_i \bowtie_v f_j; (f_i, f_j) \in F'^2, i \neq j, f_i \bowtie_v f_j \text{ evaluable}\}$
 $F \leftarrow (F \setminus F') \cup F''$
 $F \leftarrow F \setminus \{f \in F; \exists f_i \in F, f_i \neq f, f \succ f_i\}$
end while
return F

Sometimes, because the number of testable propagation formulae is very high, it is quicker to find only a subset of all the MTPF. In order to reduce to number of propagations but also to check all the constraints, propagations of variables that are known because they have been measured or controlled, can be reduced: propagations may indeed be stopped when a known variable is found. The resulting MTPF are called basic MTPF. They are evaluable because basic MTPF are a small subset of all the MTPF that covers all the exploitable structures present in all the MTPF. Algorithm 2 can still be used but when a variable v is involved in a terminal structure, the join operator \bowtie_v is only applied, on the one hand, between the terminal structures and the other structures involving v , and on the other hand, between the terminal structures themselves if many of them include the variable v : it corresponds to the so-called material redundancy.

3.3 Particular modeling contexts

In this section, two particular contexts are examined: the dynamical systems and the systems with branchings.

Dynamical systems Generally speaking, a model is said to be dynamic if either:

- a variable appears several times in a system but at different timestamps
- or a variable and some of its derivatives or summations (whatever the order is) appear in the system.

The first case mainly concerns time-delays and discrete time recurrent systems. According to section 2, each variable stands for a tube in a phenomenological space. Therefore a time delay, modelled by $y(t + \Delta) = x(t)$, is a constraint that establishes a link between two tubes: $\{dom(y(t + \Delta)); \forall t\}$ and $\{dom(x(t)); \forall t\}$. Therefore, even if the two variables models the same phenomenon, in the structural model, they cannot be merged. Consider now the following discrete-time recurrent model:

$$\begin{aligned} x((k+1)T_e) &= Ax(kT_e) + Bu_k(kT_e) \\ y(kT_e) &= Cx(kT_e) \end{aligned} \quad ; k \in \mathbb{N}$$

where T_e stands for the sampling period.

The phenomenon modelled by x appears twice. Therefore, the constraint must be implicitly completed by a time delay between variables $x((k+1)T_e)$ and $x(kT_e)$. Structurally speaking, these constraints are modelled by the following structures:

$$\begin{aligned} \lrcorner \{x(kT_e), x((k+1)T_e), u(kT_e)\} \lrcorner \\ \lrcorner \{x(kT_e), x((k+1)T_e)\} \lrcorner \\ \lrcorner \{x(kT_e), y(kT_e)\} \lrcorner \end{aligned}$$

Moreover, if the tube corresponding to $x((k+1)T_e)$ appears only in this constraints, the join operator $\bowtie_{x((k+1)T_e)}$ can be applied between $\lrcorner \{x(kT_e), x((k+1)T_e), u(kT_e)\} \lrcorner$ and $\lrcorner \{x(kT_e), x((k+1)T_e)\} \lrcorner$ and it results:

$$\begin{aligned} \lrcorner \{x(kT_e), u(kT_e)\} \lrcorner \\ \lrcorner \{x(kT_e), y(kT_e)\} \lrcorner \end{aligned}$$

The second case mainly concerns integrations and differential equations. Consider for instance the following model: $\frac{dx}{dt} = u$. $\frac{dx}{dt}$ corresponds to a tube, which can be connected to x in adding the implicit constraint: $x = \int \frac{dx}{dt} dt$. The initial condition could also be taken into account by considering $x = \int_0^{t_f} \frac{dx}{dt} dt + x_0$. In this case, the structures become: $\lrcorner \{\frac{dx}{dt}, u\} \lrcorner$ and $\lrcorner \{x, \frac{dx}{dt}, x_0\} \lrcorner$. In the same way as time-delays, the join operator $\bowtie_{\frac{dx}{dt}}$ can be applied to obtain the following structure: $\lrcorner \{u, x\} \lrcorner$ or, if the initial condition is considered, $\lrcorner \{u, x, x_0\} \lrcorner$. This result remains true for summations and derivatives of any order. Let us now consider an ordinary differential equation:

$$\begin{aligned} \frac{dx}{dt} &= Ax + Bu \\ y &= Cx \end{aligned}$$

Here also, an implicit constraint has to be added: $x = \int \frac{dx}{dt} dt$. The following structures arise:

$$\begin{aligned} \lrcorner \{x, \frac{dx}{dt}, u\} \lrcorner \\ \lrcorner \{x, \frac{dx}{dt}\} \lrcorner \\ \lrcorner \{x, y\} \lrcorner \end{aligned}$$

Using the join operator $\bowtie_{\frac{dx}{dt}}$, it yields:

$$\begin{aligned} \lrcorner \{x, u\} \lrcorner \\ \lrcorner \{x, y\} \lrcorner \end{aligned}$$

Systems with branchings Using a structural approach for the design of testable subsystems leads to a general method that may potentially be applied to any kind of systems. The case of dynamical systems have been examined but another context requires also a special attention: the systems with branchings. These systems can be managed by the method presented in section 3.2 but the amount of computations can be drastically reduced. Indeed, consider for instance the roads in figure 1. Denoting R_{XY} a road section linking X to Y , a structural model of the road network is given by:

$$\begin{aligned} s(R_{AB_1}) &= \lrcorner A, B_1 \lrcorner \\ s(R_{B_2C}) &= \lrcorner B_2, C \lrcorner \\ s(R_{B_3D}) &= \lrcorner B_3, D \lrcorner \\ s(\text{crossroad}) &= \lrcorner B_1, B_2 \lrcorner \\ s(\text{crossroad}) &= \lrcorner B_1, B_3 \lrcorner \\ s(\text{crossroad}) &= \lrcorner B_2, B_3 \lrcorner \end{aligned}$$

Generally speaking, if a car goes from A to C , it is improbable that its route could also pass by B_3 . Then, a testable subsystem gathering all the constraints of the crossroad would not make sense. If it is not taken into account, the number of generated TSS will be very high. In that simple case, unrealistic routes would appear: (A, B_1, B_2, B_3, D) , (A, B_1, B_3, B_2, C) , $(C, B_2, B_1, B_3, D), \dots$ Imagine now a complete network with many crossroads. Unrealistic combinations of constraints will be recombined and it will provide new unrealistic constraints and so on until a possibly huge number of unrealistic TSS is obtained. In order to avoid this phenomena, junctions have to be taken into account during the generation of formulae.

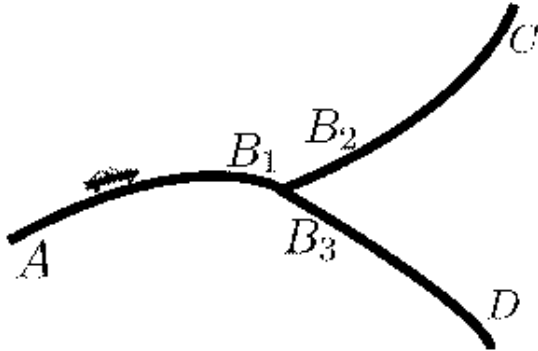


Fig. 1. A road network is a system with branchings

In order to avoid unrealistic TSS, a set of exclusions can be defined. In order to model that only two constraints of the crossroad can be gathered in one formula, the following exclusion set can be defined: $\{\lrcorner B_1, B_2 \lrcorner, \lrcorner B_1, B_3 \lrcorner, B_2, B_3 \lrcorner\}$. It means that all these structures cannot appear in the same formula. Generally speaking, a set of exclusion sets, denoted \mathbb{X} , can be defined. Thus, during the computations of formulae (section 3.2), each time a new formula is found, it is examined in order to determine if it does not gather all the structures belonging to an exclusion set of \mathbb{X} . In that case, the new formula is removed from F'' in algorithm 2.

Systems with branchings are not rare: they can be encountered in Discrete Event Systems with conveyors, transportation networks but also in diagnosis of cognitive skills Ploix et al. [2004] where different alternatives in reasoning may be adopted.

4. CONCLUSION

This paper formalizes structural modeling and shows how it can support engineers in computing detection tests for the system to be diagnosed. The proposed methods provides the constraints to be used to design each test, but also a way of combining one each other. It then let the engineer free to choose its preferred detection test (for instance, parity relations, Luenberger state observers or Kalman filters in dynamical continuous time systems), the way of a posteriori tuning the detection tests in order to take into account modeling uncertainties.

The main advantage of structural modeling is that it makes it possible to handle any kind of systems e.g. dynamical continuous-time, discrete event or rule-based systems. In the paper, it has been shown on a road network that, even if the behavioral constraints are not

available, it remains possible to compute the composition of all the possible ARR. The proposed procedure has been designed in order to reduce as much as possible the number of computations. It has been compared to two alternative approaches and significant improvements have been pointed out.

REFERENCES

- M. Blanke, M. Kinnaert, and M. Staroswiecki. *Diagnosis and fault tolerant control*. Springer, 2003.
- L. Chittaro and R. Ranon. Hierarchical model-based diagnosis based on structural abstraction. *Artificial Intelligence*, 1-2:147–182, 2004.
- R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.
- J. De Kleer and B. C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32:97–130, 1987.
- P. M. Frank. Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy - a survey and some new results. *Automatica*, 26(3):459–471, 1990.
- E. Frisk. Residual generator for non-linear polynomial systems - a grobner basis approach. In *IFAC Fault Detection, Supervision and Safety for Technical Processes*, 2000.
- Y. Iwasaki and H. A. Simo. Causality and model abstraction. *Artificial Intelligence*, 67:143–194, 1994.
- M. Krysander, J. Aslund, and M. Nyberg. An efficient algorithm for finding over-constrained sub-systems for construction of diagnostic tests. In *16th International Workshop on Principles of Diagnosis (DX-05)*, 2005.
- M. Nyberg and M. Krysander. Combining ai, fdi, and statistical hypothesis-testing in a framework for diagnosis. In *IFAC Safeprocess'03*, Washington, U.S.A., 2003.
- R. Patton, P. Frank, and R. Clark (Eds). *Fault diagnosis in dynamic systems*. International series in systems and control engineering. Prentice Hall, 1989.
- S. Ploix, S. Touaf, and J. M. Flaus. A logical framework for isolation in fault diagnosis. In *SAFEPROCESS'2003*, Washington D.C., U.S.A., 2003.
- S. Ploix, M. Désinde, and F. Michau. Assessment and diagnosis for virtual reality training. In *CALLIE2004*, Grenoble, France, 2004.
- S. Ploix, M. Desinde, and S. Touaf. Automatic design of detection tests in complex dynamic systems. In *16th IFAC World Congress*, Prague, Czech republic, 2005.
- B. Pulido and C. Alonso. Possible conflicts, arrs, and conflicts. In *13th International Workshop on Principles of Diagnosis (DX02)*, pages 122–128, May 2002.
- R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.
- M. Staroswiecki and P. Declerck. Analytical redundancy in nonlinear interconnected systems by means of structural analysis. In *IFAC AIPAC'89 Symposium*, Nantes, France, 1989.
- M. Staroswiecki, V. Cocquempot, and J. P. Cassar. Observer based and parity space approaches for failure detection and identification. In *IMACS-IFAC International Symposium*, Lille, France, 1991.
- L. Travé-Massuyès, T. Escobet, and X. Olive. Diagnosability analysis based on component supported analytical redundancy relations. *IEEE transactions on Systems, Man, And Cybernetics - Part A: Systems and Humans*, 36(6):1146–1160, November 2006.